numC

1.0

Generated by Doxygen 1.8.13

Contents

1 Bug List

File Matrix.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

Public Member Functions

- **Matrix** (std::string name="New")
- **Matrix** (int n, int m, std::string name="New")
- **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

Definition at line 9 of file Matrix_gsl.h.

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

- Matrix_gsl.h
- Matrix_gsl.C

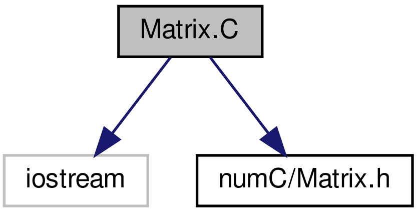
5 File Documentation

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().

```
                                {
    static double** result;

    result = new double*[a];
    for(int i=0;i<a;i++) *(result+i) = new double [a];

    for(int i=0;i<a;i++){
        for(int j=0;j<a;j++){
            if(i==j) *(*(result+i)+j) = val;
            else *(*(result+i)+j) = 0.;
        }
    }

     return result;
}
```

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;
```

```
        while(*(*(sqA+j)+j) <=tol && i<b){
            // Save zero pivot row
            row = *(sqA+j);
            invrow = *(invsqA+j);

            // Exchange with row below
            *(sqA+j) = *(sqA+j+i);
            *(invsqA+j) = *(invsqA+j+i);

            *(sqA+j+i) = row;
            *(invsqA+j+i) = invrow;

            i++;
        }
    }
    print(sqA,3,3);

    // Gauss Jordan method
    for(int i=0;i<b;i++){
        // Scale
        pivot = *(*(sqA+i)+i);
        printf("pivot : %f\n",pivot);
        if(pivot!=1.){
            for(int j=0;j<b;j++){
                *(*(invsqA+i)+j)/=pivot;
                *(*(sqA+i)+j)/=pivot;
            }
        }
        // Subtraction
        for(int j=0;j<b;j++){
            if(j!=i){
                double target = *(*(sqA+j)+i);
                for(int k=0;k<b;k++){
                    *(*(invsqA+j)+k) -= *(*(invsqA+i)+k) * target;
                    *(*(sqA+j)+k) -= *(*(sqA+i)+k) * target;
                }
            }
        }
    }

    result = matmul(invsqA, A_t, b,b,a);

    return result;
}
```

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 inverse().

```
                                                                  {

    static double** result;

    result = new double*[a];
    for(int i=0;i<a;i++) *(result + i) = new double [c];

    for(int i=0;i<a;i++){
        for(int j=0;j<c;j++) *(*(result+i)+j) = 0.;
    }

    for(int i=0;i<a;i++){
        for(int j=0;j<c;j++){
            for(int k=0;k<b;k++){
                *(*(result+i)+j) += *(*(mat1+i)+k) * *(*(mat2+k)+j);
            }
        }
    }

    return result;
}
```

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.

```
31                                                         {
32
33     static double** result;
34
35     result = new double*[a];
```

```
    for(int i=0;i<a;i++) *(result+i) = new double [b];

    for(int i=0;i<a;i++){
        for(int j=0;j<b;j++) *(*(result+i)+j) = *(*(mat1+i)+j) - *(*(mat2+i)+j);
    }

    return result;
}
```

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

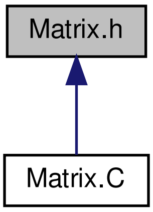
Definition at line 212 of file Matrix.C.

```
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 
```

```
            i++;
        }
    }
    print(sqA,3,3);

    // Gauss Jordan method
    for(int i=0;i<b;i++){
        // Scale
        pivot = *(*(sqA+i)+i);
        printf("pivot : %f\n",pivot);
        if(pivot!=1.){
            for(int j=0;j<b;j++){
                *(*(invsqA+i)+j)/=pivot;
                *(*(sqA+i)+j)/=pivot;
            }
        }
        // Subtraction
        for(int j=0;j<b;j++){
            if(j!=i){
                double target = *(*(sqA+j)+i);
                for(int k=0;k<b;k++){
                    *(*(invsqA+j)+k) -= *(*(invsqA+i)+k) * target;
                    *(*(sqA+j)+k) -= *(*(sqA+i)+k) * target;
                }
            }
        }
    }

    result = matmul(invsqA, A_t, b,b,a);

    return result;
}
```

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().

```
                                            {
    static double** result;

    result = new double*[a];
    for(int i=0;i<a;i++) *(result+i) = new double [b];

    for(int i=0;i<a;i++){
        for(int j=0;j<b;j++) *(*(result+i)+j) = *(*(mat+i)+j);
    }

    return result;
}
```

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 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.

```
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.

```
                           {
    static double** result;

    result = new double*[a];
    for(int i=0;i<a;i++) *(result+i) = new double [b];

    for(int i=0;i<a;i++){
        for(int j=0;j<b;j++) *(*(result+i)+j) = 0.;
    }

    return result;
}
```

Index