A Wrapper Class For LAPACK and BLAS

Minh Tran
Email: minh.t.tran@adelaide.edu.au

September 11, 2018

1 Introduction

The Matrix class is a wrapper class for LAPACK and BLAS. The class implements matrix solvers and operators typically used in mathematical modelling of physical systems. It is the hope of the author that the class will be useful for anyone working with C++ projects that require matrix operations.

2 Examples

The use of class will be demonstrated through examples.

2.1 Example 1

$$\boldsymbol{A}_1 = \begin{bmatrix} 5 & 7 & 6 & 5 \\ 7 & 10 & 8 & 7 \\ 6 & 8 & 10 & 9 \\ 5 & 7 & 9 & 10 \end{bmatrix} \quad , \quad \boldsymbol{b}_1 = \begin{bmatrix} 23 \\ 32 \\ 33 \\ 31 \end{bmatrix}$$

Consider the system,

$$(5 * \boldsymbol{A}_1 + \boldsymbol{A}_1 * 15) * \boldsymbol{x}_1 = \boldsymbol{b}_1$$

$$\boldsymbol{x}_1 = (5 * \boldsymbol{A}_1 + \boldsymbol{A}_1 * 15)^{-1} * \boldsymbol{b}_1$$

$$\boldsymbol{x}_1 = \begin{bmatrix} 0.050 \\ 0.050 \\ 0.050 \\ 0.050 \end{bmatrix}$$

Below are codes to compute $\boldsymbol{x}_1$. The result is stored in the variable `example1`.

```
double A1[] = { 5 ,  7 ,  6 , 5 ,
                7 , 10 ,  8 , 7 ,
                6 ,  8 , 10 , 9 ,
                5 ,  7 ,  9 , 10 };
double b1[] = { 23 , 32 , 33 , 31 };
Matrix P1(A1,4);
Matrix example1 = (5*P1+P1*15) | b1;
```

2.2 Example 2

$$A_2 = \begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 0 \end{bmatrix} \quad , \quad b_2 = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

Consider the system,

$$(\boldsymbol{A}_2 * \boldsymbol{A}_2 * \boldsymbol{A}_2) * \boldsymbol{x}_2 = \boldsymbol{b}_2$$

$$\boldsymbol{x}_2 = (\boldsymbol{A}_2 * \boldsymbol{A}_2 * \boldsymbol{A}_2)^{-1} * \boldsymbol{b}_2$$
$$\boldsymbol{x}_2 \approx \begin{bmatrix} -3.088 \\ 2.694 \\ -0.569 \end{bmatrix}$$

Below are codes to compute $\boldsymbol{x}_2$. The result is stored in the variable `example2`.

```
double A2[] = { 0 , 1 , 2 ,
                3 , 4 , 5 ,
                6 , 7 , 0  };
double B2[] = { 1 , 2 , 3 };
Matrix P2(A2, 3);
Matrix example2 = (P2*P2*P2) | B2;
```

2.3 Example 3 & 4

$$\boldsymbol{A}_3 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \\ 0 & 0 & -1 & 0 \end{bmatrix} \quad , \quad \boldsymbol{d} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

Consider the system,

$$\boldsymbol{A}_3 * \boldsymbol{x}_3 = \boldsymbol{d}$$

$$\boldsymbol{x}_3 = \boldsymbol{A}_3^{-1} * \boldsymbol{d}$$
$$\boldsymbol{x}_3 \approx \begin{bmatrix} -6 \\ 1 \\ -4 \\ 4 \end{bmatrix}$$

Below are codes to compute $\boldsymbol{x}_3$. The result is stored in the variables `example3`. The system for `example4` is a similar to `example3`. The main difference is $\boldsymbol{d}$ is an array and $\boldsymbol{Q}_3$ is a matrix object.

```
double a[] = { -1, -1, -1 };
double b[] = { 0, 0, 0, 0};
double c[] = { 1, 1, 1 };
double d[] = { 1, 2, 3, 4 };
Matrix Q3(d, 4,1);
Matrix P3(a,b,c,4);
Matrix example3 = P3 | d;
Matrix example4 = P3 | Q3;
```

2.4 Example 5 & 6

This example demonstrates the use of pseudo inverse to compute a solution of the system below.

$$\boldsymbol{A}_5 = \begin{bmatrix} -74 & 80 & 18 & -11 & -4 \\ 14 & -69 & 21 & 28 & 0 \\ 66 & -72 & -5 & 7 & 1 \\ -12 & 66 & -30 & -23 & 3 \\ 3 & 8 & -7 & -4 & 1 \\ 4 & -12 & 4 & 4 & 0 \end{bmatrix} \quad , \quad \boldsymbol{d} = \begin{bmatrix} 51 \\ -61 \\ -56 \\ 69 \\ 10 \\ -12 \end{bmatrix}$$

Consider the system,

$$\boldsymbol{A}_5 * \boldsymbol{x}_5 = \boldsymbol{d}$$

$$\boldsymbol{x}_5 = \boldsymbol{A}_5^{-1} * \boldsymbol{d}$$

$$\boldsymbol{x}_5 = \begin{bmatrix} 1 \\ 2 \\ -1 \\ 3 \\ -4 \end{bmatrix}$$

Below are codes to compute $\boldsymbol{x}_5$. The result is stored in the variables `example5`. The system for `example6` is a similar to `example5`. The main difference is $\boldsymbol{b}_5$ is an array and $\boldsymbol{Q}_5$ is a matrix object.

```
double A5[30] = {  -74 ,  80 ,  18 , -11 , -4 ,
                    14 , -69 ,  21 ,  28 , 0 ,
                    66 , -72 ,  -5 ,   7 , 1 ,
                   -12 ,  66 , -30 , -23 , 3 ,
                     3 ,   8 ,  -7 ,  -4 , 1 ,
                     4 , -12 ,   4 ,   4 , 0};
double b5[6] = { 51 , -61 , -56 , 69 , 10 , -12 };
Matrix P5(A5,6,5);
Matrix example5 = P5 | b5;
Matrix Q5(b5, 6, 1);
Matrix example6 = P5 | Q5;
```

2.5 Example 7

Consider solving the system below,

$$\min_{\boldsymbol{x},\boldsymbol{y}} \ ||\boldsymbol{y}||_2 \qquad \text{subject to} \qquad \boldsymbol{d}_7 = \boldsymbol{A}_7\boldsymbol{x} + \boldsymbol{B}_7\boldsymbol{y}$$

$$\boldsymbol{A}_7 = \begin{bmatrix} 1 & 2 & 1 & 4 \\ -1 & 1 & 1 & 1 \\ -1 & -2 & -1 & 1 \\ -1 & 2 & -1 & -1 \\ 1 & 1 & 1 & 2 \end{bmatrix} \quad , \quad \boldsymbol{B}_7 = \begin{bmatrix} 1 & 2 & 2 \\ -1 & 1 & -2 \\ 3 & 1 & 6 \\ 2 & -2 & 4 \\ 1 & -1 & 2 \end{bmatrix} \quad , \quad \boldsymbol{d}_7 = \begin{bmatrix} 7.99 \\ 0.98 \\ -2.98 \\ 3.04 \\ 4.02 \end{bmatrix}$$

Solution:

$$\boldsymbol{x} = \begin{bmatrix} 1.002951 \\ 2.001436 \\ -0.987798 \\ 0.990908 \end{bmatrix} \quad , \quad \boldsymbol{y} = \begin{bmatrix} 0.003436 \\ -0.004417 \\ 0.006871 \end{bmatrix}$$

Below are codes to compute $\boldsymbol{x}$ and $\boldsymbol{y}$. The variable `info` is used to flag whether the algorithm is able to obtain a solution to the system.

```
double *x7 = new double[4];
double *y7 = new double[3];
double A7[] = { 1 ,  2 ,  1 ,  4 ,
               -1 ,  1 ,  1 ,  1 ,
               -1 , -2 , -1 ,  1 ,
               -1 ,  2 , -1 , -1 ,
                1 ,  1 ,  1 ,  2} ;
double B7[] = { 1 ,  2 ,  2 ,
               -1 ,  1 , -2 ,
                3 ,  1 ,  6 ,
                2 , -2 ,  4 ,
                1 , -1 , 2 };
double d7[] = { 7.99 , 0.98 , -2.98 , 3.04 , 4.02 };
Matrix P7(A7, B7, d7, 5, 4, 3);
int info7 = P7.solve(x7,y7);
```

2.6 Example 8

Consider solving the system below,

$$\min_{\boldsymbol{x}_8} \ ||\boldsymbol{d}_8 - \boldsymbol{A}_8\boldsymbol{x}_8||_2 \qquad \text{subject to} \qquad \boldsymbol{H}_8\boldsymbol{x}_8 = \boldsymbol{f}_8$$

$$\boldsymbol{A}_8 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 3 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad , \quad \boldsymbol{d}_8 = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} \quad , \quad \boldsymbol{H}_8 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & -1 \end{bmatrix} \quad , \quad \boldsymbol{f}_8 = \begin{bmatrix} 7 \\ 4 \end{bmatrix}$$

Solution:

$$\boldsymbol{x}_8 = \begin{bmatrix} 5.75 \\ -0.25 \\ 1.5 \end{bmatrix}$$

Below are codes to compute a solution to the system. The variable `info` is used to flag whether the algorithm is able to obtain a solution to the system.

```
double A8[] = { 1 ,  1 , 1 ,
                1 ,  3 , 1 ,
                1 , -1 , 1 ,
                1 ,  1 , 1};
double H8[] = { 1 , 1 , 1 ,
                1 , 1 , -1 };
double D8[] = { 1 ,  2 , 3 , 4 };
double F8[] = { 7 ,
4 };
double *x8 = new double[3];
Matrix P8(A8, D8, H8, F8, 4, 3, 2);
int info8 = P8.solve(x8);
printf("\n X8 = \n ");
Matrix::printMatrix(x8,3,1);
```

3 Constructor

Below is a description of the syntax used to create a single matrix or a matrix system.

3.1 Create a Single Matrix Object

3.1.1 `Matrix( double Ain[], int nd )`

This is used to create a square matrix with dimension of $n_d * n_d$. The variable A_{in} is an array storing entries of the matrix. The entries are read in row by row.

3.1.2 `Matrix(double Ain[], int nd , int md)`

This is used to create a general matrix with dimension of $n_d * m_d$. The variable A_{in} is an array storing entries of the matrix. The entries are read in row by row.

3.1.3 `Matrix(double ain[], double bin[], double cin[], int nd)`

Create a tridiagonal matrix with dimension of $n_d * n_d$. The array b_{in} is the main diagonal with n_d number of entries. The array a_{in} is the subdiagonal with $n_d - 1$ number of entries. The array c_{in} is the superdiagonal with $n_d - 1$ number of entries.

3.1.4 `Matrix(double Ain[], int nd, int md, bool isTranspose)`

This is used to create a general matrix with dimension of $n_d * m_d$. The variable A_{in} is an array storing entries of the matrix. The entries are read in row by row if `isTranspose` is `false`. On the other hand, if `isTranspose` is `true` entries are read in column by column.

3.1.5 `Matrix(double Ain[], int nd, int md, bool isTranspose, int flag)`

This is used to create a general matrix with dimension of $n_d * m_d$. The variable A_{in} is an array storing entries of the matrix. The entries are read in row by row if `isTranspose` is `false`. On the other hand, if `isTranspose` is `true` entries are read in column by column. The parameter input `flag` is used to set the default `info` about the matrix. That is, if `flag` = 0, then the matrix is non-singular. It should be noted that `info` is an instance field of the Matrix class.

3.2 Create a Matrix System

3.2.1 `Matrix(double Ain[], double din[], double Hin[], double fin[], int nd , int md, int pd)`

This is used to construct the following matrix system.

$$\min_{\boldsymbol{x}} \; ||\boldsymbol{d} - \boldsymbol{A}\boldsymbol{x}||_2 \qquad \text{subject to} \qquad \boldsymbol{H}\boldsymbol{x} = \boldsymbol{f}$$

Dimension of the matrix:

$\boldsymbol{A}$	dimension $n_d * m_d$
$\boldsymbol{H}$	dimension $p_d * m_d$
$\boldsymbol{d}$	dimension $n_d * 1$
$\boldsymbol{f}$	dimension $p_d * 1$
$\boldsymbol{x}$	dimension $m_d * 1$

The method "`int Matrix::solve(double xinout[])`" is used to solve the system.

3.2.2 `Matrix(double Ain[], double Bin[], double din[], int nd, int md, int pd)`

This is used to construct the following matrix system.

$$\min_{\boldsymbol{x},\boldsymbol{y}} \ ||\boldsymbol{y}||_2 \qquad \text{subject to} \qquad \boldsymbol{d} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{y}$$

Dimension of the matrix:

$\boldsymbol{A}$	dimension $n_d * m_d$
$\boldsymbol{B}$	dimension $n_d * p_d$
$\boldsymbol{d}$	dimension $n_d * 1$
$\boldsymbol{x}$	dimension $m_d * 1$
$\boldsymbol{y}$	dimension $p_d * 1$

The method “`int Matrix::solve(double xinout[], double yinout[])`” is used to solve the system.

4 Methods

4.0.1 `int Matrix::solve(double xinout[])`

This method is used to solve the system

$$\min_{\boldsymbol{x}} \ ||\boldsymbol{d} - \boldsymbol{A}\boldsymbol{x}||_2 \qquad \text{subject to} \qquad \boldsymbol{H}\boldsymbol{x} = \boldsymbol{f}$$

Solution of the system is stored in the array x_{inout}.

4.0.2 `int Matrix::solve(double xinout[], double yinout[])`

This method is used to solve the system

$$\min_{\boldsymbol{x},\boldsymbol{y}} \ ||\boldsymbol{y}||_2 \qquad \text{subject to} \qquad \boldsymbol{d} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{y}$$

Solution of the system is stored in the array x_{inout} and y_{inout} .

4.0.3 `void Matrix::printMatrix()`

This method is used to print to console all the matrix entries stored in the Matrix object.

4.0.4 `static void Matrix::printMatrix(double input[], int nd, int md)`

This method is used to print to console all the matrix entries stored in an array `input`. The dimension of the matrix is $n_d * m_d$.

5 Operators

Operator	Example	Description
$+$	$\boldsymbol{A}+\boldsymbol{B}$	Adding two matrix object.
-	$\boldsymbol{A}-\boldsymbol{B}$	Substract two matrix object.
$*$	$c*\boldsymbol{A}$	multiply matrix $\boldsymbol{A}$ with a scalar c
$*$	$\boldsymbol{A}*c$	multiply matrix $\boldsymbol{A}$ with a scalar c
$*$	$\boldsymbol{A}*\boldsymbol{B}$	multiply matrix two matrix together
$\mid$	$\boldsymbol{A} \mid \boldsymbol{b}$	$\boldsymbol{b}$ is an array, compute $\boldsymbol{A}^{-1}\boldsymbol{b}$. If $\boldsymbol{A}$ is not a square, matrix pseudo inverse is used.
$\mid$	$\boldsymbol{A} \mid \boldsymbol{B}$	$\boldsymbol{B}$ is matrix, compute $\boldsymbol{A}^{-1}\boldsymbol{B}$. If $\boldsymbol{A}$ is not a square matrix, pseudo inverse is used.