numC

1.0

Generated by Doxygen 1.8.13

Contents

1 Bug List

File Matrix.h

No known bugs.

File Matrix_gsl.h

No known bugs.

File Stats.h

No known bugs.

File Utils.h

No known bugs.

2 Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

3 File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

4 Data Structure Documentation

4.1 Matrix Class Reference

This is Matrix class.

```
#include <Matrix_gsl.h>
```

Public Member Functions

- **Matrix** (std::string name="")
- **Matrix** (int n, int m, std::string name="")
- **Matrix** (const Matrix &A)
- void **init** (int, int, std::string name="New")
- Matrix & **operator=** (const Matrix &A)
- Matrix **operator%** (const Matrix &A)
- Matrix **operator+** (const Matrix &A)
- Matrix **operator-** (const Matrix &A)
- Matrix & **operator∗=** (double a)

- Matrix **operator/=** (double a)
- void **printM** (std::string="matrix")
- Matrix **flat** ()
- double **trace** ()
- Matrix **T** ()
- void **asMat** (double $**$)
- Matrix **diag** ()
- Matrix **clip** (double, std::string="upper")
- Matrix **inv** ()
- double **det** ()
- void **eig** ()
- double **min** (std::string="matrix")
- double **max** (std::string="matrix")

Data Fields

- int **shape** [2]
- double $**$ **matrix**
- double $**$ **eigval**
- double $**$ **eigvec**
- std::string **name**

4.1.1 Detailed Description

This is Matrix class.

Matrix class contains double$**$ matrix as a matrix elements containor. So Matrix A can be used with Matrix.h as putting A.matrix in to Matrix.h functions. Afterwards this matrix instance would likely to be private.

Tag: Matrix Vector

Definition at line 31 of file Matrix_gsl.h.

The documentation for this class was generated from the following files:

- Matrix_gsl.h
- Matrix_gsl.C

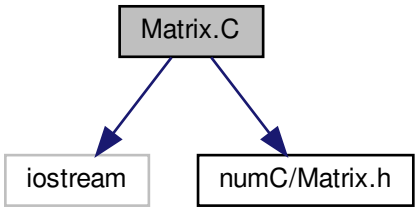
5 File Documentation

5.1 Matrix.C File Reference

Various matrix operations.

```
#include <iostream>
#include <numC/Matrix.h>
```

Include dependency graph for Matrix.C:



Functions

- double ** sum (double **mat1, double **mat2, int a, int b)

  Elementwise summation.
- double ** sub (double **mat1, double **mat2, int a, int b)

  Elementwise subtraction.
- double ** matmul (double **mat1, double **mat2, int a, int b, int c)

  Matrix multiplication.
- double ** transpose (double **mat, int a, int b)

  Matrix transpose.
- double ** identity (int a, double val)

  Identity matrix.
- double ** mat_copy (double **mat, int a, int b)

  Hard copy matrix.
- double ** inverse (double **A, int a, int b)

  Matrix left inverse.
- double ** zero (int a, int b)

  Initialize matrix.
- void print (double **A, int a, int b)

  Print matrix.

5.1.1 Detailed Description

Various matrix operations.

Date

Mar 19, 2021

Author

C J Park
chanjure@snu.ac.kr

5.1.2 Function Documentation

5.1.2.1 identity()

```
double** identity (
            int ,
            double val = 1.  )
```

Identity matrix.

create identity matrix of given size square matrix will be created

Parameters:

Parameters

a	(int) dimension of identity matrix
val	(double) value to initalize (default = 1.)

Returns:

Returns

result (double∗∗) identity matrix

Example:

A = identity(3, 0.5);

Tag: identity initalize

Definition at line 88 of file Matrix.C.

Referenced by inverse().

```
88                                      {
89     static double** result;
90
91     result = new double*[a];
92     for(int i=0;i<a;i++) *(result+i) = new double [a];
93
94     for(int i=0;i<a;i++){
95         for(int j=0;j<a;j++){
96             if(i==j) *(*(result+i)+j) = val;
97             else *(*(result+i)+j) = 0.;
98         }
99     }
100
101     return result;
102 }
```

5.1.2.2 inverse()

```
double** inverse (
            double ** ,
            int ,
            int  )
```

Matrix left inverse.

Matrix left inverse using Gaussian reduction

Parameters:

Parameters

A	(double∗∗) matrix to inverse
a	(int) number of rows of A
b	(int) number of columns of A

Returns:

Returns

result (double∗∗) left inverse of A

Example:

A_inv = inverse(A,2,2);

Tag: left inverse gaussian reduction

Definition at line 117 of file Matrix.C.

References identity(), mat_copy(), matmul(), print(), and transpose().

```
117                                         {
118
119     static double** result; // output matmul(invsqA, A.T)
120     double** sqA; // squarized matrix = matmul(A.transpose, A)
121     double** invsqA; // inverse of squared A
122     double** A_t; // Transposed matrix
123     double* row; // Temporary row container
124     double* invrow; // Temporary row container for inverse matrix - memorize sort order
125     double det; // determinant
126     double pivot;
127
128     const double tol = 1e-14;
129
130     result = new double*[b];
131     for(int i=0;i<b;i++) *(result+i) = new double [a];
132
133     sqA = new double*[b];
134     for(int i=0;i<b;i++) *(sqA+i) = new double [b];
135
136     invsqA = new double*[b];
137     for(int i=0;i<b;i++) *(invsqA+i) = new double [b];
138
139     row = new double [b];
140
141     // Initially identity matrix for inverse matrix container.
142     invsqA = identity(b);
143
144     // Initialize transposed matrix.
145     A_t = new double*[b];
146     for(int i=0; i<b;i++) *(A_t+i) = new double [b];
147
148     if(a!=b){
149         A_t = transpose(A,a,b);
150         // Squarize the matrix
151         for(int i=0;i<b;i++){
152             for(int j=0;j<b;j++){
153                 for(int k=0;k<a;k++){
154                     *(*(sqA+i)+j) = *(*(A_t+i)+k) * *(*(A+k)+j);
155                 }
156             }
157         }
158     }
159     else{
160         A_t = identity(a);
161         sqA = mat_copy(A,a,a);
162     }
163
164     // Sort pivot
165     for(int j=0;j<b;j++){
166         int i = 1;
```

```
167         while(*(*(sqA+j)+j) <=tol && i<b){
168             // Save zero pivot row
169             row = *(sqA+j);
170             invrow = *(invsqA+j);
171 
172             // Exchange with row below
173             *(sqA+j) = *(sqA+j+i);
174             *(invsqA+j) = *(invsqA+j+i);
175 
176             *(sqA+j+i) = row;
177             *(invsqA+j+i) = invrow;
178 
179             i++;
180         }
181     }
182     print(sqA,3,3);
183 
184     // Gauss Jordan method
185     for(int i=0;i<b;i++){
186         // Scale
187         pivot = *(*(sqA+i)+i);
188         printf("pivot : %f\n",pivot);
189         if(pivot!=1.){
190             for(int j=0;j<b;j++){
191                 *(*(invsqA+i)+j)/=pivot;
192                 *(*(sqA+i)+j)/=pivot;
193             }
194         }
195         // Subtraction
196         for(int j=0;j<b;j++){
197             if(j!=i){
198                 double target = *(*(sqA+j)+i);
199                 for(int k=0;k<b;k++){
200                     *(*(invsqA+j)+k) -= *(*(invsqA+i)+k) * target;
201                     *(*(sqA+j)+k) -= *(*(sqA+i)+k) * target;
202                 }
203             }
204         }
205     }
206 
207     result = matmul(invsqA, A_t, b,b,a);
208 
209     return result;
210 }
```

5.1.2.3 mat_copy()

```
double** mat_copy (
              double ** ,
              int ,
              int  )
```

Hard copy matrix.

create hard copy of given matrix

Parameters:

Parameters

mat	(double**) matrix to copy
a	(int) number of rows of mat
b	(int) number of columns of mat

Returns:

Returns

result (double∗∗) hard copy of mat

Example:

A = mat_copy(B,3,3);

Tag: ftn

Definition at line 104 of file Matrix.C.

Referenced by inverse().

```
104                                             {
105     static double** result;
106
107     result = new double*[a];
108     for(int i=0;i<a;i++) *(result+i) = new double [b];
109
110     for(int i=0;i<a;i++){
111         for(int j=0;j<b;j++) *(*(result+i)+j) = *(*(mat+i)+j);
112     }
113
114     return result;
115 }
```

5.1.2.4 matmul()

```
double** matmul (
              double ** ,
              double ** ,
              int ,
              int ,
              int  )
```

Matrix multiplication.

mat1 ∗ mat2 = result number of column of mat1 must match number of column of mat2

Parameters:

Parameters

mat1,mat2	(double∗∗) matrices to be multiplied
a	(int) number of rows of mat1
b	(int) number of folumns of mat1 = number of rows of mat2
c	(int) number of columns of mat2

Returns:

Returns

result (double∗∗) matrix of dimension a x c

Example:

A = matmul(A,B,2,3,2);

Tag: multiplication

Definition at line 45 of file Matrix.C.

Referenced by chisqr(), and inverse().

```
45                                                                      {
46
47     static double** result;
48
49     result = new double*[a];
50     for(int i=0;i<a;i++) *(result + i) = new double [c];
51
52     for(int i=0;i<a;i++){
53         for(int j=0;j<c;j++) *(*(result+i)+j) = 0.;
54     }
55
56     for(int i=0;i<a;i++){
57         for(int j=0;j<c;j++){
58             for(int k=0;k<b;k++){
59                 *(*(result+i)+j) += *(*(mat1+i)+k) * *(*(mat2+k)+j);
60             }
61         }
62     }
63
64     return result;
65 }
```

5.1.2.5 print()

```
void print (
              double ** ,
              int ,
              int  )
```

Print matrix.

print matrix of given size

Parameters:

Parameters

A	(double∗∗) matrix to print
a	(int) number of rows of A
b	(int) number of columns of A

Returns:

Returns

stdout(matrix A)

Example:

print(A,2,3);

Tag: print

Definition at line 225 of file Matrix.C.

Referenced by inverse().

```
225                                     {
226     for(int i=0;i<a;i++){
227         for(int j=0;j<b;j++) printf("%10.4e ", *(*(A+i)+j));
228         printf("\n");
229     }
230 }
```

5.1.2.6 sub()

```
double** sub (
              double ** ,
              double ** ,
              int ,
              int  )
```

Elementwise subtraction.

mat1 - mat2 = result Two input matrices must be in same shape axb

Parameters:

Parameters

mat1,mat2	(double∗∗) matrices to be subtracted
a	(int) number of rows of mat1 and mat2
b	(int) number of columns of mat1 and mat2

Returns:

Returns

result (double∗∗)

Example:

A = sub(A,B,2,2);

Tag: subtract, element wise

Definition at line 31 of file Matrix.C.

Referenced by chisqr().

```
31                                                      {
32
33     static double** result;
34
35     result = new double*[a];
36     for(int i=0;i<a;i++) *(result+i) = new double [b];
37
38     for(int i=0;i<a;i++){
39         for(int j=0;j<b;j++) *(*(result+i)+j) = *(*(mat1+i)+j) - *(*(mat2+i)+j);
40     }
41
42     return result;
43 }
```

5.1.2.7 sum()

```
double** sum (
              double ** ,
              double ** ,
              int ,
              int  )
```

Elementwise summation.

mat1 + mat2 = result Two input matrices must be in same shape axb

Parameters:

Parameters

mat1,mat2	(double∗∗) matrices to be added
a	(int) number of rows of mat1 and mat2
b	(int) number of columns of mat1 and mat2

Returns:

Returns

result (double∗∗)

Example:

A = sum(A,B,2,2);

Tag: add, sum, element wise

Definition at line 17 of file Matrix.C.

```
17                                                      {
18
19     static double** result;
20
21     result = new double*[a];
22     for(int i=0;i<a;i++) *(result+i) = new double [b];
23
24     for(int i=0;i<a;i++){
25         for(int j=0;j<b;j++) *(*(result+i)+j) = *(*(mat1+i)+j) + *(*(mat2+i)+j);
26     }
27
28     return result;
29 }
```

5.1.2.8 transpose()

```
double** transpose (
            double ** ,
            int ,
            int  )
```

Matrix transpose.

mat^T

Parameters:

Parameters

mat	(double**) matrix to be transposed
a	(int) number of rows of mat
b	(int) number of columns of mat

Returns:

Returns

result (double**) transposed mat

Example:

A_T = transpose(A,2,2);

Tag: transpose

Definition at line 67 of file Matrix.C.

Referenced by inverse().

```
67                                                  {
68
69     static double** result;
70
71     result = new double*[b];
72     for(int i=0;i<b;i++) *(result+i) = new double [a];
73
74     for(int i=0;i<a;i++){
75         for(int j=0;j<b;j++){
76             *(*(result+j)+i) = *(*(mat+i)+j);
77         }
78     }
79
80     return result;
81 }
```

5.1.2.9 zero()

```
double** zero (
            int ,
            int  )
```

Initialize matrix.

Create and initialize matrix elements to zero

Parameters:

Parameters

a	(int) number of rows
b	(int) number of columns

Returns:

Returns

result (double∗∗) zero matrix of size axb

Example:

A = zero(2,3);

Tag: initalize zero

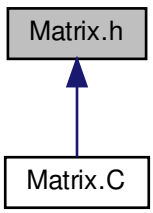
Definition at line 212 of file Matrix.C.

```
212                             {
213     static double** result;
214
215     result = new double*[a];
216     for(int i=0;i<a;i++) *(result+i) = new double [b];
217
218     for(int i=0;i<a;i++){
219         for(int j=0;j<b;j++) *(*(result+i)+j) = 0.;
220     }
221
222     return result;
223 }
```

5.2 Matrix.h File Reference

Header file for Matrix algebra.

This graph shows which files directly or indirectly include this file:



Functions

- double ** sum (double **, double **, int, int)
  Elementwise summation.
- double ** sub (double **, double **, int, int)
  Elementwise subtraction.
- double ** matmul (double **, double **, int, int, int)
  Matrix multiplication.
- double ** transpose (double **, int, int)
  Matrix transpose.
- double ** inverse (double **, int, int)
  Matrix left inverse.
- double ** identity (int, double val=1.)
  Identity matrix.
- double ** mat_copy (double **, int, int)
  Hard copy matrix.
- double ** zero (int, int)
  Initialize matrix.
- void print (double **, int, int)
  Print matrix.

5.2.1 Detailed Description

Header file for Matrix algebra.

Date

Mar 19, 2021

Author

C J Park
chanjure@snu.ac.kr

Bug No known bugs.

Version

0.1

5.2.2 Function Documentation

5.2.2.1 identity()

```
double** identity (
            int ,
            double val = 1.  )
```

Identity matrix.

create identity matrix of given size square matrix will be created

Parameters:

Parameters

a	(int) dimension of identity matrix
val	(double) value to initalize (default = 1.)

Returns:

Returns

result (double∗∗) identity matrix

Example:

A = identity(3, 0.5);

Tag: identity initalize

Definition at line 88 of file Matrix.C.

Referenced by inverse().

```
88                                      {
89     static double** result;
90
91     result = new double*[a];
92     for(int i=0;i<a;i++) *(result+i) = new double [a];
93
94     for(int i=0;i<a;i++){
95         for(int j=0;j<a;j++){
96             if(i==j) *(*(result+i)+j) = val;
97             else *(*(result+i)+j) = 0.;
98         }
99     }
100
101     return result;
102 }
```

5.2.2.2 inverse()

```
double** inverse (
             double ** ,
             int ,
             int  )
```

Matrix left inverse.

Matrix left inverse using Gaussian reduction

Parameters:

Parameters

A	(double∗∗) matrix to inverse
a	(int) number of rows of A
b	(int) number of columns of A

Returns:

Returns

result (double∗∗) left inverse of A

Example:

A_inv = inverse(A,2,2);

Tag: left inverse gaussian reduction

Definition at line 117 of file Matrix.C.

References identity(), mat_copy(), matmul(), print(), and transpose().

```
117                                         {
118
119     static double** result; // output matmul(invsqA, A.T)
120     double** sqA; // squarized matrix = matmul(A.transpose, A)
121     double** invsqA; // inverse of squared A
122     double** A_t; // Transposed matrix
123     double* row; // Temporary row container
124     double* invrow; // Temporary row container for inverse matrix - memorize sort order
125     double det; // determinant
126     double pivot;
127
128     const double tol = 1e-14;
129
130     result = new double*[b];
131     for(int i=0;i<b;i++) *(result+i) = new double [a];
132
133     sqA = new double*[b];
134     for(int i=0;i<b;i++) *(sqA+i) = new double [b];
135
136     invsqA = new double*[b];
137     for(int i=0;i<b;i++) *(invsqA+i) = new double [b];
138
139     row = new double [b];
140
141     // Initially identity matrix for inverse matrix container.
142     invsqA = identity(b);
143
144     // Initialize transposed matrix.
145     A_t = new double*[b];
146     for(int i=0; i<b;i++) *(A_t+i) = new double [b];
147
148     if(a!=b){
149         A_t = transpose(A,a,b);
150         // Squarize the matrix
151         for(int i=0;i<b;i++){
152             for(int j=0;j<b;j++){
153                 for(int k=0;k<a;k++){
154                     *(*(sqA+i)+j) = *(*(A_t+i)+k) * *(*(A+k)+j);
155                 }
156             }
157         }
158     }
159     else{
160         A_t = identity(a);
161         sqA = mat_copy(A,a,a);
162     }
163
164     // Sort pivot
165     for(int j=0;j<b;j++){
166         int i = 1;
167         while(*(*(sqA+j)+j) <=tol && i<b){
168             // Save zero pivot row
169             row = *(sqA+j);
170             invrow = *(invsqA+j);
171
172             // Exchange with row below
173             *(sqA+j) = *(sqA+j+i);
174             *(invsqA+j) = *(invsqA+j+i);
175
176             *(sqA+j+i) = row;
177             *(invsqA+j+i) = invrow;
178
```

```
179             i++;
180         }
181     }
182     print(sqA,3,3);
183 
184     // Gauss Jordan method
185     for(int i=0;i<b;i++){
186         // Scale
187         pivot = *(*(sqA+i)+i);
188         printf("pivot : %f\n",pivot);
189         if(pivot!=1.){
190             for(int j=0;j<b;j++){
191                 *(*(invsqA+i)+j)/=pivot;
192                 *(*(sqA+i)+j)/=pivot;
193             }
194         }
195         // Subtraction
196         for(int j=0;j<b;j++){
197             if(j!=i){
198                 double target = *(*(sqA+j)+i);
199                 for(int k=0;k<b;k++){
200                     *(*(invsqA+j)+k) -= *(*(invsqA+i)+k) * target;
201                     *(*(sqA+j)+k) -= *(*(sqA+i)+k) * target;
202                 }
203             }
204         }
205     }
206 
207     result = matmul(invsqA, A_t, b,b,a);
208 
209     return result;
210 }
```

5.2.2.3 mat_copy()

```
double** mat_copy (
              double ** ,
              int ,
              int  )
```

Hard copy matrix.

create hard copy of given matrix

Parameters:

Parameters

mat	(double∗∗) matrix to copy
a	(int) number of rows of mat
b	(int) number of columns of mat

Returns:

Returns

result (double∗∗) hard copy of mat

Example:

A = mat_copy(B,3,3);

Tag: ftn

Definition at line 104 of file Matrix.C.

Referenced by inverse().

```
104                                                 {
105     static double** result;
106
107     result = new double*[a];
108     for(int i=0;i<a;i++) *(result+i) = new double [b];
109
110     for(int i=0;i<a;i++){
111         for(int j=0;j<b;j++) *(*(result+i)+j) = *(*(mat+i)+j);
112     }
113
114     return result;
115 }
```

5.2.2.4 matmul()

```
double** matmul (
              double ** ,
              double ** ,
              int ,
              int ,
              int  )
```

Matrix multiplication.

mat1 ∗ mat2 = result number of column of mat1 must match number of column of mat2

Parameters:

Parameters

mat1,mat2	(double∗∗) matrices to be multiplied
a	(int) number of rows of mat1
b	(int) number of folumns of mat1 = number of rows of mat2
c	(int) number of columns of mat2

Returns:

Returns

result (double∗∗) matrix of dimension a x c

Example:

A = matmul(A,B,2,3,2);

Tag: multiplication

Definition at line 45 of file Matrix.C.

Referenced by chisqr(), and inverse().

```
45                                                                  {
46
47     static double** result;
48
49     result = new double*[a];
50     for(int i=0;i<a;i++) *(result + i) = new double [c];
```

```
51
52      for(int i=0;i<a;i++){
53          for(int j=0;j<c;j++) *(*(result+i)+j) = 0.;
54      }
55
56      for(int i=0;i<a;i++){
57          for(int j=0;j<c;j++){
58              for(int k=0;k<b;k++){
59                  *(*(result+i)+j) += *(*(mat1+i)+k) * *(*(mat2+k)+j);
60              }
61          }
62      }
63
64      return result;
65  }
```

5.2.2.5 print()

```
void print (
              double ** ,
              int ,
              int  )
```

Print matrix.

print matrix of given size

Parameters:

Parameters

A	(double∗∗) matrix to print
a	(int) number of rows of A
b	(int) number of columns of A

Returns:

Returns

stdout(matrix A)

Example:

print(A,2,3);

Tag: print

Definition at line 225 of file Matrix.C.

Referenced by inverse().

```
225                                     {
226      for(int i=0;i<a;i++){
227          for(int j=0;j<b;j++) printf("%10.4e ", *(*(A+i)+j));
228          printf("\n");
229      }
230  }
```

5.2.2.6 sub()

```
double** sub (
             double ** ,
             double ** ,
             int ,
             int  )
```

Elementwise subtraction.

mat1 - mat2 = result Two input matrices must be in same shape axb

Parameters:

Parameters

mat1,mat2	(double∗∗) matrices to be subtracted
a	(int) number of rows of mat1 and mat2
b	(int) number of columns of mat1 and mat2

Returns:

Returns

result (double∗∗)

Example:

A = sub(A,B,2,2);

Tag: subtract, element wise

Definition at line 31 of file Matrix.C.

Referenced by chisqr().

```
31                                                          {
32
33     static double** result;
34
35     result = new double*[a];
36     for(int i=0;i<a;i++) *(result+i) = new double [b];
37
38     for(int i=0;i<a;i++){
39         for(int j=0;j<b;j++) *(*(result+i)+j) = *(*(mat1+i)+j) - *(*(mat2+i)+j);
40     }
41
42     return result;
43 }
```

5.2.2.7 sum()

```
double** sum (
             double ** ,
             double ** ,
             int ,
             int  )
```

Elementwise summation.

mat1 + mat2 = result Two input matrices must be in same shape axb

Parameters:

Parameters

mat1,mat2	(double∗∗) matrices to be added
a	(int) number of rows of mat1 and mat2
b	(int) number of columns of mat1 and mat2

Returns:

Returns

result (double∗∗)

Example:

A = sum(A,B,2,2);

Tag: add, sum, element wise

Definition at line 17 of file Matrix.C.

```
17                                                      {
18
19     static double** result;
20
21     result = new double*[a];
22     for(int i=0;i<a;i++) *(result+i) = new double [b];
23
24     for(int i=0;i<a;i++){
25         for(int j=0;j<b;j++) *(*(result+i)+j) = *(*(mat1+i)+j) + *(*(mat2+i)+j);
26     }
27
28     return result;
29 }
```

5.2.2.8 transpose()

```
double** transpose (
             double ** ,
             int ,
             int  )
```

Matrix transpose.

mat^T

Parameters:

Parameters

mat	(double∗∗) matrix to be transposed
a	(int) number of rows of mat
b	(int) number of columns of mat

Returns:

Returns

result (double∗∗) transposed mat

Example:

A_T = transpose(A,2,2);

Tag: transpose

Definition at line 67 of file Matrix.C.

Referenced by inverse().

```
67                                              {
68
69     static double** result;
70
71     result = new double*[b];
72     for(int i=0;i<b;i++) *(result+i) = new double [a];
73
74     for(int i=0;i<a;i++){
75         for(int j=0;j<b;j++){
76             *(*(result+j)+i) = *(*(mat+i)+j);
77         }
78     }
79
80     return result;
81 }
```

5.2.2.9 zero()

```
double** zero (
              int ,
              int  )
```

Initialize matrix.

Create and initialize matrix elements to zero

Parameters:

Parameters

a	(int) number of rows
b	(int) number of columns

Returns:

Returns

result (double∗∗) zero matrix of size axb

Example:

A = zero(2,3);

Tag: initalize zero

Definition at line 212 of file Matrix.C.

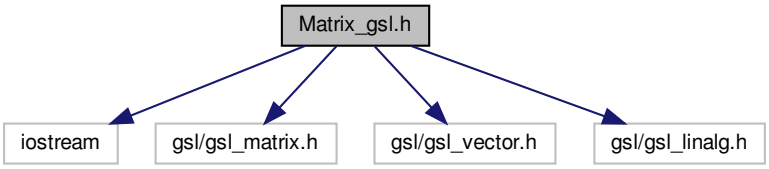
```
212                             {
213     static double** result;
214
215     result = new double*[a];
216     for(int i=0;i<a;i++) *(result+i) = new double [b];
217
218     for(int i=0;i<a;i++){
219         for(int j=0;j<b;j++) *(*(result+i)+j) = 0.;
220     }
221
222     return result;
223 }
```

5.3 Matrix_gsl.h File Reference

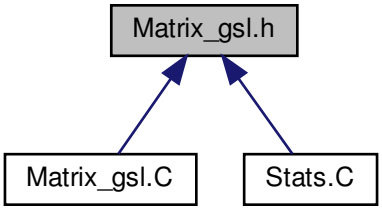
Matrix class and functions using gsl library.

```
#include <iostream>
#include <gsl/gsl_matrix.h>
#include <gsl/gsl_vector.h>
#include <gsl/gsl_linalg.h>
```

Include dependency graph for Matrix_gsl.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- class Matrix

  This is Matrix class.

Functions

- Matrix **operator**∗ (Matrix A, double a)
- Matrix **operator**∗ (double a, Matrix A)
- Matrix **operator**/ (Matrix A, double a)
- Matrix **operator**/ (double a, Matrix A)
- Matrix **add** (Matrix, Matrix)
- Matrix **sub** (Matrix, Matrix)
- Matrix **mul** (Matrix, Matrix)
- Matrix **div** (Matrix, Matrix)
- Matrix **matmul** (Matrix, Matrix)
- Matrix **c_** (Matrix, Matrix)
- Matrix **r_** (Matrix, Matrix)
- double **max** (double ∗∗, int, int)
- double **min** (double ∗∗, int, int)

5.3.1 Detailed Description

Matrix class and functions using gsl library.

Date

Nov 23, 2021

Author

C J Park
chanjure@snu.ac.kr

Bug No known bugs.

Version

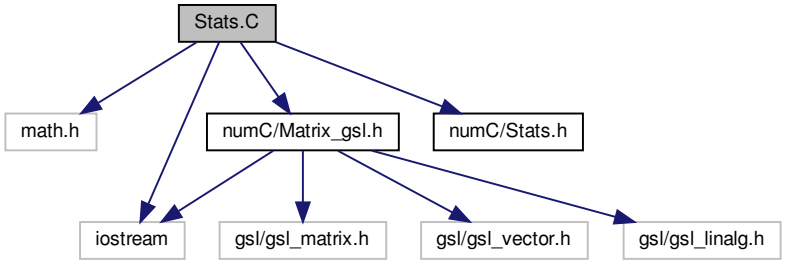
1.0

5.4 Stats.C File Reference

Statistics tools.

```
#include <math.h>
#include <numC/Matrix_gsl.h>
#include <numC/Stats.h>
#include <iostream>
```

Include dependency graph for Stats.C:



Functions

- Matrix ave (Matrix A, int axis)

  Matrix average.
- Matrix var (Matrix A, int axis)

  Matrix variance.
- Matrix cov (Matrix A, int axis)

  Covariance matrix.
- Matrix ∗ JK_resample (Matrix A, int axis)

  Jackknife resample.
- Matrix sample_ave (Matrix ∗A, int l)

  Sample average.
- Matrix JK_error (Matrix ∗A, int l)

  Jackknife standard deviation.
- Matrix BS_error (Matrix ∗A, int l)

  Bootstrap standard deviation.
- double chisqr (Matrix y_bar, Matrix c_inv, Matrix f)

  Chi square.

5.4.1 Detailed Description

Statistics tools.

5.4.2 Function Documentation

5.4.2.1 ave()

```
Matrix ave (
            Matrix A,
            int axis = 1 )
```

Matrix average.

Calculate average along the given axis

Parameters:

Parameters

A	(Matrix) Matrix to calculate
axis	(int) axis to calculate default = 1

Returns:

Returns

result (Matrix) a vector (collapsed Matrix)

Example:

B = ave(A, 0);

Tag: average vector matrix

Definition at line 13 of file Stats.C.

Referenced by cov(), and var().

```
13                              {
14
15   int n = A.shape[0];
16   int m = A.shape[1];
17
18   if(axis == 1){
19
20     Matrix result(n,1,"ave");
21
22     for(int i=0;i<n;i++){
23       for(int j=0;j<m;j++){
24         result.matrix[i][0] += A.matrix[i][j]/(1.*m);
25       }
26     }
27
28     return result;
29   }
30   else if (axis == 0){
31
32     Matrix result(1,m);
33
34     for(int i=0;i<n;i++){
35       for(int j=0;j<m;j++) result.matrix[0][j] += A.matrix[i][j]/(1.*n);
36     }
37
38     return result;
39   }
40 }
```

5.4.2.2 BS_error()

```
Matrix BS_error (
            Matrix * A,
            int l )
```

Bootstrap standard deviation.

Calculate standard deviation(sqrt(var), error) of Bootstrap resampled data

Parameters:

Parameters

A	(Matrix∗) Bootstrap resampled data
l	(int) number of BS samples

Returns:

Returns

result (Martix) collapsed array of Matrices

Example:

A = BS_error(B, 500);

Tag: Bootstrap standard deviation error

Definition at line 163 of file Stats.C.

References sample_ave().

```
163                                {
164   int n = A[0].shape[0];
165
166   Matrix A_ave(n,1);
167   A_ave = sample_ave(A, l);
168
169   Matrix result(n,1);
170
171   for(int i=0;i<n;i++){
172     for(int j=0;j<l;j++) result.matrix[i][0] += pow(A[j].matrix[i][0] - A_ave.matrix[i][0],2.) / (l-1.);
173   }
174
175   for(int i=0;i<n;i++) result.matrix[i][0] = sqrt(result.matrix[i][0]);
176
177   return result;
178 }
```

5.4.2.3 chisqr()

```
double chisqr (
             Matrix y_bar,
             Matrix c_inv,
             Matrix f )
```

Chi square.

Calculate chi-square of given data chi^2 = (y_bar - f)^T C^-1 (y_bar - f)

Parameters:

Parameters

y_bar	(Matrix) vector containing an average of the data
c_inv	(Matrix) inverse of covariance matrix
f	(Matrix) vector of fitting function values

Returns:

Returns

result (double) chi^2 of given data and fitting function

Example:

chisq = chiqr(Y_bar, C_inv, F);

Tag: chi-square fitting

Definition at line 180 of file Stats.C.

References matmul(), and sub().

```
180                                                 {
181
182   double result;
183
184   result = matmul(matmul(sub(y_bar, f).T(), c_inv), sub(y_bar, f)).matrix[0][0];
185
186   return result;
187 }
```

5.4.2.4 cov()

```
Matrix cov (
             Matrix A,
             int axis = 0 )
```

Covariance matrix.

Calculate covariance along the given axis

Parameters:

Parameters

A	(Matrix) Matrix of data
axis	(int) axis to calculate default = 0

Returns:

Returns

result (Matrix) covariance of vectors along axis

Example:

C = cov(A, 0);

Tag: covariance vector matrix

Definition at line 70 of file Stats.C.

References ave().

```
70                              {
71   // Default : sample covariance
72
73   int n = A.shape[0];
74   int m = A.shape[1];
75
76   if(axis == 0){
77     Matrix A_ave;
78     A_ave = ave(A, 1);
79
80     Matrix result(n,n);
```

```
81
82      for(int i=0;i<n;i++){
83        for(int j=0;j<n;j++){
84          for(int k=0;k<m;k++){
85            result.matrix[i][j] += (A.matrix[i][k] - A_ave.matrix[i][0]) * (A.matrix[j][k] - A_ave.matrix[j][
       0])/(1.*m)/(m-1.);
86          }
87        }
88      }
89
90      return result;
91    }
92    else if(axis == 1){
93      Matrix A_ave = ave(A, 0);
94
95      Matrix result(m,m);
96
97      for(int i=0;i<m;i++){
98        for(int j=0;j<m;j++){
99          for(int k=0;k<n;k++) result.matrix[i][j] += (A.matrix[k][i] - A_ave.matrix[0][i]) * (A.matrix[k][j]
       - A_ave.matrix[0][j])/(1.*m)/(m-1.);
100       }
101     }
102
103     return result;
104   }
105 }
```

5.4.2.5 JK_error()

```
Matrix JK_error (
              Matrix * A,
              int l )
```

Jackknife standard deviation.

Calculate standard deviation(sqrt(var), error) of Jackknife resampled data

Parameters:

Parameters

A	(Matrix*) Jackknife resampled data
l	(int) number of JK samples (N)

Returns:

Returns

result (Matrix) collapsed array of Matrices

Example:

A = JK_error(B, 200);

Tag: Jackknife standard deviation error

Definition at line 145 of file Stats.C.

References sample_ave().

```
145                                 {
146
147   int n = A[0].shape[0];
148
149   Matrix A_ave(n,1);
150   A_ave = sample_ave(A, 1);
151
152   Matrix result(n, 1);
153
154   for(int i=0;i<n;i++){
155     for(int j=0;j<1;j++) result.matrix[i][0] += pow(A[j].matrix[i][0] - A_ave.matrix[i][0], 2.) /(1.*1)*(1-
      1.);
156   }
157
158   for(int i=0;i<n;i++) result.matrix[i][0] = sqrt(result.matrix[i][0]);
159
160   return result;
161 }
```

5.4.2.6 JK_resample()

```
Matrix* JK_resample (
            Matrix A,
            int axis = 1 )
```

Jackknife resample.

Resample along the given axis It samples maximum amount of data (N samples)

Parameters:

Parameters

A	(Matrix) Matrix of data
axis	(int) axis to calculate default = 1

Returns:

Returns

result (Matrix∗) Array(Pointer) of matrix

Example:

X = JK_resample(A, 0);

Tag: Jackknife resampling

Definition at line 107 of file Stats.C.

```
107                                     {
108
109   int n = A.shape[0];
110   int m = A.shape[1];
111
112   Matrix *result;
113   result = new Matrix [m];
114   for(int i=0;i<m;i++) result[i].init(n, m-1);
115
116   int p=0;
117   for(int i=0;i<m;i++){
```

```
118     for(int j=0;j<n;j++){
119       p = 0;
120       for(int k=0;k<m;k++){
121         if(i != k){
122           result[i].matrix[j][p] = A.matrix[j][k];
123           p++;
124         }
125       }
126     }
127   }
128
129   return result;
130 }
```

5.4.2.7 sample_ave()

```
Matrix sample_ave (
             Matrix * A,
             int l )
```

Sample average.

Calculate average of array(pointer) of Matrices

Parameters:

Parameters

A	(Matrix∗) Pointer to Matrix eg) resampled data
l	(int) number of JK samples (N)

Returns:

Returns

result (Matrix) collapsed array of Matrices

Example:

A = sample_ave(B, 200);

Tag: Jackknife average

Definition at line 132 of file Stats.C.

Referenced by BS_error(), and JK_error().

```
132                                  {
133
134   int n = A[0].shape[0];
135
136   Matrix result(n,1);
137
138   for(int i=0;i<n;i++){
139     for(int j=0;j<l;j++) result.matrix[i][0] += A[j].matrix[i][0]/(1.*l);
140   }
141
142   return result;
143 }
```

5.4.2.8 var()

```
Matrix var (
             Matrix A,
             int axis = 1 )
```

Matrix variance.

Calculate variance along the given axis

Parameters:

Parameters

A	(Matrix) Matrix to calculate
axis	(int) axis to calculate default = 1

Returns:

Returns

result (Matrix) a vector (collapsed Matrix)

Example:

B = var(A,0);

Tag: variance vector matrix

Definition at line 42 of file Stats.C.

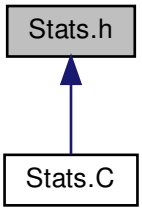
References ave().

```
42                              {
43   // Default : sample var
44
45   int n = A.shape[0];
46   int m = A.shape[1];
47
48   if(axis == 1){
49     Matrix A_ave = ave(A, 1);
50     Matrix result(n,1);
51
52     for(int i=0;i<n;i++){
53       for(int j=0;j<m;j++) result.matrix[i][0] += pow(A.matrix[i][j] - A_ave.matrix[i][0], 2.)/(1.*m)/(m -
      1.);
54     }
55
56     return result;
57   }
58   else if(axis == 0){
59     Matrix A_ave = ave(A, 0);
60     Matrix result(1,m);
61
62     for(int i=0;i<n;i++){
63       for(int j=0;j<m;j++) result.matrix[0][j] += pow(A.matrix[i][j] - A_ave.matrix[0][j], 2.)/(1.*n)/(n -
      1.);
64     }
65
66     return result;
67   }
68 }
```

5.5 Stats.h File Reference

Statistics tools.

This graph shows which files directly or indirectly include this file:



Functions

- Matrix ave (Matrix A, int axis=1)
  Matrix average.
- Matrix var (Matrix A, int axis=1)
  Matrix variance.
- Matrix cov (Matrix A, int axis=0)
  Covariance matrix.
- Matrix ∗ JK_resample (Matrix A, int axis=1)
  Jackknife resample.
- Matrix sample_ave (Matrix ∗A, int l)
  Sample average.
- Matrix JK_error (Matrix ∗A, int l)
  Jackknife standard deviation.
- Matrix BS_error (Matrix ∗A, int l)
  Bootstrap standard deviation.
- double chisqr (Matrix y_bar, Matrix c_inv, Matrix f)
  Chi square.

5.5.1 Detailed Description

Statistics tools.

Date

Nov 22, 2021

Author

C J Park
chanjure@snu.ac.kr

Bug No known bugs.

Version

1.0

5.5.2 Function Documentation

5.5.2.1 ave()

```
Matrix ave (
            Matrix A,
            int axis = 1 )
```

Matrix average.

Calculate average along the given axis

Parameters:

Parameters

A	(Matrix) Matrix to calculate
axis	(int) axis to calculate default = 1

Returns:

Returns

result (Matrix) a vector (collapsed Matrix)

Example:

B = ave(A, 0);

Tag: average vector matrix

Definition at line 13 of file Stats.C.

Referenced by cov(), and var().

```
13                              {
14
15   int n = A.shape[0];
16   int m = A.shape[1];
17
18   if(axis == 1){
19
20     Matrix result(n,1,"ave");
21
22     for(int i=0;i<n;i++){
23       for(int j=0;j<m;j++){
24         result.matrix[i][0] += A.matrix[i][j]/(1.*m);
25       }
26     }
27
28     return result;
29   }
30   else if (axis == 0){
31
32     Matrix result(1,m);
33
34     for(int i=0;i<n;i++){
35       for(int j=0;j<m;j++) result.matrix[0][j] += A.matrix[i][j]/(1.*n);
36     }
37
38     return result;
39   }
40 }
```

5.5.2.2 BS_error()

```
Matrix BS_error (
            Matrix * A,
            int l )
```

Bootstrap standard deviation.

Calculate standard deviation(sqrt(var), error) of Bootstrap resampled data

Parameters:

Parameters

A	(Matrix*) Bootstrap resampled data
l	(int) number of BS samples

Returns:

Returns

result (Martix) collapsed array of Matrices

Example:

A = BS_error(B, 500);

Tag: Bootstrap standard deviation error

Definition at line 163 of file Stats.C.

References sample_ave().

```
163                                 {
164   int n = A[0].shape[0];
165
166   Matrix A_ave(n,1);
167   A_ave = sample_ave(A, l);
168
169   Matrix result(n,1);
170
171   for(int i=0;i<n;i++){
172     for(int j=0;j<l;j++) result.matrix[i][0] += pow(A[j].matrix[i][0] - A_ave.matrix[i][0],2.) / (l-1.);
173   }
174
175   for(int i=0;i<n;i++) result.matrix[i][0] = sqrt(result.matrix[i][0]);
176
177   return result;
178 }
```

5.5.2.3 chisqr()

```
double chisqr (
            Matrix y_bar,
            Matrix c_inv,
            Matrix f )
```

Chi square.

Calculate chi-square of given data chi^2 = (y_bar - f)^T C^-1 (y_bar - f)

Parameters:

Parameters

y_bar	(Matrix) vector containing an average of the data
c_inv	(Matrix) inverse of covariance matrix
f	(Matrix) vector of fitting function values

Returns:

Returns

result (double) chi^2 of given data and fitting function

Example:

chisq = chiqr(Y_bar, C_inv, F);

Tag: chi-square fitting

Definition at line 180 of file Stats.C.

References matmul(), and sub().

```
180                                                 {
181
182   double result;
183
184   result = matmul(matmul(sub(y_bar, f).T(), c_inv), sub(y_bar, f)).matrix[0][0];
185
186   return result;
187 }
```

5.5.2.4 cov()

```
Matrix cov (
             Matrix A,
             int axis = 0 )
```

Covariance matrix.

Calculate covariance along the given axis

Parameters:

Parameters

A	(Matrix) Matrix of data
axis	(int) axis to calculate default = 0

Returns:

Returns

result (Matrix) covariance of vectors along axis

Example:

C = cov(A, 0);

Tag: covariance vector matrix

Definition at line 70 of file Stats.C.

References ave().

```
70                                {
71   // Default : sample covariance
72
73   int n = A.shape[0];
74   int m = A.shape[1];
75
76   if(axis == 0){
77     Matrix A_ave;
78     A_ave = ave(A, 1);
79
80     Matrix result(n,n);
81
82     for(int i=0;i<n;i++){
83       for(int j=0;j<n;j++){
84         for(int k=0;k<m;k++){
85           result.matrix[i][j] += (A.matrix[i][k] - A_ave.matrix[i][0]) * (A.matrix[j][k] - A_ave.matrix[j][
      0])/(1.*m)/(m-1.);
86         }
87       }
88     }
89
90     return result;
91   }
92   else if(axis == 1){
93     Matrix A_ave = ave(A, 0);
94
95     Matrix result(m,m);
96
97     for(int i=0;i<m;i++){
98       for(int j=0;j<m;j++){
99         for(int k=0;k<n;k++) result.matrix[i][j] += (A.matrix[k][i] - A_ave.matrix[0][i]) * (A.matrix[k][j]
      - A_ave.matrix[0][j])/(1.*m)/(m-1.);
100       }
101     }
102
103     return result;
104   }
105 }
```

5.5.2.5 JK_error()

```
Matrix JK_error (
              Matrix * A,
              int l )
```

Jackknife standard deviation.

Calculate standard deviation(sqrt(var), error) of Jackknife resampled data

Parameters:

Parameters

A	(Matrix∗) Jackknife resampled data
l	(int) number of JK samples (N)

Returns:

Returns

result (Matrix) collapsed array of Matrices

Example:

A = JK_error(B, 200);

Tag: Jackknife standard deviation error

Definition at line 145 of file Stats.C.

References sample_ave().

```
145                                {
146
147   int n = A[0].shape[0];
148
149   Matrix A_ave(n,1);
150   A_ave = sample_ave(A, l);
151
152   Matrix result(n, 1);
153
154   for(int i=0;i<n;i++){
155     for(int j=0;j<l;j++) result.matrix[i][0] += pow(A[j].matrix[i][0] - A_ave.matrix[i][0], 2.) /(1.*l)*(l-
      1.);
156   }
157
158   for(int i=0;i<n;i++) result.matrix[i][0] = sqrt(result.matrix[i][0]);
159
160   return result;
161 }
```

5.5.2.6 JK_resample()

```
Matrix* JK_resample (
             Matrix A,
             int axis = 1 )
```

Jackknife resample.

Resample along the given axis It samples maximum amount of data (N samples)

Parameters:

Parameters

A	(Matrix) Matrix of data
axis	(int) axis to calculate default = 1

Returns:

Returns

result (Matrix∗) Array(Pointer) of matrix

Example:

X = JK_resample(A, 0);

Tag: Jackknife resampling

Definition at line 107 of file Stats.C.

```
107                                        {
108
109   int n = A.shape[0];
110   int m = A.shape[1];
111
112   Matrix *result;
113   result = new Matrix [m];
114   for(int i=0;i<m;i++) result[i].init(n, m-1);
115
116   int p=0;
117   for(int i=0;i<m;i++){
118     for(int j=0;j<n;j++){
119       p = 0;
120       for(int k=0;k<m;k++){
121         if(i != k){
122           result[i].matrix[j][p] = A.matrix[j][k];
123           p++;
124         }
125       }
126     }
127   }
128
129   return result;
130 }
```

5.5.2.7 sample_ave()

```
Matrix sample_ave (
            Matrix * A,
            int l )
```

Sample average.

Calculate average of array(pointer) of Matrices

Parameters:

Parameters

A	(Matrix∗) Pointer to Matrix eg) resampled data
l	(int) number of JK samples (N)

Returns:

Returns

result (Matrix) collapsed array of Matrices

Example:

A = sample_ave(B, 200);

Tag: Jackknife average

Definition at line 132 of file Stats.C.

Referenced by BS_error(), and JK_error().

```
132                                     {
133
134   int n = A[0].shape[0];
135
136   Matrix result(n,1);
137
138   for(int i=0;i<n;i++){
139     for(int j=0;j<1;j++) result.matrix[i][0] += A[j].matrix[i][0]/(1.*1);
140   }
141
142   return result;
143 }
```

5.5.2.8 var()

```
Matrix var (
              Matrix A,
              int axis = 1 )
```

Matrix variance.

Calculate variance along the given axis

Parameters:

Parameters

A	(Matrix) Matrix to calculate
axis	(int) axis to calculate default = 1

Returns:

Returns

result (Matrix) a vector (collapsed Matrix)

Example:

B = var(A,0);

Tag: variance vector matrix

Definition at line 42 of file Stats.C.

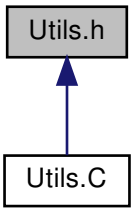
References ave().

```
42                                {
43   // Default : sample var
44
45   int n = A.shape[0];
46   int m = A.shape[1];
47
48   if(axis == 1){
49     Matrix A_ave = ave(A, 1);
50     Matrix result(n,1);
51
52     for(int i=0;i<n;i++){
53       for(int j=0;j<m;j++) result.matrix[i][0] += pow(A.matrix[i][j] - A_ave.matrix[i][0], 2.)/(1.*m)/(m -
      1.);
54     }
55
56     return result;
57   }
58   else if(axis == 0){
59     Matrix A_ave = ave(A, 0);
60     Matrix result(1,m);
61
62     for(int i=0;i<n;i++){
63       for(int j=0;j<m;j++) result.matrix[0][j] += pow(A.matrix[i][j] - A_ave.matrix[0][j], 2.)/(1.*n)/(n -
      1.);
64     }
65
66     return result;
67   }
68 }
```

5.6 Utils.h File Reference

File io uitls This functions are not compatible with Matrix_gsl functions. This is compatible with Matrix functions.

This graph shows which files directly or indirectly include this file:



Functions

- void readfile (const char *fname, double **x, double **y, int n_x=7, int n_y=200)

  readfile

5.6.1 Detailed Description

File io uitls This functions are not compatible with Matrix_gsl functions. This is compatible with Matrix functions.

Date

Nov 23, 2021

Author

C J Park
chanjure@snu.ac.kr

Bug No known bugs.

Version

1.0

5.6.2 Function Documentation

5.6.2.1 readfile()

```
void readfile (
            const char * fname,
            double ** x,
            double ** y,
            int n_x = 7,
            int n_y = 200 )
```

readfile

read file with specific format. To be generalized.

Parameters:

Parameters

fname	(const char∗) file name to read
x	(double∗∗) data points
y	(double∗∗) data values
n_x	(int) number of data points (default = 7)
n_y	(int) number of data values (default = 200)

Returns:

Returns

void

Example:

readfile(data6, X, Y, 7, 200);

Tag: file io read file

Definition at line 5 of file Utils.C.

```
5                                                                                {
6
7   FILE *data;
8   char buff[255];
9   std::string X = "X";
10   std::string data_str = "DATA";
11
12   int count=0;
13   double temp=0.;
14
15   data = fopen(fname,"r");
16
17   while(fscanf(data,"%s",buff)!=EOF){
18     if(!X.compare(buff)){
19       for(int i=0;i<n_x;i++){
20         fscanf(data, "%lf", &temp);
21         *(*(x+i)+0) = temp;
22       }
23       continue;
24     }
25
26     if(!data_str.compare(buff)){
27       fscanf(data, "%s",buff);
28       continue;
29     }
30
31     for(int i=0;i<n_x;i++){
32       fscanf(data,"%lf",&temp);
33       *(*(y+i)+count) = temp;
34     }
35     count++;
36   }
37
38   fclose(data);
39
40   /*for(int i=0;i<7;i++) printf("%8.5f\n",*(*(x+i)+0));
41   for(int i=0;i<7;i++){
42     for(int j=0;j<200;j++) printf("%8.5f\t",*(*(y+i)+j));
43     printf("\n");
44   }*/
45 }
```

Index