Matrix and Vector class (C++) Manual

1 Introduction

This documents describes the constructors methods and properties which are included in 'CMatrix.dll' and which may be accessed in any traditional Microsoft C++ Win32 environment. There are five classes provided: CVector, CMatrix, CComplex, CComplexVector and CcomplexMatrix. These perform many common matrix and vector manipulations. In particular there is the inclusion of

(i) the powerful SVD calculation which permits the solution of (in a least squares sense) any linear system of the form $Ax = b$.

(ii) The Fast Fourier Transform routine and its inverse.

2 Version history

2.1 Version 1.0.0.0

1.0.0.0	Initial Version
1.0.0.1	Change of name of 'max_no_SVD_loops' to 'max_no_loops'. Addition of Jacobi, Gauss-Seidel nd SOR technigues for solving a non-singular liner system of the form $Ax = b$.
1.0.0.2	Addition of Jacobi_SVD routine
1.0.0.3	Addition of CComplex and CComplexVector class including the FFT routines.
1.0.0.4	Addition of SVD_Light routine.
1.0.0.5	Improvements to SVD routines.
1.0.0.6	Further improvement to speed of SVD routines.
1.0.0.7	Bug fixed in SVD routine.
1.0.0.8	Speed improvements to the SVD_Light routine.
1.0.0.10	Addition of CComplexMatrix class
1.0.0.11	Addition of further Eigenvalue/vector related functions.
1.0.0.12	Addition of CMatrix View method.
1.0.0.13	Removal of restriction that computation of a stepwise SVD (for embedded environments must start with a bi-diagonal matrix.
1.0.0.14	Addition of support for explicit linking.
1.0.0.15	Addition of LU decomposition.

2.2 Manual History

25th June 2013	Initial document.
4th July 2013	The routines concerned with bi-diagonalisation are made more precise.
24th February 2014	Addition of vector insertion, deletion, min., max., mean, sin, cos and quick sort routines
31st March 2014	Change of name of 'max_no_SVD_loops' to 'max_no_loops'. Addition of Jacobi, Gauss-Seidel nd SOR technigues for solving a non-singular liner system of the form $Ax = b$.

18th April 2014	Addition of Jacobi_SVD routine.
25th April 2014	Addition of CComplex and CComplexVector class including the FFT routines.
13th May 2014	Addition of SVD_Light routine.
18th May 2014	Improvements to SVD routines.
22nd May 2014	Further improvement to speed of SVD routines.
19th September 2014	Addition of CComplexMatrix class.
31st October 2014	Change of signature of singular_values method.
18th November 2014	Addition of further Eigenvalue/vector related routines.
24th November 2014	Addition of CMatrix view() method.
2nd December 2014	Removal of restriction that computation of a stepwise SVD (for embedded environments must start with a bi-diagonal matrix.
7th December 2014	Alteration of the installation and usage instructions section to illustrate that when linking statically to a .lib file CMatrix.dll is not needed.
21st March 2015	Clarification of use of CMatrix.dll during run-time.
24th March 2015	Addition of information on explicit linking.
26th April 2015	Addition of LU decomposition.

3 Installation and Usage Instructions

Unzip the downloaded file to reveal four files:

- The CMatrix.dll Win32 dll file
- The CMatrix.lib file
- The CMatrix.h file
- The license file

Use of this software implies acceptance of the license condition contained in the license file.

3.1 Implicit Linking

The following information describes how to use the classes and function via load-time dynamic linking, also referred to as implicit linking (this is the way of using these functions implied in most of the examples in sections 4 to 8). In this manner the application makes calls to the exported Dll functions and classes just as if they were local functions. The header file (CMatrix.h) and import library file (CMatrix.lib) will be needed to compile and link your source code to the classes and functions in the dll just as if you had written them. The file CMatrix.dll is needed during run-time.

To write C++ files which access the classes and functions contained in most of the examples in sections 4 to 8 you will need to do the following:

- Include the header file CMatrix.h in each file including references to the CMatrix classes: e.g.:

```
#include "stdafx.h"
#include "button_implementations.h"
#include "CMatrix.h"
#include "stdlib.h"
#include <math.h>
#include <time.h>
...
```

In order for the compiler to find this additional header file it will be necessary for you to configure your Visual Studio project to find this file. This is done (in VS2010) in Project Properties -> Configuration Properties -> C/C++ -> General -> Additional Include Directories. See the screen shot in Figure 1.

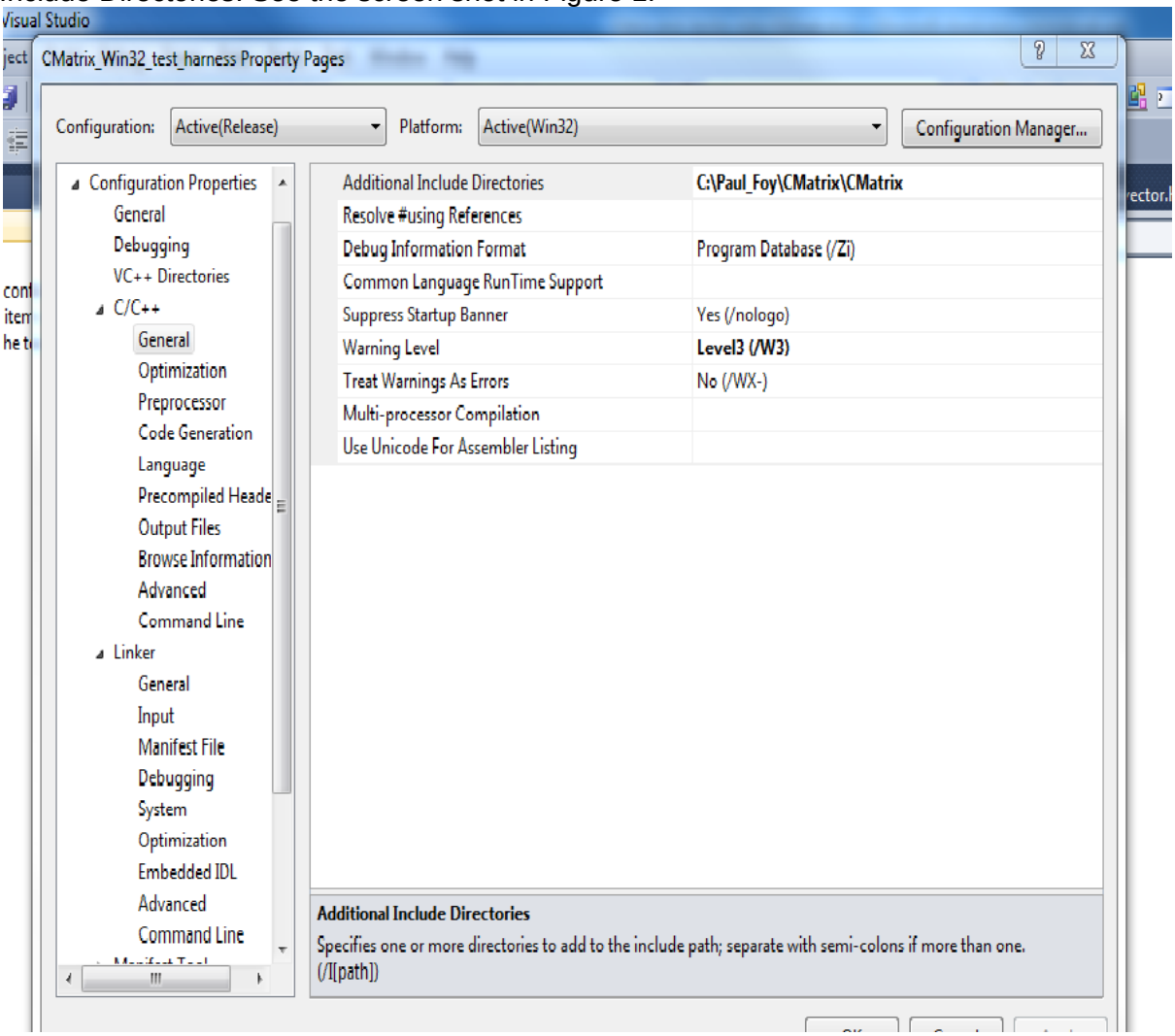


Figure 1 - Adding an additional include directory in Visual Studio 2010

- In order for the compiler/linker to resolve the CMatrix classes and functions it will be necessary to add the library file CMatrix.lib to the compilation process. In Visual Studio this is done in: Project Properties -> Configuration Properties -> Linker -> Input -> Additional Dependencies. In additional the location of the CMatrix.lib library file must be specified. This is done in: Project Properties -> configuration Properties -> Linker -> General -> Additional Library directories.
- CMatrix.dll must be present in the directory of your executable at run time.

The classes in this dll are hidden behind the namespace ‘MathematicalServices’. Hence it will be necessary to use a statement such as

```
using namespace MathematicalServices;
```

in your application files (or more local `using` declaration).

3.2 Explicit Linking

In this way of working it is not necessary to link with CMatrix.lib at compile time. Instead the dll CMatrix.dll is loaded when needed and its member functions accessed. The header file CMatrix.h is also needed.

To link in this way it is necessary to use the exported functions which return a pointer to the classes. The member functions of the class are then accessed via this pointer. These functions have prototypes:

- `CVector * CreateCVectorClassInstance(unsigned int)`
- `CMatrix * CreateCMatrixClassInstance(unsigned int, unsigned int)`
- `CComplex * CreateCComplexClassInstance(double, double)`
- `CComplexVector * CreateCComplexVectorClassInstance(unsigned int)`
- `CComplexMatrix * CreateCComplexMatrixClassInstance(unsigned int, unsigned int)`

Examples of explicit linking are given in sections 5.4.42 and 8.4.35 and elsewhere. Static functions, which can not be accessed by the class pointer, are exported via the friendly names shown in the description of the static functions of each class.

4 Vector class

4.1 Constructors

4.1.1 CVector()

This default constructor initialises a 3-dimensional vector and sets all of its elements to zero. The index of the array of elements is zero based.

C++ example:

```
CVector v = CVector();
```

4.1.2 CVector(int n)

This constructor initialises a vector of dimension n and sets all of its elements to zero. The index of the array of elements is zero based.

C++ example:

```
CVector v = CVector(4);
```

4.1.3 CVector(const CVector &v)

The copy constructor is used to

- Initialize an object from another of the same type.
- Copy an object to pass it as an argument to a function.
- Copy an object to return it from a function.

C++ example:

```
CVector v = CVector(4);
CVector v2 = v;
```

4.2 Overloaded operators

Because this package targets embedded applications as well as windows applications the operations here produce no error message. Instead an inappropriate operation will default to that of a well-defined one in a deterministic way. The way this is done is described for each operation. It is the user's responsibility to ensure the arguments of operations are appropriate.

4.2.1 + operator

If v_1 and v_2 are vectors this permits the expression $v_1 + v_2$ to return a vector. If v_1 and v_2 are not of the same dimension the addition of elements occurs up to the minimum dimension of v_1 and v_2.

C++ example:

```
CVector v = v1 + v2;
```

4.2.2 – operator

If v_1 and v_2 are vectors this permits the expression $v_1 - v_2$ to return a vector. . If v_1 and v_2 are not of the same dimension the subtraction of elements occurs up to the manimum dimension of v_1 and v_2.

C++ example:

```
CVector v = v1 – v2;
```

4.2.3 * operator

If v is a vector and λ a scalar this permits both the expression $\lambda * v$ and $v * \lambda$ to return a vector. The multiplication of the vector by the scalar is component-wise.

If $v1$ and $v2$ are vectors this operator permits the component-wise multiplication of $v1$ and $v2$.

C++ example:

```
double lambda1 = 1.4;
double lambda2 = 0.6;
CVector v = (lambda1 * v1) * lambda2;
```

4.2.4 operator /

If v is a vector and λ a scalar this permits the expression v / λ to return a vector. If λ is zero the original vector v is returned. The division of the vector by the scalar is component-wise.

If $v1$ and $v2$ are vectors this operator permits the component-wise division of $v1$ and $v2$.

C++ example:

```
double lambda = 2.0;
CVector v = v1 / lambda;
```

4.2.5 [i] operator

This operator permits indexing of the elements of the vector (0 based). If i is greater than or equal to the dimension of the vector, n, the index used is $n-1$

C++ example:

```
CVector v = CVactor(4);
double d;
v[0] = 1.0; v[1] = 2.0;
d = v[1];
```

4.2.6 = operator

This permits one vector to be set equal to another. The elements of the vector are copied in a componentwise manner. If the dimension of source vector does not equal the dimension of that of the target de/reallocation of memory occurs.

C++ example:

```
CVector v1 = CVector(4);
v1[0] = 1.0;
CVector v2 = CVector(4);
v2 = v1;
```

4.3 Functions

4.3.1 CComplexVector complex() const

Return the vector as a complex vector:

C++ example:

```
CVector v = CVector(3);
CComplexVector vc = v.complex();
```

4.3.2 CVector concat(const CVector & *rhs*)

Returns a vector which is the concatenation of the elements of the parent vector with those of rhs.

C++ example:

```
CVector v = CVector(3);
v[0] = 1.0;
v[1] = 3.0;
v[2] = 2.0;

CVector cc = v.concat(v);
```

4.3.3 CVector Convolution(CVector v) const

Returns the vector which is the convolution of the parent with v.

If u is the parent vector of length m and v has length n then this method returns the vector w of length $m+n-1$ whose i th element is:

$$w[i] = \sum_j u[j]v[i-j+1]$$

Here the summation is over all values of j which give rise to legal subscripts for $u[j]$ and $v[i-j+1]$.

C++ example:

```
h = f.Convolution(g);
```

4.3.4 double dot (const CVector & v)

Returns the dot product with another vector v. If v is not of the same dimension as the parent, componentwise addition occurs up to the minimum of the two dimensions.

C++ example:

```
if ( cos( v1.dot(v2) ) > 0.99999 )
{
   // vectors are parallel
}
```

4.3.5 void erase(const unsigned int loc)

A function to erase the element at location loc. The vector is redimensioned. If loc is greater than or equal to the dimension of the vector the last element is deleted.

C++ example:

```
CVector v = CVector(3);
v[0] = 1.0;
v[1] = 3.0;
v[2] = 2.0;

v.erase(2);
```

4.3.6 CVector GaussVector() const

Given an n dimensional vector x, this method returns an n dimensional vector v with the property that:

i. $v[0] = 1.0$
ii. $v[i] = -x[i]/x[0]$ for $i = 1$ to $n-1$

If $x[0]$ is zero an error is thrown.

C++ example:

```
CVector x = CVector(5);
For (unsigned int i = 0; i < x.Dim; i++)
{
  x[i] = i + 1.0;
}
CVector v = x.GaussVector();
```

4.3.7 unsigned int GetSize()

Returns the dimension of the vector.

C++ example:

```
unsigned int dim = v.GetSize()
```

4.3.8 double * GetData()

Returns a pointer to the data of the vector.

C++ example:

```
double *data = v.GetData()
```

4.3.9 CVector Householder_vector()

Given an n dimensional vector x this method returns an n dimensional vector v with the property that:

i. $v[0] = 1.0$

ii. $\left(I - 2\frac{vv^T}{v^T v} \right) x$ is zero in all but the first component (here I_n is the n by n identity matrix)

C++ example:

```
CVector x = new CVector(5);
For (int i = 0; i < x.dim; i++)
{
  x[i] = i + 1.0;
}
CVector v = x.Householder_vector();
```

4.3.10 void insert(const unsigned int loc, const double d)

Inserts d at location loc. The vector is redimensioned. If loc is greater than or equal to the dimension of the vector, then d is inserted at the end.

C++ example:

```
CVector v = CVector(3);
v[0] = 1.0;
v[1] = 3.0;
v[2] = 2.0;

v.erase(2);
v.insert(2,2.0);
```

4.3.11 CMatrix matrix()

This returns an n by 1 matrix from an n dimensional vector. This method is provided so that a vector can be used in matrix computations such as in ii of 4.3.9.

C++ example:

```
CVector x = new CVector(5);
For (int i = 0; i < x.dim; i++)
{
  x[i] = i + 1.0;
}
CVector v = x.Householder_vector();
CMatrix P = CMatrix::Identity(5)-   ...
2.0*v.matrix()*(v.matrix().transpose())/v.dot(v);
```

4.3.12 double modulus()

Returns the modulus (2-norm) of the vector.

C++ example:

```
double d = v.modulus();
```

4.3.13 CVector qsort()

Returns a vector whose elements are those of the parent vector sorted in terms of descending size. The algorithm used is the quicksort algorithm.

C++ example:

```
CVector unsorted = CVector(9);
unsorted[0]=3;unsorted[1]=7;unsorted[2]=8;
unsorted[3]=5;unsorted[4]=2;unsorted[5]=1;
unsorted[6]=9;unsorted[7]=5;unsorted[8]=4;

CVector sorted = unsorted.qsort();
// sorted contains 9,8,7,5,5,4,3,2,1
```

4.3.14 SetSubVector(unsigned int i_1,unsigned int i_2,CVector v)

Replaces the elements in the parent between the indices specified with those of vector v. If i_1 or i_2 are greater or equal to the dimension of the vector then they are set to one less than this dimension. If $i_1 > i_2$ then i_1 is set equal to i_2 .If $\dim(v) \neq (i_2 - i_1 + 1)$ then the minimum of these two quantities is used to set the parent starting from i_1

C++ example:

```
v1.SetSubVector(0,2,v2);
```

4.3.15 CVector sort(int [ ] *location*)

This routine performs a bubble sort to return the elements of the vector sorted with respect to size - with the largest element the first element in the sorted vector. Due regard is taken to the sign of the elements. The new sorted location of each original element is held in *location* (zero based)

C++ example:

```
//Order the singular values so that the largest appears first along the diagonal.
```

```
CVector diag = CVector(S.GetDiagonal());
Unsigned int * loc   = new unsigned int[cols];
CVector sort = CVector(cols);
sort         = diag.sort(loc);
S.SetDiagonal(sort);
CMatrix tempU  = U;
CMatrix tempV  = V;

for (int i = 0; i < cols; i++)
{
   U.SetColumnVector(loc[i],tempU.GetColumnVector(i));
   V.SetColumnVector(loc[i],tempV.GetColumnVector(i));
}
```

4.3.16 CVector SubVector(int i_1,int i_2)

This returns the vector between indices i_1 and i_2 inclusive of the parent v. If i_1 or i_2 are greater or equal to the dimension of the vector then they are set to one less than this dimension. If $i_1 > i_2$ then i_1 is set equal to i_2.

C++ example:

```
CVector v1 = v2.Subvector(0,2);
```

4.3.17 void Swap(unsigned int i , unsigned int j)

Interchanges elements i and j.

C++ example:

```
CVector v1 = CVector(4);
...
v1.Swap(0,3);
```

4.3.18 Matrix transpose()

This method returns the vector as a 1 by n matrix.

C++ example:

```
CVector v2 = new CVector(v1);
CMatrix A = v2.transpose();
```

4.3.19 CVector unit_vector() const

Returns the unit vector corresponding to the vector. If the parent is the zero vector then the parent is returned.

C++ example:

```
CVector v2 = v.unit_vector();
```

4.3.20 CVector vector_cos() const

Returns a vector whose elements are the cosine of the elements of the parent vector.

C++ example:

```
CVector vcos = CVector(3);
vcos = v.vector_cos();
```

4.3.21 CVector vector_exp() const

Returns a vector whose elements are the exponential of the elements of the parent vector.

C++ example:

```
CVector vexp = CVector(3);
vexp = v.vector_exp();
```

4.3.22 double vector_max() const

Returns the maximum of the elements of the vector.

C++ example:

```
double vmax;
vmax = v.vector_max();
```

4.3.23 double vector_maxL(unsigned int & *loc*)

Returns the maximum of the elements of the vector. *loc* contains the location at which this occurs.

C++ example:

```
double vmax;
unsigned int loc;
vmax = v.vector_max(loc);
```

4.3.24 double vector_mean() const

Returns the average of the elements of the vector.

C++ example:

```
double vmean;
vmean = v.vector_mean();
```

4.3.25 double vector_min() const

Returns the minimum of the elements of the vector.

C++ example:

```
double vmin;
vmin = v.vector_min();
```

4.3.26 double vector_min(unsigned int *loc*)

Returns the minimum of the elements of the vector. *loc* contains the location at which this occurs.

C++ example:

```
double vmin;
unsigned int loc;
vmin = v.vector_min(loc);
```

4.3.27 CVector vector_sin() const

Returns a vector whose elements are the sine of the elements of the parent vector.

C++ example:

```
CVector vsin = CVector(3);
vsin = v.vector_sin();
```

5 Matrix class

5.1 Constructors

5.1.1 CMatrix()

The default constructor initialises a 3-by-3 matrix each element of which is zero. Rows are represented by the first index and columns by the second index. Both indices are zero based.

C++ example:

```
CMatrix A = CMatrix();
```

5.1.2 CMatrix(int m,int n)

This constructor initialises an m by n matrix each element of which is zero.

C++ example:

```
CMatrix A = CMatrix(3,4);
```

5.1.3 CMatrix(cont CMatrix & A)

The copy constructor is used to

- Initialize an object from another of the same type.
- Copy an object to pass it as an argument to a function.
- Copy an object to return it from a function.

C++ example:

```
CMatrix A = CMatrix(2,3);

CMatrix B = A;
```

5.2 Static member functions

5.2.1 Void Calculate_Givens_parameters(double a,double b,double & c ,double & s)

Given scalars a and b this function computes c and s (where c and s are related by $c = \cos(\theta)$, $s = \sin(\theta)$ for some θ) such that:

$$\begin{pmatrix} c & s \\ -s & c \end{pmatrix}^T \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} r \\ 0 \end{pmatrix}, \text{ for some scalar } r.$$

Friendly name export prototype, for explicit linking: `GetGivensParameters(double, double, double &,double &)`

5.2.2 static CMatrix Eye(unsigned int m ,unsigned int n)

5.2.3 static void Eye2(unsigned int m , unsigned int n , CMatrix & $result$)

A static function which returns a matrix such that:

$Eye(i, j) = 0$, for $i \neq j$

$Eye(i, i) = 1$

Friendly name export prototype, for explicit linking: `GetEye(int , int, CMatrix &)`

C++ example:

```
CMatrix A = CMatrix::Eye(4,3);
```

5.2.4 static CMatrix Identity(unsigned int m)

5.2.5 static void Identity2(unsigned int m , CMatrix & $result$)

A function which returns the m -by- m identity matrix.

Friendly name export prototype, for explicit linking: `GetIdentity(unsigned int, CMatrix &)`.

C++ example:

- Implicit linking

```
CMatrix A = CMatrix::Identity(4);
```

- Explicit linking

```
typedef CMatrix * (*pvFunctv)(unsigned int,unsigned int);
pvFunctv CreateMatrix;
CreateMatrix = ( pvFunctv ) (GetProcAddress(hdll,
"CreateCMatrixClassInstance") );
CMatrix * U = (CMatrix *) CreateMatrix(3,3);

//Set to the identity matrices
typedef void (*fnuptr)(unsigned int, CMatrix &);
fnuptr myId;
myId = (fnuptr) GetProcAddress(hdll, "GetIdentity" );
myId(3, *U);

U->View();
```

5.2.6 Static unsigned int max_no_loops()

Gets the maximum number of iterations that may be performed in the SVD or other iterative calculations.

Friendly name export prototype, for explicit linking: `GetMaxNumberOfLoops()`.

C++ example:

```
if (no_loops > CMatrix::max_no_loops() )
...
```

5.2.7 Static void max_no_loops(unsigned int *niter*)

Sets *niter* as the maximum number of iterations that may be performed in the SVD or other itterative calculations. The default is 10000.

Friendly name export prototype, for explicit linking: `SetMaxNumberOfLoops(unsigned int)`

C++ example:

```
CMatrix::max_no_loops(200);
```

5.2.8 Static double zero_tolerance()

Gets the value below which a number is considered to be zero for the purposes of numerical computations.

Friendly name export prototype, for explicit linking: `GetZeroTolerance()`

C++ example:

```
if (sing[rank] < CMatrix::zero_tolerance())
...
```

5.2.9 Static void zero_tolerance(double *tol*)

Sets *tol* as the value below which a number is considered to be zero for the purposes of numerical computations. The value is a positive number greater than zero. Decreasing this value will mean that, for example, the SVD algorithm takes more iterations to converge. The default is 1.0E-14.

Friendly name export prototype, for explicit linking: `SetZeroTolerance(double)`.

C++ example:

- Implicit linking

```
CMatrix::zero_tolerance(1.0E-8);
```

- Explicit linking

```
typedef void (*fn) (double);
fn myfn = NULL;
myfn = (fn) GetProcAddress(m_hdll, "SetZeroTolerance" );
myfn(tol);
```

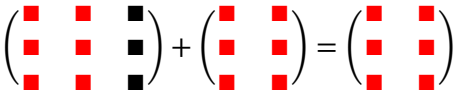
5.3 Overloaded operators

As with the Vector class the operations here produce no error message. Instead an inappropriate operation will default to that of a well-defined one in a deterministic way. The way this is done is described for each operation. It is the user's responsibility to ensure the arguments of operations are appropriate.

5.3.1 + operator

If A_1 and A_2 are matrices this operator permits $A_1 + A_2$ to return a matrix. If A_1 and A_2 are not of the same dimension component wise operations are performed up to the minimum of the corresponding dimension of A_1 and A_2. For example the diagram below illustrates how the

component-wise addition occurs when the number of columns in the two matrices are not equal:



C++ example:

```
CMatrix A = A1 + A1
```

5.3.2 – operator

If A_1 and A_2 are matrices this operator permits $A_1 - A_2$ to return a matrix. If A_1 and A_2 are not of the same dimension component wise operations are performed up to the minimum of the corresponding dimension of A_1 and A_2.

C++ example:

```
CMatrix A = A1 - A2;
```

5.3.3 * operator

- If A is a matrix and λ is a scalar this operator permits both $\lambda * A$ and $A * \lambda$ to return a matrix. The multiplication of the matrix by the scalar is component-wise.
- If A_1 and A_2 are two matrices of compatible dimensions for matrix multiplication this operator permits $A_1 * A_2$ to return a matrix. If the dimensions are not compatible then the common dimension is taken to be the minimum of the number of columns of A_1 and the number of rows of A_2 The multiplication then proceeds using this common dimension..
- If A is a matrix and v a vector this operator permits $A * v$ to return a vector. If A and v are not of compatible dimensions then the common dimension is taken to be the minimum of the number of columns of A and the dimension of v. The multiplication then proceeds using this common dimension.

C++ example:

```
CMatrix CA = CMatrix::Identity();
CMatrix B = (lambda1 * A) * lambda2;
CMatrix C = A*B;
CVector v = CVector(3);
v[0]=1.0; v[1] = 2.0;
CVector w = A*v;
```

5.3.4 Operator /

If A is a matrix and λ a scalar this operations permits the return of the matrix A/λ. The division is component wise. If λ is zero then A is returned.

C++ example:

```
CMatrix A = B / 2.5;
```

5.3.5 = operator

This permits one matrix to be set equal to another. The elements of the matrix are copied in a component-wise manner. If the dimension of source matrix does not equal the dimension of that of the target. De/reallocation of memory occurs.

C++ example:

```
CMatrix m1 = CMatrix(4,3);
m1[0,0] = 1.0;
CMatrix m2 = CMatrix(4,3);
m2 = m1;
```

5.4 Functions

5.4.1 Bidiagonalization(Matrix & U ,Matrix & V)

If A is the m -by- n matrix, $(m \geq n)$, this method overwrites A with $B = U^T AV$, where B is upper bi-diagonal, U is an m -by - m orthogonal matrix, and V is an n - by - n orthogonal matrix. A can be recovered by calculating UBV^T . The zeroing of rows and columns is performed by a succession of Givens transformations.

C++ example:

```
CMatrix A = CMatrix(4,3);
CMatrix U = CMatrix::Identity(4);
CMatrix V = CMatrix::Identity(3);
A.Bidiagonalisation(U,V);
```

5.4.2 Bidiagonalization()

If A is an m by n matrix, $(m \geq n)$, this algorithm overwrites A with the same upper bidiagonal matrix as in 5.4.1, but without accumulating U and V .

C++ example:

```
CMatrix A = CMatrix(4,3);
A.Bidiagonalisation();
```

5.4.3 Column_householder(Vector v)

Given the m -by - n matrix A and a non-zero n -vector v with $v[0] = 1.0$,this method overwrites A with AP , where $P = I - 2\frac{vv^T}{v^T v}$. The calculations are performed in a computationally efficient manner.

5.4.4 Double Determinant(unsigned int & *converged*)

Calculates the determinant of a square matrix. If the matrix is non-square, the determinant of the square matrix whose dimension is the minimum of the number of rows and columns of the matrix is returned. If the underlying SVD routine has converged, *converged* is set to 1. Otherwise it is set to 0.

C++ example:

```
int conv = 1;
double det = A.Determinant(conv);
```

5.4.5 unsigned int Eigenvalues(CComplexVector & *Eval*)

If A is the n by n parent matrix this routine calculates the n complex numbers, λ, which together with n vectors, v ,satisfy the equation $Av = \lambda v$ .The eigenvalues λ are returned in the complex vector *Eval* . The eigenvalues are sorted in terms of magnitude with the largest first. If the underlying numerical routine used has not converged *convergence* is set to zero, otherwise it is set to one. The routine used is a QR algorithm employing a Francis QR step. This should converge in most cases. The convergence criteria is determind by C*Matrix::zero_tolerance()* and the maximum number of steps in the iteration by C*Matrix::max_number_loops().* It may be possible to allow the routine to converge by increasing one or both of these parameters.

If the matrix is not square an error is thrown.

C++ example:

```
CMatrix Eval = CMatrix(7,7);
...
unsigned int Eig_return = 1;
Eig_return = A.Eigenvalues(Eval);
```

5.4.6 unsigned int Eigenvectors(CComplexVector *Eval* , CComplexMatrix *Evec*)

If A is the n by n parent matrix this routine calculates the n Eigenvalues, λ, which together with the n Eigenvectors v, satisy the equation $Av = \lambda v$. The Eigenvalues are returned as a complex vector in *Eval* . The corresponding Eigenvectors are returned as columns of the complex matrix *Evec*. The Eigenvectors and their corresponding Eigenvalues are sorted in order of magnitude, with the largest first. If the underlying numerical routine has not converged, or there is some kind of problem with the result, zero is returned, otherwise one is returned. The convergence criteria is determind by C*Matrix::zero_tolerance()* and the maximum number of steps in the iteration by C*Matrix::max_number_loops().*

The routine uses a QR algorithm employing a Francis QR step. This should converge in most cases. Convergence will be problematic in the case of an orthogonal matrix.

C++ example

```
Eig_return = A.Eigenvectors(E,P);
if (Eig_return == 1)
{
    for (i = 0; i < n; i++)
    {
        CComplexVector diff = CComplexVector(n);
        diff = E[i] * P.GetColumnVector(i) - A *
```

```
P.GetColumnVector(i);
        if (diff.Modulus > (((double)n)*1.0E-8))
        {
            MessageBox.Show("Complex Eigenvector routine has
failed");
            return;
        }
    }
}
```

5.4.7 void ElementaryRowOperation(unsigned int i, unsigned int j)

Swaps rows i and j.

C++ example:

```
CMatrix U = CMatrix (6,5);
...
U.ElementaryRowOperation(k, imax);
```

5.4.8 GaussPreMultiplication(unsigned int k, CVector & g)

Let A be the m by n parent matrix. And let g be a Gauss vector of dimension $m-k$ .Let $L(k)$ be the lower triangular m by m matrix of the form:

$$\begin{pmatrix} 1 & 0 & 0 \\ \dots & & \\ & 1 & \\ & g[1] & \\ & \dots & \\ 0 & g[n-1] & 1 \end{pmatrix}$$, where the Gauss vector occurs in the kth column.

Then this function sets A to $L(k)*A$, in a computationally efficient manner.

C++ example:

```
//Get Gauss vector
CVector gauss = matrixU.GetColumnVector(k).SubVector(k,GetRowSize()
- 1).GaussVector();

//Perform left Gauss operation to update U
matrixU.GaussPreMultiplication(k,gauss);
```

5.4.9 unsigned int Gauss_Seidel(const CVector & b ,const CVector & xo ,CVector & x)

If A is the parent matrix this routine uses the Gauss-Seidel iterative method to solve the linear system $Ax=b$, starting with the initial estimate of xo. If the system converges successfully 1 is returned. If the system fails to converge after *max_no_loops* iterations 0 is returned. If x_k is the estimate at the k th step the algorithm is deemed to have converged

when $\|x_k - x_{k-1}\| < zero\_tolerance$. The algorithm assumes that A is non singular. If a pivot element ($A_{i,i}$) is zero then the SVD approach is used to obtain the solution (in a best fit sense).

C++ example:

See: 5.4.28.

5.4.10 CVector GetColumnVector(unsigned int i)

Returns the i th column of the matrix as a vector. If i is greater or equal to the number of columns then the last column is returned.

C++ example:

```
CMatrix A = CMatrix::Identity(4);
CVector v = A.GetColumnVector(1);
```

5.4.11 unsigned int GetColumnSize()

Returns the number of columns in the matrix.

5.4.12 CVector GetDiagonal()

Returns the vector containing the diagonal entries of the matrix. The matrix does not have to be square.

5.4.13 unsigned int GetRowSize()

Returns the number of rows in the matrix.

5.4.14 CVector GetRowVector(unsigned int i)

Returns the i th row of the matrix as a vector. If i is less greater or equal to the number of rows then the last row is returned.

5.4.15 Givens_post_multiplication(unsigned int i ,unsigned int k ,double c ,double s)

Let $G(i,k,c,s)$ be the Givens matrix, where c and s are calculated according to 5.2.1. This is the identity except that

$$G(i,k,c,s)(i,i) = c$$
$$G(i,k,c,s)(k,k) = c$$
$$G(i,k,c,s)(i,k) = s$$
$$G(i,k,c,s)(k,i) = -s$$

The Givens post multiplication of a matrix A replaces A with $AG(i,k,c,s)$.

5.4.16 Givens_pre_multiplication(unsigned int i,unsigned int k,double c ,double s)

Let $G(i,k,c,s)$ be the Givens matrix, where c and s are calculated according to 5.2.1. This is the identity except that

$$G(i,k,c,s)(i,i) = c$$
$$G(i,k,c,s)(k,k) = c$$
$$G(i,k,c,s)(i,k) = s$$
$$G(i,k,c,s)(k,i) = -s$$

The Givens pre-multiplication of a matrix A replaces A with $G(i,k,c,s)^T A$.

5.4.17 Golub_Kahan_SVD_step2(CMatrix & U ,Cmatrix & V)

Given an m by n bidiagonal matrix B with no zeros on its diagonal or superdiagonal this algorithm overwrites B with the bidiagonal matrix $B\_ = U^T BV$, where U is an m by m orthogonal matrix and V is an n by n orthogonal matrix. The method chases the zero down the upper diagonal the first operation employing the Wilkinson shift (see [1]).

C++ example:

```
unsigned int no_loops = 0;
CMatrix T = A;
while ( T.is_diagonal() == false )
{
    CMatrix UB = CMatrix::Identity(m);
    CMatrix VB = CMatrix::Identity(n);
    T.Golub_Kahan_SVD_step2(UB,VB);
    U = (U*UB);
    V = (V*VB);
    no_loops++;
    if (no_loops > CMatrix::max_no_SVD_loops())
    {
        MessageBox(hWnd,L"Kolub Kahan step has failed (too many
loops)",L"ErrorMessage",0);
            return;
    }
}
```

5.4.18 void HessenbergReduction()

Operates on the parent matrix to reduce it to upper Hessenberg form. The reduction is via a series of Givens transformations.

C++ example:

```
A.HessenbergReduction();
```

5.4.19 void HessenbergReduction(CMatrix & Q)

This method operates on the parent matrix, A, to reduce it to upper Hessenberg form H. The reduction is done using a series of Givens transformations. A unitary matrix, Q, is formed such that $Q^H AQ = H$.

C++ example:

```
CMatrix A = CMatrix(7,7);
...
CMatrix Q = CMatrix::Identity(4);
A.HessenbergReduction(Q);
```

5.4.20 Householder_bidiagonalisation(CMatrix & U ,CMatrix & V)

If A is the m -by- n matrix, $(m \geq n)$, this method overwrites A with $B = U^T AV$, where B is upper bi-diagonal, U is an m -by - m orthogonal matrix, and V is an n - by - n orthogonal matrix. A can be recovered by calculating UBV^T . The zeroing of rows and columns is performed by a succession of Householder transformations.

5.4.21 Householder_QR_decomposition(Matrix & Q , matrix & R)

Given the m by n matrix A , this method computes an m by m orthogonal matrix Q and an m by n upper triangular matrix R such that $A = QR$. The zeroing of columns is performed by a succession of Housholder transformations.

5.4.22 unsigned int Inverse(CMatrix inv)

Calculates inv , the inverse of the square matrix.

The return value of the function indicates the following:

Return value	Meaning
0	Successful calculation of the inverse.
1	The matrix is non-square - the original matrix is returned.
2	The numerical method would not converge on a solution - the original matrix is returned.
3	The matrix is singular (i.e. an inverse doesn't exist) - the original matrix is returned.

C++ example:

```
unsigned int inverse_calculated = 0;
inverse_calculated = A.Inverse(inv);
```

5.4.23 bool is_bidiagonal()

Returns true if the matrix is upper bi-diagonal, to within the tolerance given by CMatrix::zero_tolerance(). Otherwise the function returns false.

As examples of bi-diagonal matrices we have (x represents a (in general) non-zero value)):

A 3 by 3 bi-diagonal matrix: $\begin{pmatrix} x & x & 0 \\ 0 & x & x \\ 0 & 0 & x \end{pmatrix}$.

A 4 by 3 bi-diagonal matrix: $\begin{pmatrix} x & x & 0 \\ 0 & x & x \\ 0 & 0 & x \\ 0 & 0 & 0 \end{pmatrix}$.

A 3 by 4 bi-diagonal matrix: $\begin{pmatrix} x & x & 0 & 0 \\ 0 & x & x & 0 \\ 0 & 0 & x & 0 \end{pmatrix}$.

5.4.24 bool is_diagonal()

Returns true if the matrix is diagonal to within the tolerance given by CMatrix::zero_tolerance(). Otherwise the function returns false.

Examples of diagonal matrices are (x represents a non-zero value):

A 3 by 3 diagonal matrix: $\begin{pmatrix} x & 0 & 0 \\ 0 & x & 0 \\ 0 & 0 & x \end{pmatrix}$.

A 4 by 3 diagonal matrix: $\begin{pmatrix} x & 0 & 0 \\ 0 & x & 0 \\ 0 & 0 & x \\ 0 & 0 & 0 \end{pmatrix}$.

A 3 by 4 diagonal matrix: $\begin{pmatrix} x & 0 & 0 & 0 \\ 0 & x & 0 & 0 \\ 0 & 0 & x & 0 \end{pmatrix}$.

5.4.25 bool is_Hessenberg()

Returns true if the matrix is upper Hessenberg - i.e. all entries are zero below the lower sub-diagonal. Otherwise the function returns false.

An example of a 4 by 4 upper Hessenberg matrix: $\begin{pmatrix} x & x & x & x \\ x & x & x & x \\ 0 & x & x & x \\ 0 & 0 & x & x \end{pmatrix}$.

5.4.26 bool is_orthogonal()

Returns true if the columns of the matrix are mutually orthogonal, to within the tolerance given by CMatrix::zero_tolerance(). Otherwise the function returns false. The routine employed is shown by the example below.

C++ example:

```
bool CMatrix::is_orthogonal()
{
```

```
    //Determines whether a matrix has got orthogonal columns
    if ((Transpose() * (*this) -
Matrix::Identity(m_ncols)).is_zero())
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

5.4.27 boolean is_zero()

Return true if all entries in the matrix are zero to within the tolerance given by CMatrix::zero_tolerance(). Otherwise the function returns false.

5.4.28 unsigned int Jacobi(const CVector & b ,const CVector & xo , CVector & x)

If A is the parent matrix this routine uses the Jacobi iterative method to solve the linear system $Ax = b$, starting with the initial estimate of xo. If the system converges successfully 1 is returned. If the system fails to converge after *max_no_loops* iterations 0 is returned. If x_k is the estimate at the k th step the algorithm is deemed to have converged when $\|x_k - x_{k-1}\| < zero\_tolerance$. The algorithm assumes that A is non singular. If a pivot element ($A_{i,i}$) is zero then the SVD approach is used to obtain the solution (in a best fit sense).

C++ example:

```
CMatrix A = CMatrix(4,4);;
CVector b = CVector(4);
CVector x = CVector(4);
CVector xo = CVector(4);
double w = 1.25;
unsigned int conv;

A(0,0) = 10.0; A(0,1) = -1.0; A(0,2) = 2.0; A(0,3) = 0.0;
A(1, 0) = -1.0; A(1, 1) = 11.0; A(1, 2) = -1.0; A(1, 3) = 3.0;
A(2, 0) = 2.0; A(2, 1) = -1.0; A(2, 2) = 10.0; A(2, 3) = -1.0;
A(3, 0) = 0.0; A(3, 1) = 3.0; A(3, 2) = -1.0; A(3, 3) = 8.0;

b[0] = 6.0; b[1] = 25.0; b[2] = -11.0; b[3] = 15.0;

CMatrix::max_no_loops(100);
CMatrix::zero_tolerance(1.0E-8);

conv = A.Jacobi(b,xo,x); // should be [1,2,-1,1]
//conv = A.Gauss_Seidel(b,xo,x);
```

```
//conv = A.SOR(b,xo,w,x);
```

5.4.29 void LUDecomposition(CMatrix & L, CMatrix U, CMatrix & P)

Given the m by n parent matrix A, this decomposition calculates an upper diagonal m by n matrix U, a lower diagonal m by m matrix L with ones on the main diagonal, and an m by m permutation matrix P such that $PA = LU$. The method is successful regardless of the rank deficiency, or not, of A.

C++ example:

```
CMatrix A = Matrix(4,4);;
CMatrix P = CMatrix.Identity(4);
CMatrix L = CMatrix.Identity(4);
CMatrix U = CMatrix(4,4);

A[0,0] = 10.0; A[0,1] = -1.0; A[0,2] = 2.0; A[0,3] = 0.0;
A[1, 0] = -1.0; A[1, 1] = 11.0; A[1, 2] = -1.0; A[1, 3] = 3.0;
A[2, 0] = 2.0; A[2, 1] = -1.0; A[2, 2] = 10.0; A[2, 3] = -1.0;
A[3, 0] = 0.0; A[3, 1] = 3.0; A[3, 2] = -1.0; A[3, 3] = 8.0;

CMatrix::ZeroTolerance() = 1.0E-10;


A.LUDecomposition(L,U,P)
CMatrix D = P*A-L*U; //D should be zero
```

5.4.30 Matrix Minor(unsigned int i,unsigned int j)

Returns the matrix which is the minor at the (i, j) station. That is, the matrix whose elements are those of the original matrix but with the i th row and j th column removed.

C++ example:

See the code corresponding to the Determinant function.

5.4.31 QR_decomposition(Matrix & Q, matrix & R)

Given the m by n matrix A, this method computes an m by m orthogonal matrix Q and an m by n upper triangular matrix R such that $A = QR$. The zeroing of columns is performed by a succession of Givens transformations.

C++ example:

```
CMatrix A = CMatrix(3,3);
A(0,0)=1.0;A(0,1)=2.0;A(0,2)=3.0;
A(1,0)=2.0;A(1,1)=1.0;A(1,2)=5.0;
A(2,0)=6.0;A(2,1)=5.0;A(2,2)=1.0;
CMatrix U = CMatrix::Identity(3);
CMatrix V = CMatrix::Identity(3);
A.QR_decomposition(Q,R);
```

```
CMatrix B = A - Q*R; //should be the zero matrix
```

5.4.32 unsigned int rank().

Returns the numerial rank of the matrix. That is the maximum number of linearly independant rows or columns. *CMatrix::zero_tolerance()* is used as the criterion for a singular value being zero, or not, in this determination.

C++ example:

```
int p;
p = min(nrows,ncols);

CVector sing = CVector(p);
unsigned int conv = 0;

conv = singular_values(sing);

unsigned int rank = 0;
for (rank = 0; rank < p; rank++)
{
  if (sing[rank] < CMatrix::zero_tolerance())
  {
    break;
  }
}
```

5.4.33 unsigned int RealSchurDecomposition(CMatrix & Q,CMatrix & T)

This routine calculates an orthogonal matrix Q and a quasi-triangular matrix T, such that $Q^T AQ = T$. A matrix is quasi-triangular if its diagonal blocks are either 1 by 1 or 2 by 2 matrices. All 2 by 2 diagonal blocks have complex eigenvalues.

If the underlying numerical routine has not converged zero is returned, otherwise one is returned.

The routine used is a QR algorithm employing a Francis QR step. This should converge in most cases. The convergence criteria is determind by *CMatrix::zero_tolerance()* and the maximum number of steps in the iteration by *CMatrix::max_no_loops().* It may be possible to allow the routine to converge by increasing one or both of these parameters.

5.4.34 Row_householder(CVector v)

Given the m-by-n matrix A and a non-zero m-vector v with $v[0] = 1.0$,this method overwrites A with PA, where $P = I - 2\frac{vv^T}{v^T v}$. The calculations are performed in a computationally efficient manner.

5.4.35 SetColumnVector(unsigned int i,CVector v)

Sets the ith column of the matrix to have the same elements as the vector v.If i is greater than or equal to the number of columns of the matrix, then the last column is set. If the

dimension of v does not equal the number of rows of the matrix the component-wise writing occurs up to the minimum of the dimension of v and the number of rows.

5.4.36 SetDiagonal(CVector v)

Sets the diagonal entries of a matrix to be those of the elements of v. The matrix does not have to be square. If the dimension of v is does not equal the minimum of the number of rows and columns of the matrix, then component-wise writing occurs up to the minimuim of these two quantities.

5.4.37 SetRowVector(unsigned int i,Vector v)

Sets the i th row of the matrix to have the same elements as the vector v .If i is greater than or equal to the number of rows of the matrix then the last row is set. If the dimension of v does not equal the number of columns of the matrix, then component-wise writing occurs up to the minimum of these two quantities.

5.4.38 SetSubMatrix(unsigne int i_1,unsigned int i_2,unsigned int j_1,unsigned int j_2,CMatrix A)

Sets the elements spanning rows i_1 to i_2 and columns j_1 to j_2 to the elements of the matrix A. j_2. If $i_2 < i_1$ or $j_2 < j_1$ or the i's or j's do not represent valid indices then they are adjusted appropriately. If the dimensions of A are not compatible with the resulting adjusted indices then component-wise writing occurs for the minimum of the corresponding dimensions.

5.4.39 unsigned int singular_values(CVector &s)

Populates a vector containing the p singular values of the matrix ordered in descending order of magnitude. Here, $p = \min(nrows, ncols)$, where $nrows$ and $ncols$ are the number of rows and columns of the matrix respectively. If the underlying iteration converges 1 is returned, otherwise 0 is returned.

5.4.40 unsigned int SOR(const Cvector & b,const CVector & xo,const double w, CVector & x)

If A is the parent matrix this routine uses the SOR iterative method to solve the linear system $Ax = b$, starting with the initial estimate of xo. If the system converges successfully 1 is returned. If the system fails to converge *after CMatrix::max_no_loops()* iterations 0 is returned. If x_k is the estimate at the k th step the algorithm is deemed to have converged when $\|x_k - x_{k-1}\| < zero\_tolerance()$. The algorithm assumes that A is non singular. If a pivot element ($A_{i,i}$) is zero then the SVD approach is used to obtain the solution (in a best fit sense).

C++ example:

See: 5.4.28.

5.4.41 CMatrix SubMatrix(unsigned int i_1,unsigned int i_2,unsigned int j_1 ,unsigned int j_2)

Gets the submatrix with rows spanning i_1 to i_2 and columns spanning j_1 to j_2. If $i_2 < i_1$ or $j_2 < j_1$ or the i's or j's do not represent valid indices they are adjusted appropriately.

5.4.42 unsigned int SVD(CMatrix & U ,CMatrix & S ,CMatrix & V)

5.4.43 unsigned int Jacobi_SVD(CMatrix & U ,CMatrix & S ,CMatrix & V)

Given an m by n matrix , these method computes an m by m orthogonal matrix U , an n by n orthogonal matrix V and an m by n diagonal matrix S such that $A = USV^T$. The symmetry of the decomposition means that both the cases $m \geq n$, and $m < n$ are handled by one routine. If $m >> n$, and it is not required to calculate all the columns of U , and workspace is at a premium, it is recommended that the routine SVD_Light is used for efficiency reasons. If the SVD algorithm converges this function returns 1; otherwise it returns 0.

The diagonal elements of S contain the singular values in descending order of magnitude. The first min(m , n) columns of U contain the corresponding left singular vectors, u_i , and the first min(m , n) columns of V contain the corresponding right singular vectors, v_i , where Av_i = $\sigma_i u_i$ $(1 \leq i \leq \min(m,n))$.

The number of iterations that will be performed in an attempt to converge on the solution is governed by *Matrix::max_no_loops()*. The threshold for a number being zero is given by *Matrix::zero_tolerance()*. If the algorithm does not converge you can always ensure a convergence by increasing *Matrix*::*max_no_loops()* or *Matrix*::*zero_tolerance()* or both.

Jacobi_SVD uses a series of Jacobi transformations to successively zero off-diagonal elements. SVD uses a Golub-Kahan step to reduce the magnitude of off diagonal elements of a bidiagonal matrix. Jacobi_SVD takes many more iterations to zero all off diagonal elements but the amount of computation done at any one iteration is smaller than that of SVD. Jacobi_SVD is generally much slower than SVD.

C++ example:

- Implicit Linking

```
unsigned int converged = 0;
CMatrix S = CMatrix(A);
CMatrix U = CMatrix::Identity(A.GetRowSize());
CMatrix V = CMatrix::Identity(A.GetColumnSize());
converged = A.SVD(U,S,V);
//converged = A.Jacobi_SVD(U,S,V);
CMatrix B = A - U*S*V.Transpose(); //should be the zero matrix
```

- Explicit linking

```
HINSTANCE hdll = NULL;
hdll = LoadLibrary(TEXT("CMatrix.dll"));
typedef CMatrix * (*pvFunctv)(unsigned int,unsigned int);
pvFunctv CreateMatrix;
CreateMatrix = ( pvFunctv ) (GetProcAddress(hdll,
"CreateCMatrixClassInstance") );

CMatrix * mat = ( CMatrix * ) ( CreateMatrix(3,3) );
CMatrix * S = (CMatrix *) CreateMatrix(3,3);
CMatrix * U = (CMatrix *) CreateMatrix(3,3);
CMatrix * V = (CMatrix *) CreateMatrix(3,3);
```

```
unsigned int Nrow = mat->GetRowSize();

mat->SetElement(2,2,1.25);
double dbl = mat->GetElement(2,2);
mat->SVD(*U,*S,*V);

double dblS2 = (*S)(2,2);
double dblS1 = (*S)(1,1);
double dblS0 = (*S)(0,0);

CMatrix * B = (CMatrix *) CreateMatrix(3,3);
CMatrix * B1 = (CMatrix *) CreateMatrix(3,3);
CMatrix * B2 = (CMatrix *) CreateMatrix(3,3);
CMatrix * B3 = (CMatrix *) CreateMatrix(3,3);

V->Transpose(*B1);
S->Multiply(*B1,*B2);
U->Multiply(*B2,*B3);
mat2->Subtract(*B3,*B);

B->View(); //should be zero

delete B3, B2, B1, B, V, U, S, mat2, mat;
```

5.4.44 void SVD2_begin(CMatrix & U ,CMatrix & S ,CMatrix & V)

5.4.45 unsigned int SVD2_iteration(CMatrix & U ,CMatrix & V)

5.4.46 unsigned int Jacobi_SVD2_iteration(CMatrix & U ,CMatrix & V)

5.4.47 void SVD2_end(CMatrix & U ,CMatrix & S ,CMatrix & V)

These four functions are used (typically in embedded environments) where it is necessary to limit the work done on any one pass. The same U, S and V must be used in each function. SVD2_begin is used to initialise the algorithm. SVD2_iteration or Jacobi_SVD2_iteration is then called repeatedly until either convergences is signalled or the maximum number of iterations has been performed. Convergence if signalled by SVD_iteration or Jacobi_SVD_iteration returning 1. Otherwise 0 is returned. Cmatrix::zero_tolerance determines the convergence criteria. SVD2_end is then called to finish the algorithm.

C# example:

```
CMatrix::zero_tolerance(1.0E-10);
max_number_of_iterations =
1000+max(A.GetRowSize(),A.GetColumnSize());

number_of_iterations = 0;
SVD_return = 0;

A.SVD2_begin(U,S,V);

while ((SVD_return == 0) && (number_of_iterations <
max_number_of_iterations))
{
    SVD_return = S.SVD2_iteration(U,V);
```

```
    //SVD_return = S.Jacobi_SVD2_iteration(U,V) - an alternative
    number_of_iterations++;

}

A.SVD2_end(U,S,V);
```

5.4.48 unsigned int SVD_Light(CMatrix U ,CMatrix S ,CMatrix V)

Let the parent A be an m by n matrix. If $m \geq n$, this method computes an m by n matrix U whose columns are orthogonal, and n by n orthogonal matrix V and an n by n diagonal matrix S . If $m < n$, this method computes an m by m orthogonal matrix U , an n by m matrix V whose columns are orthogonal and an m by m diagonal matrix S . In both cases $A = USV^T$.

This routine should be used in preference to SVD in the case that $m >> n$ or $n >> m$ and workspace is at a premium. If the SVD algorithm converges this function returns 1; otherwise it returns 0.

The diagonal elements of S contain the singular values in descending order of magnitude. The first columns of U contain the left singular vectors, u_i , and the columns of V contain the corresponding right singular vectors, v_i , where $Av_i = \sigma_i u_i \ \ (1 \leq i \leq \min(m,n))$.

The number of iterations that will be performed in an attempt to converge on the solution is governed by *Matrix::max_no_loops()*. The threshold for a number being zero is given by *Matrix::zero_tolerance()*. If the algorithm does not converge you can always ensure a convergence by increasing *Matrix*::*max_no_loops()* or *Matrix*::*zero_tolerance()* or both.

C++ example:

```
unsigned int converged = 0;
CMatrix S = CMatrix::Identity(A.GetColumnSize());
CMatrix U = CMatrix(A.GetRowSize(),A.GetColumnSize());
CMatrix V = CMatrix::Identity(A.GetColumnSize());
converged = A.SVD_Light(U,S,V);
CMatrix B = A - U*S*V.Transpose(); //should be the zero matrix
```

5.4.49 CVector SVD_solve(CVector b , unsigned int & $conv$) – paid for version only

Given the m by n matrix A and the m vector b, this method solves (in a least squares sense) for the n vector x ,where $Ax = b$. That is, x minimizes $\|Ax - b\|$ and has the smallest norm of all minimizers. *conv* indicates if the underlying SVD routine converged (*conv* set to 1) or not (0).

C++ example:

```
unsigned int converged = 0;
CVector x = A.SVD_solve(b,converged);
```

5.4.50 CMatrix Transpose()

Returns the transpose of the matrix.

5.4.51 void View()

This method displays a window containing a grid view of the elements of the matrix. This method would be useful during developing or debugging an application.

C++ example:

```
A.View()
```

A screenshot of the view, via this function, of two matrices is shown in Figure 2.

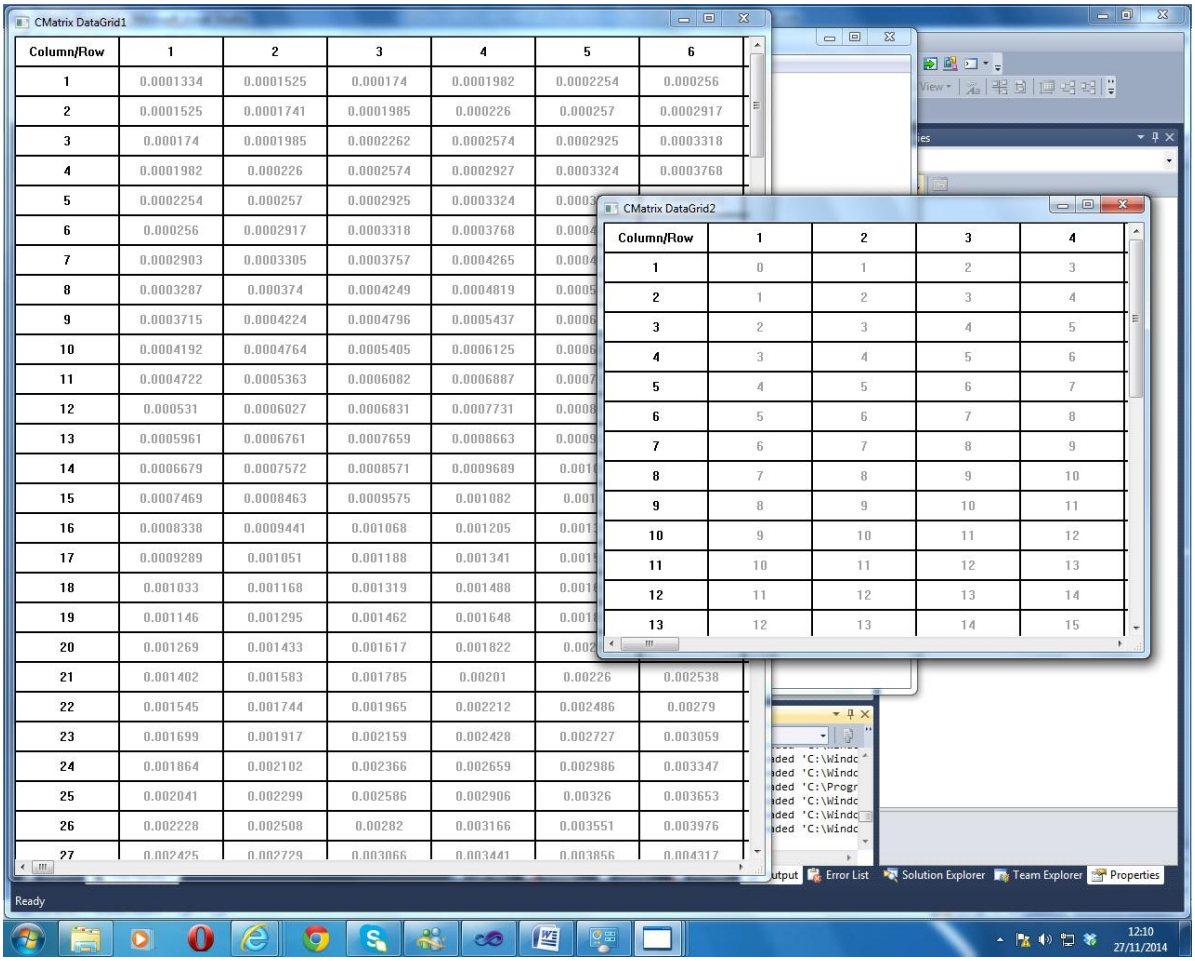


Figure 2 - A Screenshot of two views of a data grid of two matrices.

5.4.52 CComplexMatrix ViewAsComplex() const

Returns a complex matrix whose elements are the same as that of the parent.

C++ example:

```
CMatrix A = CMatrix::Identity(3);

CComplexMatrix C = A.ViewAsComplex();
```

6 Complex Class

6.1 Constructors

6.1.1 CComplex()

Constructs a complex number whose real and imaginary parts are both zero.

C++ example:

```
CComplex c = CComplex();
```

6.1.2 CComplex(const CComplex & c)

Constructs a complex number whose real and imaginary parts are the same as those of c.

C++ example:

```
CComplex c1 = CComplex();
c1.Imaginary = 1.5;
CComplex c2 = CComplex(c1);
```

6.1.3 CComplex(const double $real$,const double $imag$)

Constructs a complex number with real part $real$ and imaginary part $imag$.

C++ example:

```
CComplex c = CComplex(1.0,2.0);
```

6.2 Properties

6.2.1 CComplex Cos()

Returns the cosine of the complex number z. This is the expression $\frac{e^{iz}+e^{-iz}}{2}$. See 6.2.5.

6.2.2 Complex Exp()

Returns the exponential of the complex number. See 6.2.5

6.2.3 double GetReal()

Reads the real part.

6.2.4 double GetImaginary()

Reads the imaginary part.

C++ example:

```
double rl;
double im;
CComplex c = CComplex(c2);
```

```
rl = c.GetReal;
im = c.GetImaginary;
```

6.2.5 double Magnitude() const

Reads the magnitude of the complex number. If (x, y) represents the complex number then its magnitude is $\sqrt{x^2 + y^2}$.

C++ example:

```
double rl = 1.0;
double im = 5.0;
CComplex c = CComplex(rl,im);
double mag = c.Magnitude();
CComplex s = c.Sin;
CComplex e = c.Exp;
CComplex cs = c.Cos;
```

6.2.6 double Phase()

Reads the phase (or argument) of the complex number. This is the angle subtended in the Argand diagram. Phase φ equals $\arctan 2(y, x)$. $-\pi < \varphi \leq \pi$.

C++ example:

```
double rl = 1.0;
double im = 5.0;
CComplex c = CComplex(rl,im);
double ph = c.Phase();
double sq = c.Sqrt();
```

6.2.7 void SetImaginary(const double *imag*)

Writes *imag* as the imaginary part of the number.

C++ example:

```
double im = 5.0;
double rl = 2.0;
CComplex c = CComplex();
c.SetImaginary(im);
c.SetReal(rl);
```

6.2.8 void SetReal(const double *real*)

Writes *real* as the real part of the number.

6.2.9 CComplex Sin()

Returns the sine of the complex number z. This is the expression $\frac{e^{iz} - e^{-iz}}{2i}$. See 6.2.5.

6.2.10 Complex Sqrt()

Returns the principal part of the square root of the complex number z. If $z = x + iy$ ($y \neq 0$), this is the expression $c + id$ where:

$c = \sqrt{\frac{x + \sqrt{x^2 + y^2}}{2}}$, $d = sign(y)\sqrt{\frac{-x + \sqrt{x^2 + y^2}}{2}}$. See 6.2.6.

6.3 Overloaded operators

6.3.1 + operator

If $c1$ and $c2$ are double or complex numbers this operator permits $c1 + c2$ to return a complex number.

C++ example:

```
CComplex c = c1 + c2;
```

6.3.2 - operator

If $c1$ and $c2$ are double or complex numbers this operator permits $c1 - c2$ to return a complex number.

C++ example:

```
CComplex c = c1 - c2;
```

6.3.3 * operator

If $c1$ and $c2$ are complex numbers and λ is a scalar real number this operator permits $c1 * \lambda$, $\lambda * c2$ and $c1 * c2$ to return complex numbers.

C++ example:

```
double lambda;
CComplex c = c1*c2 + lambda*c2;
```

6.3.4 operator /

If $c1$ and $c2$ are complex numbers and λ is a scalar real number this operator permits $c1 / \lambda$, $\lambda / c2$ and $c1 / c2$ to return a complex number. If λ or $c2$ are zero, when acting as a divisor, the original number is returned .

C++ example:

```
double lambda = 2.5;
CComplex c = c1/c2 + c2/lambda;
```

6.4 Methods

6.4.1 CComplex Conjugate()

If (x, y) is the complex number this function returns the complex conjugate: $(x,-y)$. In this document the conjugate of c will be denoted by $\bar{c}$

C++ example:

```
double lambda;
CComplex c = c1.Conjugate();
```

6.4.2 bool IsZero() const

If the magnitude of the complex number is less than Matrix::zero_tolerance() true is returned; else false.

C++ example:

```
While (!dx.IsZero())
{
  //do something
  ...
}
```

7 ComplexVector class

This class permits the construction and manipulation of vectors whose elements are complex numbers.

7.1 Constructors

7.1.1 CComplexVector()

Initialises a 3 dimensional vector all of whose elements are zero.

C++ example:

```
CComplexVector v = CComplexVector();
```

7.1.2 CComplexVector(const unsigned int dim)

Initialises a vector of dimension dim all of whose elements are zero.

C++ example:

```
int dim = 3;
CComplexVector v = CComplexVector(dim);
```

7.1.3 CComplexVector(const CComplexVector &v)

Initialises a vector to have the same elements and dimension as v.

C++ example:

```
CComplexVector v1 = CComplexVector(3);
CComplex c = CComplex(3,4);
v1[1]=c;
CComplexVector v2 = CComplexVector(v1);
```

7.2 Properties

7.2.1 CComplex * GetData() const

Returns a pointer to the data of the vector.

C++ example:

```
CComplex *data = v.GetData()
```

7.2.2 Unsigned int GetSize() const

Returns the dimension of the vector - read only.

C++ example:

```
Unsigned int dimension;
CComplexVector v = CComplexVector(3);
dimension = v.GetSize();
```

7.2.3 [i]

Gets and sets the i th element of the vector. Here i is greater than or equal to zero and less than the dimension of the vector. If access outside of this range is requested the first or last elements are read or written respectively.

C++ example:

```
CComplex element1, element2;
CComplexVector v = CComplexVector();
element1 = CComplex(1.5,1.0);
v[0]=element1;
element2=v[1];
```

7.2.4 CVector Abs() const

Returns the real vector, of the same dimension as the parent, whose i th element is the magnitude of the i th element of the parent.

C++ example:

see 7.2.7 .

7.2.5 CComplexVector vector_cos() const

Returns the complex vector whose i th element is the cosine of the corresponding element in the parent vector.

C++ example:

See 7.2.6.

7.2.6 ComplexVector Exp() const

Returns the complex vector whose i ith element is the exponential of the corresponding element in the parent vector.

C++ example:

```
CComplexVector v = CComplexVector();
element1 = CComplex(1.5,1.0);
v[0]=element1;
CComplexVector v2 = v.Exp();
CComplexVector v3 = v.vector_sin();
CComplexvector v4 = v.vetor_cos();
CComplexVector v5 = v.Sqrt();
```

7.2.7 CVector Imaginary() const

Returns the real vector with the same dimension as the parent but each of whose elements is the corresponding imaginary part of the parent.

C++ example:

```
CComplexVector v = CComplexVector();
v[0] = CComplex(1.5,1.0);
CVector vi = v.Imaginary();
CVector vr = v.Real();
CVector va = v.Abs();
```

7.2.8 double modulus() const

Read only property to return the modulus or 2-norm of the vector. If $v = (v_0, v_1, ..., v_{n-1})$ then the modulus is: $\|v\| = \sqrt{v_0 * \bar{v}_0 + ... + v_{n-1} * \bar{v}_{n-1}}$.

C++ example:

```
If (v.modulus() < 1.0e-7)
{
   break
}
```

7.2.9 CVector Real() const

Returns the real vector with the same dimension as the parent but each of whose elements is the corresponding real part of the parent.

C++ example:

see 7.2.7 .

7.2.10 CComplexVector Sin() const

Returns the complex vector whose i th element is the sine of the corresponding element in the parent vector.

C++example:

See 7.2.6.

7.2.11 ComplexVector Sqrt() const

Returns the complex vector whose ith element is the sqrt of the corresponding element in the parent vector.

C++ example:

See 7.2.6.

7.3 Overloaded operators

7.3.1 + operator

If v_1 and v_2 are vectors this permits the expression $v_1 + v_2$ to return a vector. If v_1 and v_2 are not of the same dimension the addition of elements occurs up to the minimum dimension of v_1 and v_2.

C++ example:

```
CComplexVector v = v1 + v2;
```

7.3.2 - operator

If v_1 and v_2 are vectors this permits the expression $v_1 - v_2$ to return a vector. . If v_1 and v_2 are not of the same dimension the subtraction of elements occurs up to the minimum dimension of v_1 and v_2.

C++ example:

```
CComplexVector v = v1 - v2;
```

7.3.3 * operator

If v is a vector and λ a scalar this permits both the expression $\lambda * v$ and $v * \lambda$ to return a vector. λ may be real or complex. The multiplication of the vector by the scalar is component-wise.

If $v1$ and $v2$ are vectors this operator permits the component-wise multiplication of $v1$ and $v2$.

C++ example:

```
double lambda1 = 1.4;
CComplex lambda2 = Complex(0.0,0.6);
CComplexVector v = (lambda1 * v1) * lambda2;
CComplexVector w = v*v;
```

7.3.4 operator /

If v is a vector and λ a scalar this permits the expression $v * / \lambda$ to return a vector. λ may be real or complex. The division of the vector by the scalar is component-wise. If λ is zero the original vector is returned.

If $v1$ and $v2$ are vectors this operator permits the component-wise division of $v1$ and $v2$.

C++ example:

```
double lambda1 = 1.4;
CComplex lambda2 = CComplex(0.0,0.6);
CComplexVector v = (lambda1 / v1) / lambda2;
```

```
CComplexVector w = v/v;  //should be 1 in each component
```

7.4 Methods

7.4.1 CComplexVector Conjugate() const

Returns a vector whose elements are the complex conjugate of those of the parent.

7.4.2 CComplexVector Convolution(CComplexVector v) const

Returns the vector which is the convolution of the parent with v.

If u is the parent vector of length m and v has length n then this method returns the vector w of length $m+n-1$ whose i th element is:

$$w[i] = \sum_j u[j]v[i-j+1]$$

Here the summation is over all values of j which give rise to legal subscripts for $u[j]$ and $v[i-j+1]$.

C++ example:

```
h = f.Convolution(g);
```

7.4.3 CComplex dot(CComplexVector v) const

Returns the dot product with another vector v. If v is not of the same dimension as the parent, componentwise addition occurs up to the minimum of the two dimensions.

C++ example:

```
CComplex c = w.dot(w.Conjugate())
double d = c.Real() - w.modulus()*w.modulus() //should be zero
```

7.4.4 CComplexVector Fft() const

This function uses a Cooley-Tukey method to return the discrete Fourier transform of the parent vector. If x_n is the series representing the parent vector where $0 \le n < N$, N being the dimension of the vector, then the vector returned is X_k $(0 \le k < N)$ where:

$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N}nk}$. Here i represents the square root of minus 1.

C++ example:

```
CMatrix mat = CMatrix.csv_read(str);
CVector r = mat.GetColumnVector(1);
CComplexVector R = CComplexVector(r.GetSize());
R = r.Fft();
```

7.4.5 CComplexVector Ifft() const

This function computes the inverse discrete Fourier transform of the parent vector. Using the notation of 7.4.4,

$$x_n = \frac{1}{N}\sum_{k=0}^{N-1} X_k e^{\frac{2\pi i}{N}nk}$$

C++ example:

```
CMatrix mat = CMtrix.csv_read(str);
CVector t = mat.GetColumnVector(0);
CVector r = mat.GetColumnVector(1);

int N = t.GetSize();
double dt = Math.Abs(t[1] - t[0]) / 1.0E12;
Vector f = new Vector(N);
for (int i = 0; i < N; i++)
{
  f[i] = (1.0 / 1.0E12) * (1.0 / dt) * i / N;
}

CComplexVector R = CComplexVector(N);
R = r.complex().Fft();
CComplexVector ir = R.Ifft();
CVector irr = ir.Real(); //r and irr should be the same

Graph gr = new Graph();
gr.plot(t, r);
gr.plot(t, ir.Real);
gr.add_title("signal against time"); //graphs should be the same
```

7.4.6 void SetSubVector(unsigned int $i1$,unsigned int $i2$,CComplexVector v)

Replaces the elements in the parent between the indices specified with those of vector v. . If $i_1 > i_2$ then i_1 is set equal to i_2 .If $\dim(v) \neq (i_2 - i_1 + 1)$ then the minimum of these two quantities is used to set the parent starting from i_1.

C++ example:

```
v1.SetSubVector(0,2,v2);
```

7.4.7 CComplexVector SubVector(unsigned int $i1$,unsigned int $i2$) const

This returns the vector between indices i_1 and i_2 inclusive of the parent. If i_1 or i_2 are greater or equal to the dimension of the vector then they are set to one less than this dimension. If $i_1 > i_2$ then i_1 is set equal to i_2 .

C++ example:

```
CComplexVector v1 = v2.SubVector(0,2);
```

7.4.8 CComplexVector unit_vector() const

Returns the unit vector corresponding to the vector. If the parent is the zero vector then the parent is returned.

C++ example:

```
CComplexVector v2 = v.unit_vector();
```

8 ComplexMatrix class

This class permits the construction and manipulation of matrices whose elements are complex numbers.

8.1 Constructors

8.1.1 CComplexMatrix()

The default constructor initialises a 3-by-3 matrix each element of which is zero. Rows are represented by the first index and columns by the second index. Both indices are zero based.

C++ example:

```
CComplexMatrix A = CComplexMatrix();
```

8.1.2 CComplexMatrix(const unsigned int m,const unsigned int n)

This constructor initialises an m by n matrix each element of which is zero.

C++ example:

```
CComplexMatrix A = CComplexMatrix(3,4);
```

8.1.3 CComplexMatrix(const CComplexMatrix & rhs)

The copy constructor is used to

- Initialize an object from another of the same type.
- Copy an object to pass it as an argument to a function.
- Copy an object to return it from a function.

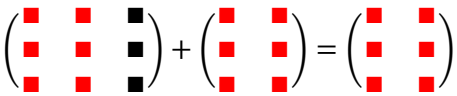
C++ example:

```
CComplexMatrix A = CComplexMatrix(2,3);
CComplexMatrix B = A;
```

8.2 Operators

8.2.1 + operator

If A_1 and A_2 are matrices this operator permits $A_1 + A_2$ to return a matrix. If A_1 and A_2 are not of the same dimension component wise operations are performed up to the minimum of the corresponding dimension of A_1 and A_2. For example the diagram below illustrates how the component-wise addition occurs when the number of columns in the two matrices are not equal:



A_1 and A_2 can be either real or complex matrices interchangeably.

C++ example:

```
CComplexMatrix A = A1 + A1
```

8.2.2 - operator

If A_1 and A_2 are matrices this operator permits $A_1 - A_2$ to return a matrix. If A_1 and A_2 are not of the same dimension component wise operations are performed up to the minimum of the corresponding dimension of A_1 and A_2. A_1 and A_2 can be either real or complex matrices.

C++ example:

```
CComplexMatrix A = A1 – A2;
```

8.2.3 * operator

- If A is a matrix and λ is a scalar this operator permits both $\lambda * A$ and $A * \lambda$ to return a matrix. The multiplication of the matrix by the scalar is component-wise. λ can be real or complex.
- If A_1 and A_2 are two matrices of compatible dimensions for matrix multiplication this operator permits $A_1 * A_2$ to return a matrix. If the dimensions are not compatible then the common dimension is taken to be the minimum of the number of columns of A_1 and the number of rows of A_2 The multiplication then proceeds using this common dimension. A_1 and A_2 can be real or complex matrices.
- If A is a matrix and v a vector this operator permits $A * v$ to return a vector. If A and v are not of compatible dimensions then the common dimension is taken to be the minimum of the number of columns of A and the dimension of v. The multiplication then proceeds using this common dimension. v can be a real or complex vector.

C++ example:

```
CComplexMatrix A = CComplexMatrix::Identity();
CComplexMatrix B = (lambda1 * A) * lambda2;
CComplexMatrix C = A*B;
CVector v = CVector(3);
v[0]=1.0; v[1] = 2.0; v[1] = 3.0;
CComplexVector w = A*v;
```

8.2.4 / operator

If A is a matrix and λ a scalar this operations permits the return of the matrix A/λ. The division is component wise. λ may be real or complex. If λ is zero then A is returned.

C++ example:

```
CComplex lambda = CComplex(1.0,2.0);
CComplexMatrix A = B / lambda;
```

8.2.5 = operator

This permits one matrix to be set equal to another. The elements of the matrix are copied in a component-wise manner. If the dimension of source matrix does not equal the dimension of that of the target. De/reallocation of memory occurs.

C++ example:

```
CComplexMatrix m1 = CComplexMatrix(4,3);
```

```
m1[0,0] = CComplex(1.0,1.0);
CComplexMatrix m2 = CComplexMatrix(4,3);
m2 = m1;
```

8.3 Static Member Functions

8.3.1 static Calculate_Givens_parameters(CComplex a,CComplex b, CComplex & c, CComplex & s)

Given scalars a and b this function computes c and s (where c is real, and $c^2 + |s|^2 = 1$) such that:

$\begin{pmatrix} c & \bar{s} \\ -s & c \end{pmatrix}^T \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} r \\ 0 \end{pmatrix}$, for some scalar r.

Friendly name export prototype, for explicit linking: `GetComplexGivensParameters(CComplex,CComplex,CComplex &,CComplex &).`

8.3.2 static CComplexMatrix Eye(unsigned int $nrows$,unsigned int $ncols$)

8.3.3 static void Eye2(unsigned int $nrows$,unsigned int $ncols$,CComplexMatrix & $result$)

Static functions which returns a matrix such that:

$Eye(i, j) = 0$, for $i \neq j$

$Eye(i, i) = 1$

Friendly name export prototype, for explicit linking: `GetComplexEye(unsigned int,unsigned int, CComplexMatrix &).`

C++ example:

```
CComplexMatrix A = CComplexMatrix::Eye(4,3);
```

8.3.4 static CComplexMatrix Identity(unsigned int m)

8.3.5 static void Identity2(unsigned int m,CComplexMatrix & $result$)

Constructs the identity matrix as a complex matrix.

Friendly name export prototype, for explicit linking: `GetComplexIdentity(unsigned int, CComplexMatrix &).`

C++ example:

```
CComplexMatrix A = CComplexMatrix::Identity(4);
```

8.4 Functions

8.4.1 void Bidiagonalization()

8.4.2 void Bidiagonalization(CComplexMatrix & U ,CcomplexMatrix & V)

If A is the m -by- n matrix, $(m \geq n)$, this method overwrites A with $B = U^H AV$, where B is upper bi-diagonal, U is an m -by - m unitary matrix, and V is an n - by - n unitary matrix. A can be recovered by calculating UBV^H . The zeroing of rows and columns is performed by a succession of Givens transformations. The first function performs the bi-diagonalisation without accumulating U and V .

C++ example:

```
CComplexMatrix A = CComplexMatrix(4,3);
CComplexMatrix U = CComplexMatrix::Identity(4);
CComplexMatrix V = CComplexMatrix::Identity(3);
A.Bidiagonalization(U,V);
```

8.4.3 CComplexMatrix Conjugate() const

Returns the matrix of the same dimension as the parent whose elements are the complex conjugate of the parents elements.

C++ example:

```
CComplexMatrix A = B.Conjugate();
```

8.4.4 CComplexMatrix ConjugateTranspose() const

Returns the conjugate transpose of the parent matrix.

C++ example:

```
CComplexMatrix A = B.ConjugateTranspose();
```

8.4.5 CComplex Determinant(unsigned int & *converged*)

Calculates the determinant of a square matrix using a numerical technique. If the underlying numerical routine has converged *converged* is set to 1. Otherwise it is set to zero.

C++ example:

```
 int conv = 1;
 CComplex det = A.Determinant(ref conv);
```

8.4.6 unsigned int Eigenvalues(CComplexVector E)

If A is the n by n parent matrix this routine calculates the n complex numbers, λ , which together with n vectors, v ,satisfy the equation $Av = \lambda v$ .The eigenvalues λ are returned as a complex vector E . The eigenvalues are sorted in terms of magnitude with the largest first. If the underlying numerical routine used has not converged zero isn returned, otherwise one is returned. The routine used is a QR algorithm employing a double shift. This should converge in most cases. The convergence criteria is determind by *CMatrix::zero_tolerance()* and the maximum number of steps in the iteration by *CMatrix::max_no_loops().* It may be possible to allow the routine to converge by increasing one or both of these parameters.

C++ example:

```
CComplexVector E = CComplexVector(7);
unsigned int Eig_return1 = 1;
Eig_return1 = A.Eigenvalues(E);
```

8.4.7 unsigned int Eigenvectors(CComplexVector $Eval$,CComplexMatrix $Evec$)

If A is the n by n parent matrix this routine calculates the n Eigenvalues, λ , which together with the n Eigenvectors v , satisy the equation $Av = \lambda v$. The Eigenvalues are returned as a complex vector in $Eval$. The corresponding Eigenvectors are returned as columns of the complex matrix $Evec$. The Eigenvectors and their corresponding Eigenvalues are sorted in order of magnitude, with the largest first. If the underlying numerical routine has not converged zero is returned, otherwise one is returned.

The routine used is a QR algorithm employing a double shift. This should converge in most cases. The convergence criteria is determind by *CMatrix::zero_tolerance()* and the maximum number of steps in the iteration by *CMatrix::max_no_loops().* It may be possible to allow the routine to converge by increasing one or both of these parameters.

This routine is not currently optimised for speed.

C++ example:

```
Eig_return = A.Eigenvectors(E,P);
if (Eig_return == 1)
{
    for (i = 0; i < n; i++)
    {
        CComplexVector diff = CComplexVector(n);
        diff = E[i] * P.GetColumnVector(i) - A *
P.GetColumnVector(i);
        if (diff.modulus() > (((double)n)*1.0E-8))
        {
            MessageBox.Show("Complex Eigenvector routine has
failed");
            return;
        }
    }
}
```

8.4.8 CComplexMatrix Eigenvectors2(CComplexMatrix & Q , CcomplexMatrix & T)

A is the n by n parent matrix and Q is a unitary matrix an T an upper triangular matrix such that $Q^H AQ = T$. Then this routine computes the eigenvectors of A as the return matrix. The i th column of the return matrix corresponds to the ith eigenvalue on the diagonal of T .

8.4.9 unsigned int GetColumnSize() const

Returns the number of columns in the matrix.

C++ example:

```
unsigned int ncols = A.GetColumnSize();
```

8.4.10 CComplexVector GetColumnVector(unsigned int j) const

Returns the j th column of the matrix as a vector. If j is less than or equal to the number of columns then the last column is returned.

C++ example:

```
CComplexMatrix B = CComplexMatrix::Eye(3,4);
CComplexVector v = CComplexVector(3);
v = B.GetColumnVector(3);
```

8.4.11 CComplexVector GetDiagonal() const

Returns the vector containing the diagonal entries of the matrix. The matrix does not have to be square.

C++ example:

```
CComplexMatrix B = CComplexMatrix::Eye(3,4);
CComplexVector v = CComplexVector(3);
v = B.GetDiagonal();
```

8.4.12 unsigned int GetRowSize() const

Returns the number of rows in the matrix.

C++ example:

```
unsigned int ncols = A.GetRowSize();
```

8.4.13 CComplexVector GetRowVector(unsigned int i) const

Returns the i th row of the matrix as a vector. If i is less greater or equal to the number of rows then the last row is returned.

C++ example:

```
CComplexMatrix B = CComplexMatrix::Eye(3,4);
CComplexVector v = CComplexVector(4);
v = B.GetRowVector(0);
```

8.4.14 void Givens_pre_multiplication(unsigned int i, unsigned int k, CComplex c,CComplex s)

Let $G(i,k,c,s)$ be the Givens matrix, where c and s are calculated according to 8.3.1.

This is the identity except that
$$G(i,k,c,s)(i,i) = c$$
$$G(i,k,c,s)(k,k) = c$$
$$G(i,k,c,s)(i,k) = \bar{s}$$
$$G(i,k,c,s)(k,i) = -s$$

The Givens pre-multiplication of a matrix A replaces A with $G(i,k,c,s)^T A$.

C++ example:

```
A.Givens_pre_multiplication(i - 1, i, c, s);
```

8.4.15 Givens_post_multiplication(unsigned int i, unsigned int k, CComplex c ,CComplex s)

Let $G(i,k,c,s)$ be the Givens matrix, where c and s are calculated according to 8.3.1.

This is the identity except that
$$G(i,k,c,s)(i,i) = c$$
$$G(i,k,c,s)(k,k) = c$$
$$G(i,k,c,s)(i,k) = \bar{s}$$
$$G(i,k,c,s)(k,i) = -s$$

The Givens pre-multiplication of a matrix A replaces A with $AG(i,k,c,s)$.

C++ example:

```
A.Givens_post_multiplication(i - 1, i, c, s);
```

8.4.16 void HessenbergReduction()

The parent is operated upon to reduce it to an upper Hessenberg form i.e. zeros on all elements below the sub-diagonal. The reduction is done using a series of Givens transformations.

8.4.17 void HessenbergReduction(CComplexMatrix & Q)

This method operates on the parent matrix, A, to reduce it to an upper Hessenberg form H. The reduction is done using a series of Givens transformations. A unitary matrix, Q, is constructed such that $Q^{*T} AQ = H$.

C++ example:

```
CComplexMatrix A = CComplexMatrix(7,7);
...
CComplexMatrix Q = ComplexMatrix::Identity(4);
A.HessenbergReduction(Q);
```

8.4.18 CMatrix Imaginary() const

Returns the matrix of the same dimension as the parent whose elements are the real part of the parents elements.

C++ example:

```
CMatrix A = B.Imaginary();
```

8.4.19 bool is_Hessenberg()

Returns TRUE if the matrix is upper Hessenberg, to within the tolerance given by *CMatrix::zero_tolerance()*. Otherwise the function returns FALSE.

8.4.20 void QR_decomposition(CComplexMatrix & Q,CComplexMatrix & R)

Given the m by n matrix A, this method computes an m by m unitary matrix Q and an m by n upper triangular matrix R such that $A = QR$. The zeroing of columns is performed by a succession of Givens transformations.

C++ example:

```
CcomplexMatrix A = CComplexMatrix(3,3);
A(0,0)=CComplex(1.0,1.0); A(0,1)=CComplex(2.0,0.5);
A(1,0)=CComplex(2.0,7.0);
A(2,0)=CComplex(6.0,0.0);
CComplexMatrix U = CComplexMatrix::Identity(3);
CcomplexMatrix V = CComplexMatrix::Identity(3);
A.QR_decomposition(Q,R);
CComplexMatrix B = A - Q*R; //should be the zero matrix
```

8.4.21 unsigned int rank()

Returns the numerial rank of the matrix. That is the maximum number of linearly independant rows or columns.

C++ example:

```
unsigned int rank = A.rank();
```

8.4.22 CMatrix Real() const

Returns the matrix of the same dimension as the parent whose elements are the real part of the parents elements.

C++ example:

```
CMatrix A = B.Real();
```

8.4.23 void RealBidiagonalization()

8.4.24 void RealBidiagonalization(CComplexMatrix & U , CComplexMatrix & V)

If A is the m -by- n matrix, $(m \geq n)$, this method overwrites A with $B = U^H AV$, where B is upper bi-diagonal with real elements, U is an m -by - m unitary matrix, and V is an n - by - n unitary matrix. A can be recovered by calculating UBV^H . The zeroing of rows and columns is performed by a succession of Givens transformations. The first function performs the bi-diagonalisation without accumulating U and V .

C++ example:

```
CComplexMatrix A = CComplexMatrix(4,3);
...
CComplexMatrix U = CComplexMatrix::Identity(4);
CComplexMatrix V = CComplexMatrix::Identity(3);
A.RealBidiagonalization(U,V);
```

8.4.25 unsigned int SchurDecomposition(CComplexMatrix & Q , CComplexMatrix & T)

This routine calculates a unitary matrix Q and an upper triangular matrix T , such that $Q^H AQ = T$.

If the underlying numerical routine has not converged zero is returned, otherwise one is returned.

The routine used is a QR algorithm employing a double shift. This should converge in most cases. The convergence criteria is determind by *CMatrix::zero_tolerance()* and the maximum number of steps in the iteration by *CMatrix::max_no_loops().* It may be possible to allow the routine to converge by increasing one or both of these parameters.

C++ example:

```
CComplexMatrix Q = CComplexMatrix::Identity(Nrows);
CComplexMatrix T = CComplexMatrix(Nrows,Nrows);

unsigned int conv = S.SchurDecomposition(Q, T);
```

8.4.26 void SetColumnVector(unsigned int j , const CComplexVector & v)

8.4.27 void SetColumnVector(unsigned int j , const CVector & v)

Sets the j th column of the matrix to have the same elements as the vector v .If j is greater than or equal to the number of columns of the matrix, then the last column is set. If the dimension of v does not equal the number of rows of the matrix the component-wise writing occurs up to the minimum of the dimension of v and the number of rows.

C++ example:

```
CComplexMatrix A = CComplexMatrix(3,4);
CVector v = CVector(3);
v[0]= 1.0;
A.SetColumnVector(2,v);
```

8.4.28 void SetDiagonal(const CComplexVector & v)

8.4.29 void SetDiagonal(const CVector & v)

Sets the diagonal entries of a matrix to be those of the elements of v . The matrix does not have to be square. If the dimension of v is does not equal the minimum of the number of rows and columns of the matrix, then component-wise writing occurs up to the minimuim of these two quantities.

C++ example:

```
CComplexMatrix A = CComplexMatrix(3,4);
CVector v = CVector(3);
v[0]= 1.0;
A.SetDiagonal(v);
```

8.4.30 void SetRowVector(unsigned int i, const CComplexVector & v)

8.4.31 void SetRowVector(unsigned int i, const CVector & v)

Sets the i th row of the matrix to have the same elements as the vector v .If i is greater than or equal to the number of rows of the matrix then the last row is set. If the dimension of v does not equal the number of columns of the matrix, then component-wise writing occurs up to the minimum of these two quantities.

C++ example:

```
CComplexMatrix A = CComplexMatrix(3,4);
CVector v = CVector(4);
v[0]= 1.0;
A.SetRowVector(1,v);
```

8.4.32 void SetSubMatrix(unsigned int i_1, unsigned int i_2 ,unsigned int j_1, unsigned int j_2, const CComplexMatrix & A)

8.4.33 void SetSubMatrix(unsigned int i_1 ,unsigned int i_2 ,unsigned int j_1 ,unsigned int j_2 , const CMatrix & A)

Sets the elements spanning rows i_1 to i_2 and columns j_1 to j_2 to the elements of the matrix A. j_2 . If $i_2 < i_1$ or $j_2 < j_1$ or the i 's or j 's do not represent valid indices then they are adjusted appropriately. If the dimensions of A are not compatible with the resulting adjusted indices then component-wise writing occurs for the minimum of the corresponding dimensions.

C++ example:

```
CComplexMatrix A = CComplexMatrix(3,4);
CComplexMatrix A_ = CComplexMatrix(2,2);
A.SetSubMatrix(0,1,0,1,A_);
```

8.4.34 unsigned int singular_values(CVector & s)

Populates a vector containing the p singular values of the matrix ordered in descending order of magnitude. Here, $p = \min(nrows, ncols)$, where $nrows$ and $ncols$ are the number of rows and columns of the matrix respectively. If the underlying iteration converges 1 is returned, otherwise 0 is returned.

C++ example:

```
CComplexMatrix A = CComplexMatrix(4,4);
...
CComplexMatrix S = CComplexMatrix(A);
CComplexMatrix U = CComplexMatrix::Identity(A.GetRowSize());
CComplexMatrix V = CComplexMatrix::Identity(A.GetColumnSize());
unsigned int conv, SVD_return, i;
i = 1;
SVD_return = A.SVD(U,S,V)
conv = A.singular_values(sing);
CComplexVector diff = A*V.GetColumnVector(i) -
sing[i]*U.GetColumnVector(i) //should be the zero vector i = 1, ...,
//3
```

8.4.35 unsigned int SVD(CComplexMatrix & U , CComplexMatrix & S , CComplexMatrix & V)

Given an m by n matrix , these method computes an m by m unitary matrix U , an n by n unitary matrix V and an m by n diagonal matrix S , whose elements are real, such that $A = USV^H$. The symmetry of the decomposition means that both the cases $m \geq n$, and $m < n$ are handled by one routine. If the SVD algorithm converges this function returns 1; otherwise it returns 0.

The diagonal elements of S contain the real singular values in descending order of magnitude. The first min(m , n) columns of U contain the corresponding left singular vectors, u_i , and the first min(m , n) columns of V contain the corresponding right singular vectors, v_i , where $Av_i = \sigma_i u_i \ (1 \leq i \leq \min(m,n))$.

The number of iterations that will be performed in an attempt to converge on the solution is governed by *Matrix::max_no_loops()*. The threshold for a number being zero is given by *Matrix::zero_tolerance()*. If the algorithm does not converge you can always ensure a convergence by increasing *Matrix*::*max_no_loops()* or *Matrix*::*zero_tolerance()* or both.

C++ example:

- Implicit Linking

```
unsigned int converged = 0;
CComplexMatrix S = CComplexMatrix(A);
CComplexMatrix U = CComplexMatrix::Identity(A.GetRowSize());
CComplexMatrix V = CComplexMatrix::Identity(A.GetColumnSize());
converged = A.SVD(U,S,V);
CComplexMatrix B = A - U*S*V.ConjugateTranspose(); //should be the
zero matrix
```

- Explicit Linking

```
HINSTANCE hdll = NULL;
```

```
hdll = LoadLibrary(TEXT("CMatrix.dll"));

typedef CComplexMatrix * (*pvFuncCtv)(unsigned int,unsigned int);
typedef CComplex * (*pvtv)(double, double);
pvFuncCtv CreateMatrixC;
pvtv CreateComplex;
CreateMatrixC = ( pvFuncCtv ) (GetProcAddress(hdll,
"CreateCComplexMatrixClassInstance") );
 CreateComplex = ( pvtv ) (GetProcAddress(hdll,
"CreateCComplexClassInstance") );
CComplexMatrix * matC = (CComplexMatrix *) CreateMatrixC(3,3);
CComplexMatrix * SC = (CComplexMatrix *) CreateMatrixC(3,3);
CComplexMatrix * UC = (CComplexMatrix *) CreateMatrixC(3,3);
CComplexMatrix * VC = (CComplexMatrix *) CreateMatrixC(3,3);
CComplex * cpx = (CComplex *) CreateComplex(1.25,1.26);
CComplex * cpxS2 = (CComplex *) CreateComplex(0,0);
CComplex * cpxS1 = (CComplex *) CreateComplex(0,0);
CComplex * cpxS0 = (CComplex *) CreateComplex(0,0);
(*matC)(2,2) = *cpx;

matC->SVD(*UC,*SC,*VC);

*cpxS2 = (*SC)(2,2);
*cpxS1 = (*SC)(1,1);
*cpxS0 = (*SC)(0,0);

delete cpx, cpxS0, cpxS1, cpxS2, VC, UC, SC, matC;
```

8.4.36 CComplexVector SVD_solve(CComplexVector b, unsigned int & $conv$)

Given the m by n matrix A and the m vector b, this method solves (in a least squares sense) for the n vector x,where $Ax = b$. That is, x minimizes $\|Ax - b\|$ and has the smallest norm of all minimizers. $conv$ indicates if the underlying SVD routine converged ($conv$ set to 1) or not (0).

C++ example:

```
unsigned int converged = 0;

CComplexVector x = A.SVD_solve(b,converged);
```

8.4.37 CComplexMatrix SubMatrix(unsigned int i_1,unsigned int i_2,unsigned int j_1, unsigned int j_2)

Gets the submatrix with rows spanning i_1 to i_2 and columns spanning j_1 to j_2. If $i_2 < i_1$ or $j_2 < j_1$ or the i's or j's do not represent valid indices they are adjusted appropriately.

C++ example:

```
CComplexMatrix V1 = V.SubMatrix(0,n-1,0,p-1);
```

8.4.38 CComplexMatrix Transpose() const

Returns then transpose of the matrix.

C++ example:

```
CComplexMatrix A = B.Transpose();
```

8.4.39 void View()

Displays a window containing a grid containing the values of the matrix at each position.

9 References

[1] Matrix Computations (2nd Edition), Gene H Golub, Charles F. Van Loan, The John Hopkins University Press, 1989.

10 Contact

Paul D. Foy – paulfoy@mathematicalservices.co.uk