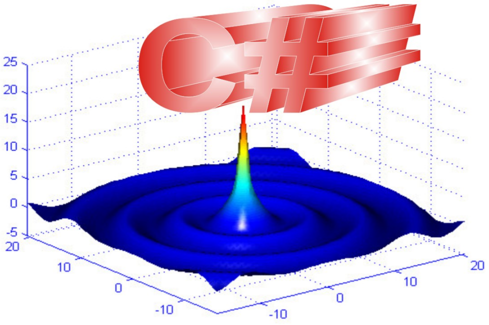
MATLAB® – C#®

for Engineers



This book is a great tutorial for C# programmers
who use MATLAB to develop applications and solutions

MATLAB® C#®
for Engineers (Ebook)

Published by
LePhan Publishing
Mountain View, CA 94043

ISBN 978-0-9826516-1-2

Contents

Preface

MATLAB provides the toolboxes MATLAB Compiler and MATLAB Builder for .NET to handle technical problems between MATLAB and programming languages. The common task is to support MATLAB built-in functions for computer programmers in development. Microsoft Corporation has been developing C# programming language in Microsoft Visual .Net and this C# language is a power language, easy to develop, easy to maintain, and other advanced features, especially in the memory handling. C# is one of the programming languages can use classes created from MATLAB M-files. In addition, the MATLAB Builder for .NET toolbox provides a special feature that allows the user to create a Component Object Model (COM) from MATLAB M-files. The generated COM then can be used in the other programming languages that support COM applications. C# is one of the languages that supports COM, so C# programmers can use MATLAB M-files in the wrapper COM to develop applications as stand-alone applications. This book, ***MATLAB C# for Engineers***, implements the combination of the advances of C# and MATLAB to solve the technical problems. The features of this book are designed to handle the following projects:

- C# functions use MATLAB built-in functions in the class created from MATLAB M-files to solve the mathematical problems
- C# Windows applications use MATLAB built-in functions
- C# functions plot figures from MATLAB Graphics
- C# functions use API functions (calling MATLAB Workspace in C# functions)
- C# functions use MATLAB Curve Fitting Toolbox functions
- C# functions use COMs generated from MATLAB M-files

This book contains all C# programming codes in all chapters that quickly help users solve their problems. This book tries to support C# programmers, especially college students and engineers, who use C# and MATLAB to develop applications and solutions for their projects and designs.

Jack Phan
February 2010

Part I:

Creating and Using MATLAB Functions to Solve Mathematical Problems In C#

Chapter 1

Introduction

1.1 Introduction

MATLAB is mathematical software that includes many toolboxes. MATLAB Compiler and MATLAB Builder for .NET are the most important toolboxes that supports C# computer programmers to use MATLAB functions. We can use MATLAB Builder for .NET to create a class from MATLAB M-files and then the generated functions in this class will then be called by C# functions.

In addition, the MATLAB Builder for .NET toolbox provides a special feature that the user can use to create Component Object Model (COM) from MATLAB M-files. The generated COM then can be used in the other programming languages that support COM applications. C# is one of the languages that supports COM, so C# programmers can use MATLAB M-files in the wrapper COM to develop applications as stand-alone applications. This book, ***MATLAB C# for Engineers***, implements the combination of advances of C# and MATLAB to solve common mathematical problems. The features of this book are designed to handle the following projects:

1. C# functions use MATLAB built-in functions in the class created from MATLAB M-files to solve the mathematical problems
2. C# Windows applications use MATLAB built-in functions
3. C# functions plot figures from MATLAB Graphics
4. C# functions use API functions (calling MATLAB Workspace in C# functions)

5. C# functions use MATLAB Curve Fitting Toolbox functions
6. C# functions use COMs generated from MATLAB M-files

1.2 Computer Software and Book Features

MATLAB Compiler has many versions and there are changes of their features in different versions. The example codes in this book are developed, compiled, and tested in Microsoft Visual .Net 2008, MATLAB 7.9 (R2009b), MATLAB Compiler 4.11, and MATLAB Builder for .NET 3.0.2 with Windows XP, Windows Vista, and Windows 7. These examples are intended to establish common work between C# programming and MATLAB. The example codes are working on scalars, vectors, and matrixes that are inputs/outputs of functions for every application. In addition, the example codes are portable and presented in the step-by-step method. Therefore, the user can easily reuses the codes or writes his/her own codes by following the step-by-step procedure while solving the problems.

The most C# common functions in the examples are `void` functions (return type is `void`) to avoid ambiguity and to emphasize the topic being explained. The book also includes all C# programming example codes and M-files in all chapters.

1.3 Reference Manuals

In working with MATLAB Compiler and MATLAB Builder for .NET, you may need more information to help your tasks. We refer here several manuals from the MATLAB website that you can download for more information.

`http://www.mathworks.com/access/helpdesk/help/pdf_doc/compiler/compiler.pdf`
`http://www.mathworks.com/access/helpdesk/help/pdf_doc/compiler/example_guide.pdf`
`http://www.mathworks.com/access/helpdesk/help/pdf_doc/dotnetbuilder/dotnetbuilder.pdf`
`http://www.mathworks.com/access/helpdesk/help/pdf_doc/curvefit/curvefit.pdf`
`http://www.mathworks.com/access/helpdesk/help/pdf_doc/matlab/refbook.pdf`
`http://www.mathworks.com/access/helpdesk/help/pdf_doc/matlab/refbook2.pdf`
`http://www.mathworks.com/access/helpdesk/help/pdf_doc/matlab/refbook3.pdf`

If you couldn't find these files at the time you are looking for, The MathWorks Inc. may change the URL of these files, but you can find them somewhere in The MathWorks website www.mathworks.com.

Chapter 2

Using Classes Created From MATLAB Compiler In C# Applications

This chapter describes how to generate a class from MATLAB M-files by using MATLAB Compiler and MATLAB Builder for .Net and using it in Microsoft Visual C# .Net 2008 (MSVC# .Net). Based on the M-files, MATLAB Compiler and MATLAB Builder will generate functions in a class. These generated functions will then be called in C# functions to solve particular problems.

Before doing work with MATLAB Compiler, we need to run the command ***mbuild - setup*** to choose a compiler to work with MATLAB Compiler. The following shows process in a computer that chose the compiler of MSVS 2008.

```
>> mbuild -setup
Please choose your compiler for building standalone MATLAB applications:

Would you like mbuild to locate installed compilers [y]/n? y

Select a compiler:
[1] Lcc-win32 C 2.4.1 in C:\PROGRA~1\MATLAB\R2009b\sys\lcc
[2] Microsoft Visual C++ 2008 SP1 in c:\Program Files\Microsoft Visual Studio 9.0
```

```
[0] None

Compiler: 2

Please verify your choices:

Compiler: Microsoft Visual C++ 2008 SP1
Location: c:\Program Files\Microsoft Visual Studio 9.0

Are these correct [y]/n? y

****************************************************************************
  Warning: Applications/components generated using Microsoft Visual Studio
           2008 require that the Microsoft Visual Studio 2008 run-time
           libraries be available on the computer used for deployment.
           To redistribute your applications/components, be sure that the
           deployment machine has these run-time libraries.
****************************************************************************

Trying to update options file:
C:\Users\ComputerNha\AppData\Roaming\MathWorks\MATLAB\R2009b\compopts.bat
From template:
C:\PROGRA~1\MATLAB\R2009b\bin\win32\mbuildopts\msvc90compp.bat

Done . . .
```

2.1 Generating a Class from a MATLAB M-File

The following are the steps of the procedure to generate a class from a MATLAB M-file, `myplus.m`.

1. Write an M-file `myplus.m` as follows:

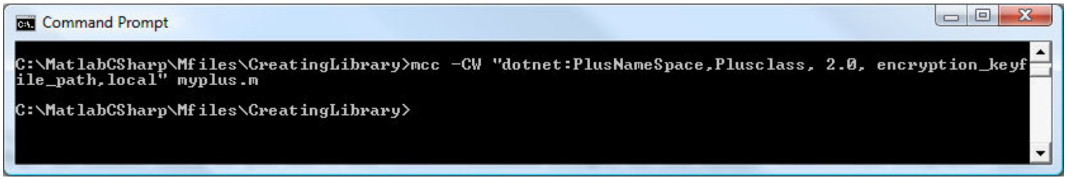
   ```
   function z = myplus(x, y)
   z = x + y ;
   ```

2. Open the command window, **Command Prompt**. Tip: in Windows Vista if you don't see **Command Prompt** in the window **All Programs** then you can find it in the directory, `<SystemRoot>\system32\cmd.exe`. For example, in my computer this file is `C:\system32\cmd.exe`.

3. In **Command Prompt** go the the directory that contains the above M-file `myplus.m`.

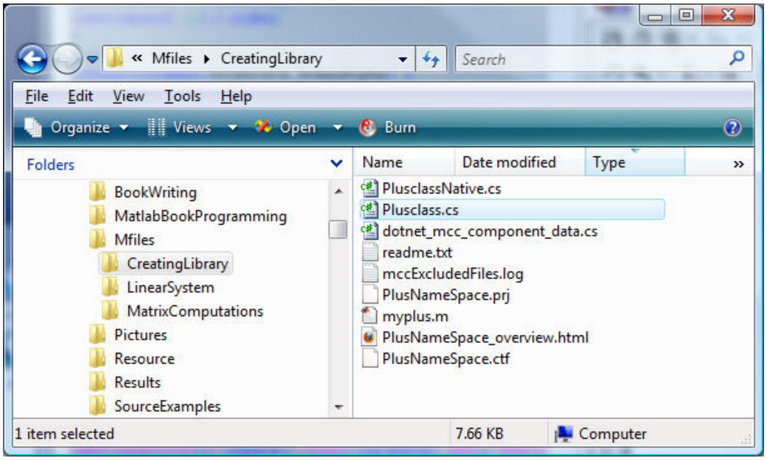
4. Run the following commands (see Fig.2.1) to create a name space ***PlusNameSpace*** with a class ***Plusclass***. With this command, MATLAB will create eight files (see Fig.2.2) (for using Version 2.0 of the framework .NET) and we'll use three files in a C# project described in the following. To have more information of this command, see the command ***mcc*** in the MATLAB Compiler User's Guide documentation [2].

```
mcc -CW "dotnet:PlusNameSpace,Plusclass, 2.0, encryption_keyfile_path,local" myplus.m
```



```
C:\MatlabCSharp\Mfiles\CreatingLibrary>mcc -CW "dotnet:PlusNameSpace,Plusclass, 2.0, encryption_keyf
ile_path,local" myplus.m

C:\MatlabCSharp\Mfiles\CreatingLibrary>
```

Figure 2.1: Command of creating a math class



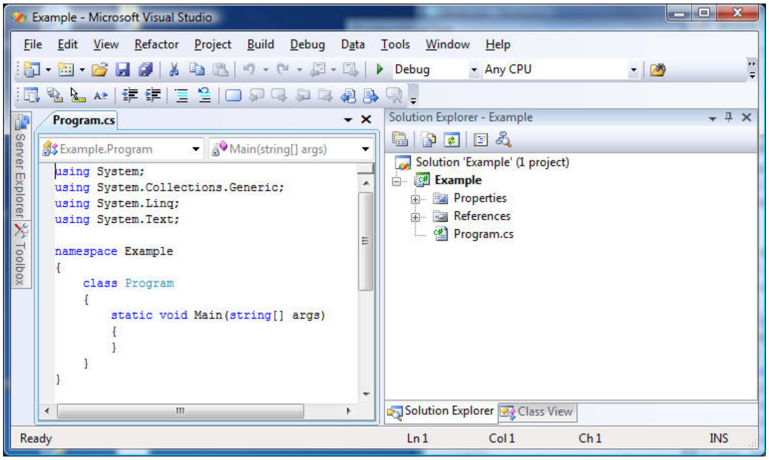
PlusclassNative.cs
Plusclass.cs
dotnet_mcc_component_data.cs
readme.txt
mccExcludedFiles.log
PlusNameSpace.prj
myplus.m
PlusNameSpace_overview.html
PlusNameSpace.ctf

1 item selected 7.66 KB Computer

Figure 2.2: Eight files created from above command

5. Create a normal C# project in Console Application of Microsoft Visual Studio 2008 (Fig. 2.3).

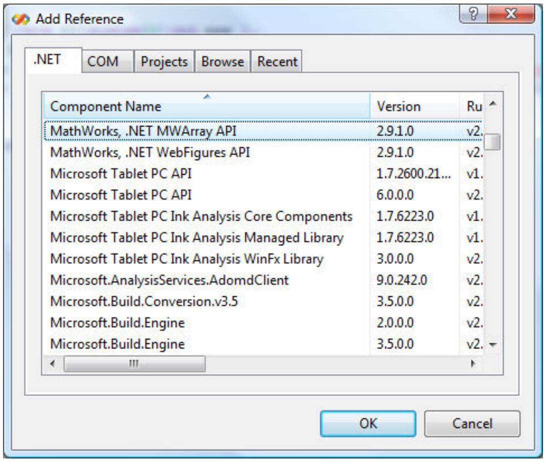
6. Click **Build** then click **Rebuild** to make sure the program works.



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Example
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

Figure 2.3: A normal project C#

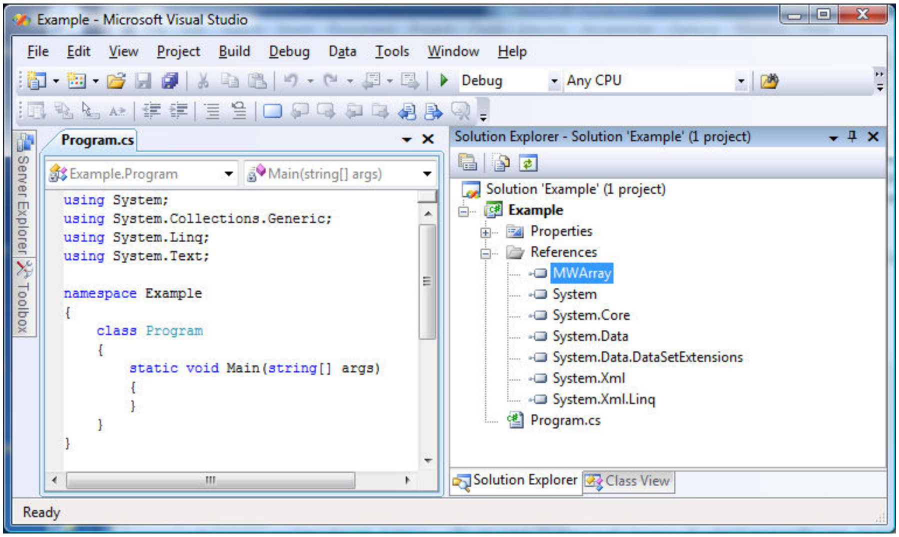
7. In the project menu, click **Project**, **Add Reference**, the project will pop up a dialog as in Fig. 17.3.



Component Name	Version	Ru
MathWorks, .NET MWArray API	2.9.1.0	v2.
MathWorks, .NET WebFigures API	2.9.1.0	v2.
Microsoft Tablet PC API	1.7.2600.21...	v1.
Microsoft Tablet PC API	6.0.0.0	v2.
Microsoft Tablet PC Ink Analysis Core Components	1.7.6223.0	v1.
Microsoft Tablet PC Ink Analysis Managed Library	1.7.6223.0	v1.
Microsoft Tablet PC Ink Analysis WinFx Library	3.0.0.0	v2.
Microsoft.AnalysisServices.AdomdClient	9.0.242.0	v2.
Microsoft.Build.Conversion.v3.5	3.5.0.0	v2.
Microsoft.Build.Engine	2.0.0.0	v2.
Microsoft.Build.Engine	3.5.0.0	v2.

Figure 2.4: References from a C# project

8. Click on tab **.Net**, **MathWorks, .NET MWArray API** then the project will add MATLAB an array wrapper class for .NET, ***MWArray*** into the C# project, see Fig.2.5.



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Example
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

Figure 2.5: Wrapper the class MWArray in a C# project

9. Copy the two files `dotnet_mcc_component_data.cs` and `Plusclass.cs` from previous steps and put into the same directory of the `Program.cs` file, see Fig.2.6.

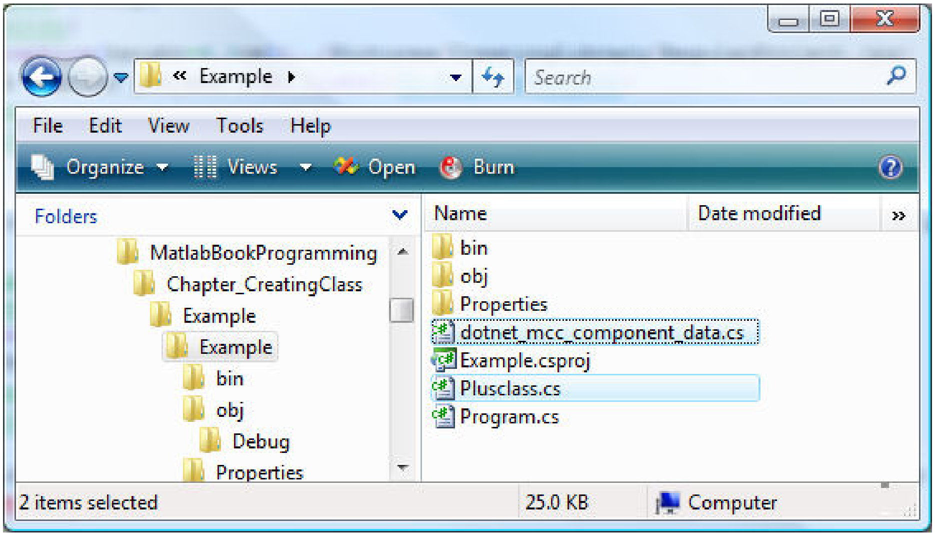


Figure 2.6: Two C# files created from an M-file

10. Copy the file `PlusNameSpace.ctf` and put into the same directory of the project executable file, `Example.exe`, see Fig.2.7.

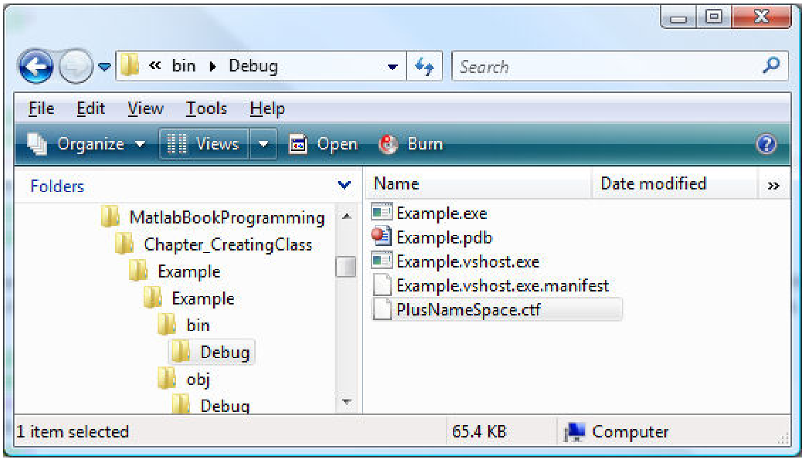
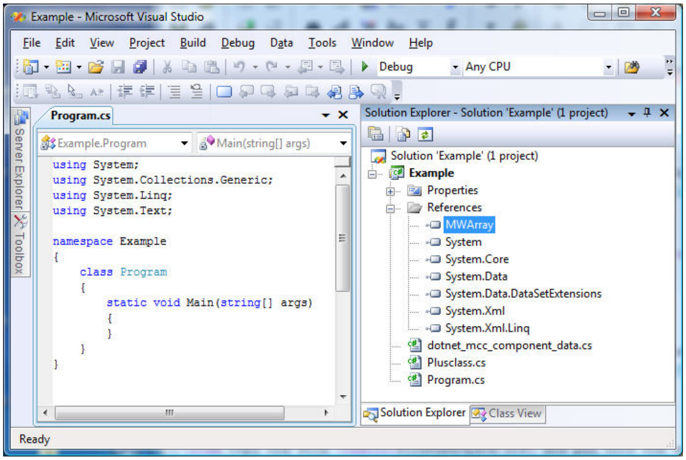


Figure 2.7: A 'ctf' file created from an M-file

11. Add these two files to the C# project by clicking **Project** on the project menu, then click **Add Existing Item**. The project will pop up a dialog, then choose the two files `dotnet_mcc_component_data.cs` and `Plusclass.cs`. The project will add these two files as shown in Fig.2.8.



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Example
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

Figure 2.8: Adding files created from an M-file to C# project

12. At the top of the file `Program.cs`, add the following:

```
using MathWorks.MATLAB.NET.Arrays;
using MathWorks.MATLAB.NET.Utility;
using PlusNameSpace;
```

13. The following code shows how to use the function `myplus.m` M-file in a C# project to solve a simple addition problem.

Listing code

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using MathWorks.MATLAB.NET.Arrays;
using MathWorks.MATLAB.NET.Utility;
using PlusNameSpace;

namespace Example
{
  class Program
  {
    static void Main(string[] args)
    {
      Program objPro = new Program() ;

      Console.WriteLine("Plus calculation example " ) ;
      Console.Write("\n") ;

      double c = objPro.CalculatePlus(2.1, 3.4) ;
      Console.WriteLine(c.ToString() ) ;
    }

    /* ****************** */
    public double CalculatePlus(double a, double b)
    {
      /* declare variables */
      MWNumericArray mw_a = new MWNumericArray(a) ;
      MWNumericArray mw_b = new MWNumericArray(b) ;
      MWNumericArray mw_c ;

      /* call the implemental function */
      Plusclass objMfile = new Plusclass() ;
      mw_c = (MWNumericArray) objMfile.myplus(mw_a, mw_b) ;

      /* convert back to Cs double */
      double result = (double) mw_c ;

      /* free memory */
      mw_a.Dispose() ;
      mw_b.Dispose() ;
      mw_c.Dispose() ;

      return result ;
    }

/* ****************** */
/* ****************** */
/* ****************** */
  }
}
```

end code

In the project, click **Build**, then click **Build Solution** and we should have no error. Click **Debug**, then click **Start Without Debugging** we then have a result.

To add more M-files in generating a class, we just regularly add as we did for the file `myplus.m` above. In the following chapters, we have commands that create a class from multiple MATLAB M-files. There is an alternative way to create a class by using command lines, refer to [3].

Chapter 3

Transfer of Values Between C# double and MATLAB Array

In the version of MATLAB 7.9 (2009b) and MATLAB Compiler 4.11, MATLAB has utility classes that allow in using MATLAB functions in a C# application easier. In order to use MATLAB functions in C#, we need to put inputs (scalar, vector, or matrix, etc.) to these functions. This chapter will show a way to transfer data between ***double*** in C# and ***MWNumericArray*** in MATLAB . To begin with, create a simple project as show in Chapter 2 then you will use this project as an example to show how to transfer values between C# ***double*** and MATLAB arrays.

3.1 Transfer of real and complex scalar values between C# double and MATLAB MWNumericArray

The following segment code shows the transfer of scalar values between C# ***double*** and MATLAB MWNumericArray.

Listing code

```
public void TransferScalarValues()
{
    //1a. Transfer real value from double to MWNumericArray
    double db_a = 1.1   ;
    MWNumericArray mw_a = new MWNumericArray(db_a) ;
```

```
Console.WriteLine("1a. double to MWNumericArray: mw_a = {0}\n", mw_a) ;

double db_b = 2.2   ;
MWNumericArray mw_b = new MWNumericArray(db_b) ;

// Convert values from MWNumericArray to double
//1b. Transfer real value from MWNumericArray to double
MWNumericArray mw_c = null ;
Plusclass objMatlab = new Plusclass() ;
mw_c = (MWNumericArray) objMatlab.myplus(mw_a, mw_b) ;

double db_c = mw_c.ToScalarDouble() ;
Console.WriteLine("1b. MWNumericArray to double: db_c = {0}\n", db_c) ;

//1c. Transfer complex value from double to MWNumericArray
double dbReal_a = 1.1 ;
double dbImag_a = 2.2 ;

double dbReal_b = 3.3 ;
double dbImag_b = 4.4 ;

MWNumericArray mwComplex_a = new MWNumericArray(dbReal_a, dbImag_a) ;
MWNumericArray mwComplex_b = new MWNumericArray(dbReal_b, dbImag_b) ;
Console.WriteLine("1c. double to MWNumericArray Complex: mwComplex_a = {0}\n", mwComplex_a) ;

//1d. Transfer complex value from MWNumericArray to double
MWNumericArray mwComplex_c = null ;
mwComplex_c = (MWNumericArray) objMatlab.myplus(mwComplex_a, mwComplex_b) ;

Console.WriteLine(mwComplex_c) ;
Console.WriteLine();

double[] dbRealBuffer_c = new double[1] ;
double[] dbImagBuffer_c = new double[1] ;

dbRealBuffer_c = (double[]) mwComplex_c.ToVector(MWArrayComponent.Real) ;
```

```
    double dbReal_c = dbRealBuffer_c[0] ;
    double dbImag_c = 0 ;
    if (mwComplex_c.IsComplex)
    {
        dbImagBuffer_c = (double[]) mwComplex_c.ToVector(MWArrayComponent.Imaginary) ;
        dbImag_c = dbImagBuffer_c[0] ;
    }

    Console.WriteLine("1d. MWNumericArray Complex to double") ;
    Console.WriteLine("The real value dbReal_c = {0}: ", dbReal_c) ;
    Console.WriteLine("The imag value dbImag_c = {0}: ", dbImag_c) ;
    Console.WriteLine() ;
}
```

end code

3.2 Transfer of real and complex vector values between C# double and MATLAB MWNumericArray

The following segment code shows the transfer of vector values between C# ***double*** and MATLAB MWNumericArray.

Listing code

```
public void TransferVectorValues()
{
    //2a. transfer real value of a vector from double to MWNumericArray
    double[] db_a = { 1.1, 2.2, 3.3 } ;
    double[] db_b = { 4.4, 5.5, 6.6 } ;

    MWNumericArray mw_a = new MWNumericArray(db_a) ;
    MWNumericArray mw_b = new MWNumericArray(db_b) ;
    Console.WriteLine("2a. double to MWNumericArray in a real vector mw_a: \n{0}", mw_a) ;

    //2b. transfer real value of a vector from MWNumericArray to double
    Plusclass objMatlab = new Plusclass() ;
```

```
MWNumericArray mw_c = null ;
mw_c = (MWNumericArray) objMatlab.myplus(mw_a, mw_b) ;
double[] db_c = (double[]) mw_c.ToVector(MWArrayComponent.Real) ;
Console.WriteLine("2b. MWNumericArray to double in a real vector: \n") ;
PrintVector(db_c) ;

//2c. transfer complex value of a vector from double to MWNumericArray
double[] dbReal_a = { 1.1, 2.2, 3.3 } ;
double[] dbImag_a = { 4.4, 5.5, 6.6 } ;

double[] dbReal_b = { 1.1, 2.2, 3.3 } ;
double[] dbImag_b = { 4.4, 5.5, 6.6 } ;

MWNumericArray mwComplex_a = new MWNumericArray(dbReal_a, dbImag_a) ;
MWNumericArray mwComplex_b = new MWNumericArray(dbReal_b, dbImag_b) ;
Console.WriteLine("2c. complex vector double to MWNumericArray \n{0}", mwComplex_a) ;

MWNumericArray mwComplex_c = null ;
mwComplex_c = (MWNumericArray) objMatlab.myplus(mwComplex_a, mwComplex_b) ;

//2d. Transfer complex value of a vector from MWNumericArray to double
double[] dbReal_c = (double[])mwComplex_c.ToVector(MWArrayComponent.Real) ;
double[] dbImag_c = (double[])Array.CreateInstance(typeof(double), dbReal_c.Length) ;

if (mwComplex_c.IsComplex)
{
    dbImag_c = (double[])mwComplex_c.ToVector(MWArrayComponent.Imaginary) ;
}

Console.WriteLine("2d. MWNumericArray to complex vector double") ;
Console.WriteLine("   The real value of the complex vector :") ;
PrintVector(dbReal_c) ;

Console.WriteLine("   The imag value of the complex vector :") ;
PrintVector(dbImag_c) ;
```

```
}
```

end code

3.3 Transfer of matrix values between C# double and MATLAB MWNumericArray

The following segment code shows the transfer of matrix values between C# and MATLAB MWNumericArray.

Listing code

```
public void TransferMatrixValues()
{
    //3a. Transfer real value of a matrix from double to MWNumericArray
    double[,] db_A = {{ 1.1, 2.2, 3.3} , {4.4, 5.5, 6.6} ,  {7.7, 8.8, 9.9} } ;
    double[,] db_B = {{ 101, 102, 103} , {104, 105, 160} ,  {107, 108, 109} } ;

    MWNumericArray mw_A = new MWNumericArray(db_A) ;
    MWNumericArray mw_B = new MWNumericArray(db_B) ;
    Console.WriteLine("3a. double to MWNumericArray in a real matrix mw_A: {0}\n", mw_A) ;

    //3b. Transfer real value of a matrix from MWNumericArray to double
    Plusclass objMatlab = new Plusclass() ;
    MWNumericArray mw_C = null ;
    mw_C = (MWNumericArray) objMatlab.myplus(mw_A, mw_B) ;
    double[,] db_C = (double[,]) mw_C.ToArray(MWArrayComponent.Real) ;

    Console.WriteLine("3b. MWNumericArray to double in a real matrix mw_A: \n") ;
    PrintMatrix(db_C) ;

    //3c. transfer complex value of a matrix from double to MWNumericArray
    double[,] dbReal_A = {{ 1.1, 2.2, 3.3}, {4.4, 5.5, 6.6},  {7.7, 8.8, 9.9} } ;
    double[,] dbImag_A = {{ 11 , 12 , 13 }, {14 , 15 , 16 },  {17 , 18 , 19 } } ;

    double[,] dbReal_B = {{ 10.1, 20.2, 30.3}, {40.4, 50.5, 60.6},  {70.7, 80.8, 90.9} } ;
    double[,] dbImag_B = {{ 101 , 102 , 103 }, {104 , 105 , 106 },  {107 , 108 , 109 } } ;
    MWNumericArray mwComplex_A = new MWNumericArray(dbReal_A, dbImag_A) ;
    MWNumericArray mwComplex_B = new MWNumericArray(dbReal_A, dbImag_B) ;
    Console.WriteLine("3c. double to MWNumericArray
                   in a complex matrix mwComplex_A {0}\n", mwComplex_A) ;

    //3d. Transfer complex value of a matrix from MWNumericArray to double
    MWNumericArray mwComplex_C = null ;
    mwComplex_C = (MWNumericArray) objMatlab.myplus(mwComplex_A, mwComplex_B) ;

    double[,] dbReal_C = (double[,])mwComplex_C.ToArray(MWArrayComponent.Real) ;

    int row = dbReal_C.GetUpperBound(0) - dbReal_C.GetLowerBound(0) + 1;
    int col = dbReal_C.GetUpperBound(1) - dbReal_C.GetLowerBound(1) + 1;

    double[,] dbImag_C = (double[,])Array.CreateInstance(typeof(double), row, col) ;

    if (mwComplex_C.IsComplex)
    {
        dbImag_C = (double[,])mwComplex_C.ToArray(MWArrayComponent.Imaginary) ;
    }

    Console.WriteLine("3d. MWNumericArray to double in a complex matrix") ;
    Console.WriteLine("   The real value of the complex matrix :") ;
    PrintMatrix(dbReal_C) ;

    Console.WriteLine("   The imag value of the complex matrix :") ;
    PrintMatrix(dbImag_C) ;
}
```

end code

The following is the code of the functions *PrintVector(..)* and *PrintMatrix(..)*

Listing code

```
public static void PrintVector(double[] vector)
{
    int i, row;
    row = vector.GetUpperBound(0) - vector.GetLowerBound(0) + 1;

    for (i = 0; i < row; i++)
    {
        Console.Write("{0} \t", vector.GetValue(i).ToString());
        Console.WriteLine();
    }
}

/* ****************************** */
public static void PrintMatrix(double[,] matrix)
{
    int i, j, row, col ;
    row = matrix.GetUpperBound(0) - matrix.GetLowerBound(0) + 1;
    col = matrix.GetUpperBound(1) - matrix.GetLowerBound(1) + 1;

    for (i=0; i<row; i++)
    {
        for (j=0; j<col; j++)
        {
            Console.Write("{0} \t", matrix.GetValue(i,j).ToString() ) ;
        }
        Console.WriteLine() ;
    }

}
```

end code

Chapter 4

Matrix Computations

In this chapter we'll generate a class ***MatrixComputations*** from common M-files working on matrix computation problems. The generated functions of this class will be used in a MSVC# .Net 2008 project to solve matrix computation problems.

We will write the M-files as shown below to generate the class ***MatrixComputations***.

`mydet.m, myinv.m, myminus.m, mymtimes.m, myplus.m, and mytranspose.m`

mydet.m

```
function y = mydet(a)


y = det(a) ;
```

myinv.m

```
function y = myinv(a)


y = inv(a) ;
```

myminus.m

```
function y = myminus(a, b)
y = a - b ;
```

mymtimes.m

```
function y = mymtimes(a, b)


y = a*b ;
```

myplus.m

```
function y = (a, b)


y = a + b ;
```

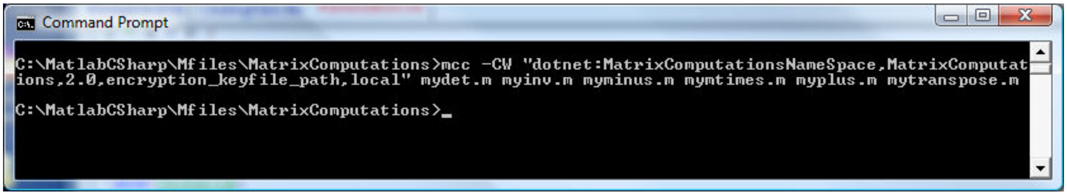
mytranspose.m

```
function y = mytranspose( x )


y = x' ;
```

The following procedure to create the class ***MatrixComputations*** is the same as the procedure in Chapter 2 as follows:

1. Write the following command in **Command Prompt** to generate the name space ***MatrixComputationsNameSpace*** that contains the class ***MatrixComputations*** (see Fig.4.1).

   ```
   mcc -CW "dotnet:MatrixComputationsNameSpace,MatrixComputations,2.0,encryption_keyfile_path,
   local" mydet.m myinv.m myminus.m mymtimes.m myplus.m mytranspose.m
   ```



```
C:\MatlabCSharp\Mfiles\MatrixComputations>mcc -CW "dotnet:MatrixComputationsNameSpace,MatrixComputat
ions,2.0,encryption_keyfile_path,local" mydet.m myinv.m myminus.m mymtimes.m myplus.m mytranspose.m

C:\MatlabCSharp\Mfiles\MatrixComputations>_
```

Figure 4.1: Command of creating the class ***MatrixComputations***

2. Create a regular C# project in Console Application of Microsoft Visual Studio 2008.

3. Click **Build**, then click **Rebuild** to make sure the program works.

4. In the project menu, click **Project**, **Add Reference**, the project will pop up a dialog to choose a reference.

5. Click on tab **.Net**, **MathWorks, .NET MWArray API**. Then the project will add MATLAB array wrapper classes for .NET, MWArray into the C# project.

6. Copy the two files `dotnet_mcc_component_data.cs` and `MatrixComputations.cs` from the previous steps and put them into the same directory of the `Program.cs` file.

7. Copy the file `MatrixComputationsNameSpace.ctf` and put it into the same directory of the project executable file, `Example.exe`.

8. Add these two files to the C# project by clicking **Project** on the project menu. Then click **Add Existing Item**. The project will pop up a dialog to choose the two files `dotnet_mcc_component_data.cs` and `MatrixComputations.cs`. This step will add these two files to the project.

9. At the top of the file `Program.cs`, add the following:

```
using MathWorks.MATLAB.NET.Arrays;
using MathWorks.MATLAB.NET.Utility;
using MatrixComputationsNameSpace;
```

The following sections show how to use the functions in the class ***MatrixComputations*** to solve common computation problems. The full code is at the end of this chapter.

4.1 Matrix Addition

In this section, we will use a MATLAB function in the class ***MatrixComputations*** to find a simple addition matrix.

Problem 1

input . Matrix **A** and **B**

$$\mathbf{A} = \begin{bmatrix} 1.1 & 2.2 & 3.3 \\ 4.4 & 5.5 & 6.6 \\ 7.7 & 8.8 & 9.9 \end{bmatrix} \quad , \quad \mathbf{B} = \begin{bmatrix} 11 & 12 & 13 \\ 14 & 15 & 16 \\ 17 & 18 & 19 \end{bmatrix}$$

output . Finding the addition matrix $\mathbf{C} = \mathbf{A} + \mathbf{B}$

The following subroutine uses a function in the class ***MatrixComputations*** to solve Problem 1 .

Listing code

```
public void AddMatrix()
{
    double [,]A = {{ 1.1, 2.2, 3.3} , {4.4, 5.5, 6.6} ,  {7.7, 8.8, 9.9} } ;
    double [,]B = {{ 11 , 12 , 13 } , {14 , 15 , 16 } ,  {17 , 18 , 19 } } ;

    /* declare variables */
    MWNumericArray mw_A = new MWNumericArray(A) ;
    MWNumericArray mw_B = new MWNumericArray(B) ;
    MWNumericArray mw_C = null ;

    /* call an implemental function */
    MatrixComputations objMatlab = new MatrixComputations() ;
    mw_C = (MWNumericArray)objMatlab.myplus(mw_A, mw_B) ;

    /* convert back to Cs double  */
    double [,] C = (double[,]) mw_C.ToArray(MWArrayComponent.Real) ;

    /* print out */
    Console.WriteLine("result=\n{0}", mw_C);

    /* or */
    PrintMatrix(C) ;

    /* free memory */
    mw_A.Dispose() ;
    mw_B.Dispose() ;
    mw_C.Dispose() ;
}
```

end code

4.2 Matrix Subtraction

In this section, we will use a MATLAB function in the class ***MatrixComputations*** to find a simple subtraction matrix.

Problem 2

input . Matrix **A** and **B**

$$\mathbf{A} = \begin{bmatrix} 1.1 & 2.2 & 3.3 \\ 4.4 & 5.5 & 6.6 \\ 7.7 & 8.8 & 9.9 \end{bmatrix} \quad , \quad \mathbf{B} = \begin{bmatrix} 11 & 12 & 13 \\ 14 & 15 & 16 \\ 17 & 18 & 19 \end{bmatrix}$$

output . Finding the subtraction matrix $\mathbf{C} = \mathbf{A} - \mathbf{B}$

The following subroutine uses a function in the class ***MatrixComputations*** to solve Problem 2 .

Listing code

```
public void SubtractMatrix()
{
    double [,]A = {{ 1.1, 2.2, 3.3} , {4.4, 5.5, 6.6} ,  {7.7, 8.8, 9.9} } ;
    double [,]B = {{ 11 , 12 , 13 } , {14 , 15 , 16 } ,  {17 , 18 , 19 } } ;

    /* declare variables */
    MWNumericArray mw_A = new MWNumericArray(A) ;
    MWNumericArray mw_B = new MWNumericArray(B) ;
    MWNumericArray mw_C = null ;
    /* call an implemental function */
    MatrixComputations objMatlab = new MatrixComputations() ;
    mw_C = (MWNumericArray)objMatlab.myminus(mw_A, mw_B) ;

    /* convert back to Cs double  */
    double [,] C = (double[,]) mw_C.ToArray(MWArrayComponent.Real) ;

    /* print out */
    Console.WriteLine("result=\n{0}", mw_C);

    /* or */
    PrintMatrix(C) ;

    /* free memory */
    mw_A.Dispose() ;
    mw_B.Dispose() ;
    mw_C.Dispose() ;
}
```

end code

4.3 Matrix Multiplication

In this section, we will use a MATLAB function in the class ***MatrixComputations*** to find a multiplication matrix.

Problem 3

input . Matrix $\mathbf{A}$ and $\mathbf{B}$

$$\mathbf{A} = \begin{bmatrix} 1.1 & 2.2 & 3.3 & 4.4 \\ 5.5 & 6.6 & 7.7 & 8.8 \\ 9.9 & 10.10 & 11.11 & 12.12 \end{bmatrix} \quad , \quad \mathbf{B} = \begin{bmatrix} 10 & 11 \\ 12 & 13 \\ 14 & 15 \\ 16 & 17 \end{bmatrix}$$

output . Finding the product matrix $\mathbf{C} = \mathbf{A} * \mathbf{B}$

The following subroutine uses a function in the class ***MatrixComputations*** to solve Problem 3 .

Listing code

```
public void MultipleMatrix()
{
    double [,]A = { { 1.1, 2.2  , 3.3  , 4.4   }    ,
                    { 5.5, 6.6  , 7.7  , 8.8   }    ,
                    { 9.9, 10.10, 11.11, 12.12 }    } ;

    double [,]B = {{ 10, 11}, {12, 13}, {14, 15}, {16, 17} } ;

    /* declare variables */
    MWNumericArray mw_A = new MWNumericArray(A) ;
    MWNumericArray mw_B = new MWNumericArray(B) ;
    MWNumericArray mw_C = null ;

    /* call an implemental function */
    MatrixComputations objMatlab = new MatrixComputations() ;
    mw_C = (MWNumericArray)objMatlab.mymtimes(mw_A, mw_B) ;

    /* convert back to Cs double  */
    double [,] C = (double[,]) mw_C.ToArray(MWArrayComponent.Real) ;

    /* print out */
    Console.WriteLine("result=\n{0}", mw_C);

    /* or */
    PrintMatrix(C) ;

    /* free memory */
    mw_A.Dispose() ;
    mw_B.Dispose() ;
    mw_C.Dispose() ;
}
```

end code

4.4 Matrix Determinant

In this section, we will use a MATLAB function in the class ***MatrixComputations*** to find determinant of a matrix.

Problem 4

input . Matrix **A**

$$\mathbf{A} = \begin{bmatrix} 1.1 & 2.2 & 3.3 \\ 7.7 & 4.4 & 9.9 \\ 4.4 & 5.5 & 8.8 \end{bmatrix}$$

output . Finding the determinant of this matrix **A**

The following subroutine uses a function in the class ***MatrixComputations*** to solve Problem 4.

Listing code

```
public void DeterminantMatrix()
{
    double [,]A = {{ 1.1, 2.2, 3.3},  {7.7, 4.4, 9.9} , {4.4, 5.5, 8.8}  } ;

    /* declare variables */
    MWNumericArray mw_A     = new MWNumericArray(A) ;
    MWNumericArray mw_detA  = null ;

    /* call an implemental function */
    MatrixComputations objMatlab = new MatrixComputations() ;
    mw_detA = (MWNumericArray)objMatlab.mydet(mw_A) ;

    /* convert back to Cs double  */
    double detA = (double) mw_detA ;

    /* print out */
    Console.WriteLine("result = \n{0}", mw_detA) ;
    /* or */
    Console.WriteLine("result = \n{0}", detA) ;

    /* free memory */
    mw_A.Dispose()      ;
    mw_detA.Dispose()   ;
}
```

end code

4.5 Inverse Matrix

In this section, we will use a MATLAB function in the class ***MatrixComputations*** to find inverse of a matrix.

Problem 5

input . Matrix **A**

$$\mathbf{A} = \begin{bmatrix} -1 & 1 & 2 \\ 3 & -1 & 1 \\ -1 & 3 & 4 \end{bmatrix}$$

output . Finding the inverse of this matrix **A**

The following code describes how to use the function myinv(..) in the generated class ***MatrixComputations*** to find a matrix inversion.

The following subroutine uses a function in the class ***MatrixComputations*** to solve Problem 5 .

Listing code

```
public void InverseMatrix()
{
    double [,]A = {{ -1 , 1 , 2} , {3 , -1 , 1} , {-1 , 3 , 4}  } ;

    /* declare variables */
    MWNumericArray mw_A         = new MWNumericArray(A) ;
    MWNumericArray mw_inverseA  = null ;

    /* call an implemental function */
    MatrixComputations objMatlab = new MatrixComputations() ;
    mw_inverseA = (MWNumericArray)objMatlab.myinv(mw_A) ;

    /* convert back to Cs double  */
    double[,] inverseA = (double[,])mw_inverseA.ToArray(MWArrayComponent.Real) ;

    /* print out */
    Console.WriteLine("result = \n{0}", mw_inverseA);

    /* or */
    PrintMatrix(inverseA) ;

    /* free memory */
    mw_A.Dispose()          ;
    mw_inverseA.Dispose()   ;
}
```

end code

4.6 Transpose Matrix

In this section, we will use a MATLAB function in the class ***MatrixComputations*** to find a transpose matrix.

Problem 6

input . Matrix $\mathbf{A}$

$$\mathbf{A} = \begin{bmatrix} -1 & 1 & 2 \\ 3 & -1 & 1 \\ -1 & 3 & 4 \end{bmatrix}$$

output . Finding the transpose of this matrix $\mathbf{A}$

The following subroutine uses a function in the class ***MatrixComputations*** to solve Problem 6.

Listing code

```
public void TransposeMatrix()
{
    double [,] A = {{ -1 , 1 , 2} , {3 , -1 , 1} , {-1 , 3 , 4}  } ;

   /* declare variables */
    MWNumericArray mw_A             = new MWNumericArray(A) ;
    MWNumericArray mw_transposeA    = null ;

    /* call an implemental function */
    MatrixComputations objMatlab = new MatrixComputations() ;
    mw_transposeA = (MWNumericArray)objMatlab.mytranspose(mw_A) ;

    /* convert back to Cs double  */
    double[,] transposeA = (double[,])mw_transposeA.ToArray(MWArrayComponent.Real) ;

    /* print out */
    Console.WriteLine("result = \n{0}", mw_transposeA);

    /* or */
    PrintMatrix(transposeA) ;

    /* free memory */
    mw_A.Dispose()          ;
    mw_transposeA.Dispose()   ;
}
```

end code

The following is the full code for this chapter.

Listing code

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using MathWorks.MATLAB.NET.Arrays;
using MathWorks.MATLAB.NET.Utility;
using MatrixComputationsNameSpace ;

namespace Example
{
  class Program
  {
    static void Main(string[] args)
    {
        Console.WriteLine("Matrix Computations") ;
        Program objProgram = new Program() ;
        Console.WriteLine() ;

        Console.WriteLine("Matrix addition") ;
        objProgram.AddMatrix() ;
        Console.WriteLine() ;

        Console.WriteLine("Matrix subtraction") ;
        objProgram.SubtractMatrix() ;
        Console.WriteLine() ;

        Console.WriteLine("Matrix multiplication") ;
        objProgram.MultipleMatrix();
        Console.WriteLine() ;

        Console.WriteLine("Matrix determinant") ;
        objProgram.DeterminantMatrix() ;
        Console.WriteLine() ;

        Console.WriteLine("Inverse matrix") ;
        objProgram.InverseMatrix() ;
        Console.WriteLine() ;

        Console.WriteLine("Transpose matrix") ;
        objProgram.TransposeMatrix() ;
        Console.WriteLine() ;
    }

    /* **************************************** */
    public void AddMatrix()
    {
      double [,]A = {{ 1.1, 2.2, 3.3} , {4.4, 5.5, 6.6} ,  {7.7, 8.8, 9.9} } ;
      double [,]B = {{ 11 , 12 , 13 } , {14 , 15 , 16 } ,  {17 , 18 , 19 } } ;

      /* declare variables */
      MWNumericArray mw_A = new MWNumericArray(A) ;
      MWNumericArray mw_B = new MWNumericArray(B) ;
      MWNumericArray mw_C = null ;

      /* call an implemental function */
      MatrixComputations objMatlab = new MatrixComputations() ;
      mw_C = (MWNumericArray)objMatlab.myplus(mw_A, mw_B) ;

      /* convert back to Cs double  */
      double [,] C = (double[,]) mw_C.ToArray(MWArrayComponent.Real) ;

      /* print out */
      Console.WriteLine("result=\n{0}", mw_C);

      /* or */
      PrintMatrix(C) ;

      /* free memory */
      mw_A.Dispose() ;
      mw_B.Dispose() ;
      mw_C.Dispose() ;
    }

    /* **************************************** */
    public void SubtractMatrix()
```

```
{
  double [,]A = {{ 1.1, 2.2, 3.3} , {4.4, 5.5, 6.6} ,  {7.7, 8.8, 9.9} } ;
  double [,]B = {{ 11 , 12 , 13 } , {14 , 15 , 16 } ,  {17 , 18 , 19 } } ;

  /* declare variables */
  MWNumericArray mw_A = new MWNumericArray(A) ;
  MWNumericArray mw_B = new MWNumericArray(B) ;
  MWNumericArray mw_C = null ;

  /* call an implemental function */
  MatrixComputations objMatlab = new MatrixComputations() ;
  mw_C = (MWNumericArray)objMatlab.myminus(mw_A, mw_B) ;

  /* convert back to Cs double  */
  double [,] C = (double[,]) mw_C.ToArray(MWArrayComponent.Real) ;

  /* print out */
  Console.WriteLine("result=\n{0}", mw_C);

  /* or */
  PrintMatrix(C) ;

  /* free memory */
  mw_A.Dispose() ;
  mw_B.Dispose() ;
  mw_C.Dispose() ;
}

/* ***************************************** */
public void MultipleMatrix()
{
  double [,]A = { { 1.1, 2.2  , 3.3  , 4.4   }    ,
                  { 5.5, 6.6  , 7.7  , 8.8   }    ,
                  { 9.9, 10.10, 11.11, 12.12 }    } ;

  double [,]B = {{ 10, 11}, {12, 13}, {14, 15}, {16, 17} } ;
  /* declare variables */
  MWNumericArray mw_A = new MWNumericArray(A) ;
  MWNumericArray mw_B = new MWNumericArray(B) ;
  MWNumericArray mw_C = null ;

  /* call an implemental function */
  MatrixComputations objMatlab = new MatrixComputations() ;
  mw_C = (MWNumericArray)objMatlab.mymtimes(mw_A, mw_B) ;

  /* convert back to Cs double  */
  double [,] C = (double[,]) mw_C.ToArray(MWArrayComponent.Real) ;

  /* print out */
  Console.WriteLine("result=\n{0}", mw_C);

  /* or */
  PrintMatrix(C) ;

  /* free memory */
  mw_A.Dispose() ;
  mw_B.Dispose() ;
  mw_C.Dispose() ;
}

/* ***************************************** */
public void DeterminantMatrix()
{
  double [,]A = {{ 1.1, 2.2, 3.3},  {7.7, 4.4, 9.9} , {4.4, 5.5, 8.8}  } ;

  /* declare variables */
  MWNumericArray mw_A     = new MWNumericArray(A) ;
  MWNumericArray mw_detA  = null ;

  /* call an implemental function */
  MatrixComputations objMatlab = new MatrixComputations() ;
  mw_detA = (MWNumericArray)objMatlab.mydet(mw_A) ;

  /* convert back to Cs double  */
  double detA = (double) mw_detA ;

  /* print out */
  Console.WriteLine("result = \n{0}", mw_detA) ;

  /* or */
  Console.WriteLine("result = \n{0}", detA) ;

  /* free memory */
  mw_A.Dispose()      ;
  mw_detA.Dispose()   ;
}

/* ***************************************** */
public void InverseMatrix()
{
  double [,]A = {{ -1 , 1 , 2} , {3 , -1 , 1} , {-1 , 3 , 4}  } ;

  /* declare variables */
  MWNumericArray mw_A         = new MWNumericArray(A) ;
  MWNumericArray mw_inverseA  = null ;

  /* call an implemental function */
  MatrixComputations objMatlab = new MatrixComputations() ;
  mw_inverseA = (MWNumericArray)objMatlab.myinv(mw_A) ;

  /* convert back to Cs double  */
  double[,] inverseA = (double[,])mw_inverseA.ToArray(MWArrayComponent.Real) ;

  /* print out */
  Console.WriteLine("result = \n{0}", mw_inverseA);

  /* or */
  PrintMatrix(inverseA) ;

  /* free memory */
  mw_A.Dispose()          ;
  mw_inverseA.Dispose()   ;
}

/* **************************************** */
public void TransposeMatrix()
{
  double [,] A = {{ -1 , 1 , 2} , {3 , -1 , 1} , {-1 , 3 , 4}  } ;

 /* declare variables */
  MWNumericArray mw_A             = new MWNumericArray(A) ;
  MWNumericArray mw_transposeA    = null ;

  /* call an implemental function */
  MatrixComputations objMatlab = new MatrixComputations() ;
  mw_transposeA = (MWNumericArray)objMatlab.mytranspose(mw_A) ;

  /* convert back to Cs double  */
  double[,] transposeA = (double[,])mw_transposeA.ToArray(MWArrayComponent.Real) ;

  /* print out */
  Console.WriteLine("result = \n{0}", mw_transposeA);

  /* or */
  PrintMatrix(transposeA) ;

  /* free memory */
  mw_A.Dispose()          ;
  mw_transposeA.Dispose()   ;
}

/* **************************************** */
public static void PrintMatrix(double[,] matrix)
```

```
    {
      int i, j, row, col ;
      row = matrix.GetUpperBound(0) - matrix.GetLowerBound(0) + 1;
      col = matrix.GetUpperBound(1) - matrix.GetLowerBound(1) + 1;

      for (i=0; i<row; i++)
      {
        for (j=0; j<col; j++)
        {
          Console.Write("{0} \t", matrix.GetValue(i,j).ToString() ) ;
        }
        Console.WriteLine() ;
      }

    }

    /* ***************************************** */
    /* ***************************************** */
    /* ***************************************** */
    }
}
```

end code

Chapter 5

Linear System Equations

In this chapter we'll generate a class ***LinearSystem*** from common M-files working on problems of linear system equations. The generated functions of this class will be used in a MSVC# .Net 2008 project to solve problems of linear system equations.

We will write the M-files as shown below to generate the class ***LinearSystem***.

`mydiag.m`, `myextractmatrix.m`, `myfull.m`, `mylu.m`, `mymldivide.m`, `mymrdivide.m`, `mysparse.m`, `myspdiags.m`, and `mytranspose.m`

mydiag.m

```
function X = mydiag(v,k)


X = diag(v,k) ;
```

myextractmatrix.m

```
function B = myextractmatrix(A, rowa, rowb, cola, colb)


B = A(rowa:rowb, cola:colb) ;
% extract from row a to row b, and from col a to col b
```

myfull.m

```
function A = myfull(S)
A = full(S) ;
```

mylu.m

```
function [L,U,P] = mylu(A)


[L,U,P] = lu(A) ;
```

mymldivide.m

```
function x = mymldivide(A, b)
%solve equation Ax = b
x = A\b ;
```

mymrdivide.m

```
function x = mymrdivide(A, b)
%solve equation xA = b ==>
x = A/b ;
```

mysparse.m

```
function  S = mysparse(A)


S = sparse(A) ;
```

myspdiags.m

```
function  A = myspdiags(B,d,m,n)
A = spdiags(B,d,m,n)
```

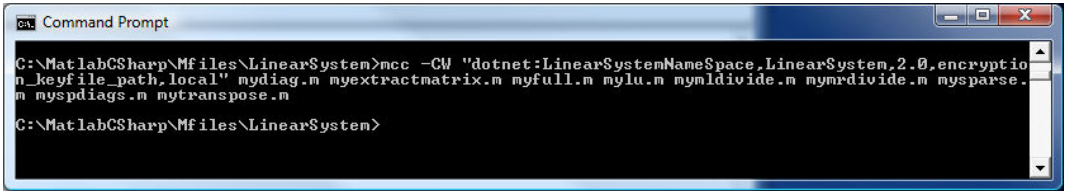
mytranspose.m

```
function y = ( x )


y = x' ;
```

The following procedure to create the class ***LinearSystem*** is the same as the procedure in Chapter 2 as follows:

1. Write the following command in **Command Prompt** to generate the name space ***LinearSystemNameSpace*** that contains the class ***LinearSystem*** (see Fig.5.1).

```
mcc -CW "dotnet:LinearSystemNameSpace,LinearSystem,2.0,encryption_keyfile_path,
local" mydiag.m myextractmatrix.m myfull.m mylu.m mymldivide.m mymrdivide.m
mysparse.m myspdiags.m mytranspose.m
```



```
C:\MatlabCSharp\Mfiles\LinearSystem>mcc -CW "dotnet:LinearSystemNameSpace,LinearSystem,2.0,encryptio
n_keyfile_path,local" mydiag.m myextractmatrix.m myfull.m mylu.m mymldivide.m mymrdivide.m mysparse.
m myspdiags.m mytranspose.m

C:\MatlabCSharp\Mfiles\LinearSystem>
```

Figure 5.1: Command of creating the class ***LinearSystem***

2. Create a regular C# project in Console Application of Microsoft Visual Studio 2008.
3. Click **Build**, then click **Rebuild** to make sure the program works.
4. In the project menu, click **Project**, **Add Reference**, the project will pop up a dialog to choose a reference.
5. Click on tab **.Net**, **MathWorks, .NET MWArray API**. Then project will add MATLAB array wrapper classes for .NET, MWArray into the C# project.
6. Copy the two files `dotnet_mcc_component_data.cs` and `LinearSystem.cs` from the previous steps and put them into the same directory of the `Program.cs` file.
7. Copy the file `LinearSystemNameSpace.ctf` and put it into the same directory of the project executable file, `Example.exe`.
8. Add these two files to the C# project by clicking **Project** on the project menu, then click **Add Existing Item**. The project will pop up a dialog to the two files `dotnet_mcc_component_data.cs` and `LinearSystem.cs`. This step will add these two files to the project.
9. At the top of the file `Program.cs`, add the following:

```
using MathWorks.MATLAB.NET.Arrays;
using MathWorks.MATLAB.NET.Utility;
using LinearSystemNameSpace;
```

The following sections show how to use the functions in the class ***LinearSystem*** to solve the common computation problems. The full code is at the end of this chapter.

5.1 Linear System Equations

In general, the form of linear system equations (size $n \times n$) is:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ a_{31}x_1 + a_{32}x_2 + \cdots + a_{3n}x_n &= b_3 \\ \cdots \qquad\qquad \cdots & \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n \end{aligned} \tag{5.1}$$

In this section, we will use a MATLAB function in the class ***LinearSystem*** to solve linear system problems.

<u>Problem 1</u>

input . Matrix $\mathbf{A}$ and vector $\mathbf{b}$

$$\mathbf{A} = \begin{bmatrix} 1.1 & 5.6 & 3.3 \\ 4.4 & 12.3 & 6.6 \\ 7.7 & 8.8 & 9.9 \end{bmatrix} \quad , \quad \mathbf{b} = \begin{bmatrix} 12.5 \\ 32.2 \\ 45.6 \end{bmatrix}$$

output . Finding the solution $\mathbf{x}$ of linear system equations, $\mathbf{Ax} = \mathbf{b}$

. Finding the lower $\mathbf{L}$ and upper $\mathbf{U}$ of the matrix $\mathbf{A}$ in the decompression method

The following subroutine uses the functions *LinearSystemEquations()* and *LU_ decompression()* in the class ***LinearSystem*** to solve Problem 1.

Listing code

```
public void LinearSystemEquations()
{
    /* Solve general linear system equations Ax = b */
    double[,] A  =  {   {1.1,  5.6, 3.3}  ,
                        {4.4, 12.3, 6.6}  ,
                        {7.7,  8.8, 9.9}  };

    double[] vectorb =  { 12.5, 32.2 , 45.6 } ;

    /* declare variables */
    MWNumericArray mw_A         = new MWNumericArray(A) ;
    MWNumericArray mw_vectorb   = new MWNumericArray(vectorb) ;
    MWNumericArray mw_x         = null ;

    /* call an implemental function */
    LinearSystem objLinear = new LinearSystem() ;
    mw_vectorb  = (MWNumericArray) objLinear.mytranspose(mw_vectorb) ;
    mw_x        = (MWNumericArray) objLinear.mymldivide(mw_A, mw_vectorb) ;

    /* convert back to Cs double  */
    double[] x = (double[])mw_x.ToVector(MWArrayComponent.Real) ;

    /* print out */
    Console.WriteLine("result=\n{0}", mw_x);

    /* or */
    PrintVector(x) ;

    /* free memory */
    mw_A.Dispose() ;
    mw_vectorb.Dispose() ;
    mw_x.Dispose() ;
}

/* *************************** */
```

```
public void LU_decompression()
{
    /* find lower and upper matrixes */
    double[,] A =  {    {1.1,  5.6, 3.3}  ,
                        {4.4, 12.3, 6.6}  ,
                        {7.7,  8.8, 9.9}  };

    /* declare variables */
    MWNumericArray mw_A = new MWNumericArray(A) ;

    MWArray [] mw_ArrayOut = null ;
    MWNumericArray mw_L = null ;
    MWNumericArray mw_U = null ;
    MWNumericArray mw_P = null ;

    /* call an implemental function */
    // value of 3 is three of ouput in mylu.m

    LinearSystem objLinear = new LinearSystem() ;
    mw_ArrayOut = objLinear.mylu(3, mw_A) ;

    mw_L = (MWNumericArray) mw_ArrayOut[0] ;
    mw_U = (MWNumericArray) mw_ArrayOut[1] ;
    mw_P = (MWNumericArray) mw_ArrayOut[2] ;

    /* convert back to Cs double  */
    double [,] L = (double[,]) mw_L.ToArray(MWArrayComponent.Real) ;
    double [,] U = (double[,]) mw_U.ToArray(MWArrayComponent.Real) ;
    double [,] P = (double[,]) mw_P.ToArray(MWArrayComponent.Real) ;

    /* print out */
    Console.WriteLine("The lower matrix") ;
    Console.WriteLine("L = \n{0}", mw_L)  ;

    Console.WriteLine("The upper matrix")   ;
    Console.WriteLine("U = \n{0}", mw_U)    ;
    Console.WriteLine("P = \n{0}", mw_P)    ;

    /* or */
    PrintMatrix(L) ;
    PrintMatrix(U) ;
    PrintMatrix(P) ;

    /* free memory */
    mw_A.Dispose() ;
    MWNumericArray.DisposeArray(mw_ArrayOut) ;
    mw_L.Dispose() ;
    mw_U.Dispose() ;
    mw_P.Dispose() ;
}
```

end code

5.2 Sparse Linear System

The sparse linear system is a common system created to solve a particular technical problem. In this system the main matrix is a sparse matrix (a matrix that has numbers where the nonzero is minor). To obtain an accurate solution and a better computational simulation in the sparse system, MATLAB provided specified functions to handle this task.

In this section, we will use MATLAB functions to solve the following problems.

Problem 2

input . Sparse matrix **A** and vector **b**

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1.1 \\ 0 & 2.2 & 0 & 0 & 0 \\ 3.3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6.6 & 0 \\ 0 & 0 & 5.5 & 0 & 0 \end{bmatrix} \quad , \quad \mathbf{b} = \begin{bmatrix} 11.1 \\ 0 \\ 22.2 \\ 0 \\ 33.3 \end{bmatrix}$$

output . Finding the solution $\mathbf{x}$ of the sparse system equations $\mathbf{Ax} = \mathbf{b}$

The common steps to solve a sparse linear system using functions in the class ***LinearSytem*** are:

1. Establishing the spare matrix by using the function mysparse(..).
2. Solving the sparse system by using the function mymldivide(..).

The following is the code to solve Problem 2 by using the functions, mysparse(..) and mymldivide(..) in the class ***LinearSytem***.

Listing code

```
public void SparseSystem()
{
    /*
    A = 0       0       0       0       1.1
        0       2.2     0       0       0
        3.3     0       0       0       0
        0       0       0       6.6     0
        0       0       5.5     0       0

    b =   11.1  0       22.2    0       33.3
    */

    /* Solve general linear system equations Ax = b */
    double[,] A =    {  {0   ,    0  ,    0  ,   0  ,    1.1 } ,
                        {0   ,    2.2,    0  ,   0  ,    0   } ,
                        {3.3,     0  ,    0  ,   0  ,    0   } ,
                        {0   ,    0  ,    0  ,   6.6,    0   } ,
                        {0   ,    0  ,    5.5,   0  ,    0   } } ;

    double[] vectorb =  { 11.1, 0,  22.2,   0,  33.3 } ;

    /* declare variables */
    MWNumericArray mw_A         = new MWNumericArray(A) ;
    MWNumericArray mw_vectorb   = new MWNumericArray(vectorb) ;
    MWNumericArray mw_x         = null ;
    /* call an implemental function */
    LinearSystem objLinear = new LinearSystem() ;
    mw_vectorb = (MWNumericArray) objLinear.mytranspose(mw_vectorb) ;

    mw_A       = (MWNumericArray) objLinear.mysparse(mw_A) ;
    mw_vectorb = (MWNumericArray) objLinear.mysparse(mw_vectorb) ;

    mw_x       = (MWNumericArray) objLinear.mymldivide(mw_A, mw_vectorb) ;
    mw_x       = (MWNumericArray) objLinear.myfull(mw_x) ;

    /* convert back to Cs double  */
    double [] x = (double[]) mw_x.ToVector(MWArrayComponent.Real) ;

    /* print out */
    Console.WriteLine("Solution x :\n{0}", mw_x);

    /* or */
    PrintVector(x) ;

    /* free memory */
    mw_A.Dispose() ;
    mw_vectorb.Dispose() ;
    mw_x.Dispose() ;
}
```

end code

5.3 Tridiagonal System Equations

This section focuses on finding a solution of the tridiagonal linear system equations $\mathbf{Ax} = \mathbf{d}$:

$$\begin{bmatrix} a_1 & b_1 & 0 & & \cdots & 0 \\ c_2 & a_2 & b_2 & & \cdots & 0 \\ 0 & c_3 & a_3 & b_3 & \cdots & 0 \\ \cdots & 0 & \cdots & & \cdots & \cdots \\ 0 & \cdots & 0 & c_{n-1} & a_{n-1} & b_{n-1} \\ 0 & \cdots & 0 & 0 & c_n & a_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \cdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \cdots \\ d_{n-1} \\ d_n \end{bmatrix} \tag{5.2}$$

Problem 3

input . Matrix **B** includes column vectors **c**, **a**, and **b**

. Vector right-hand side **d**

$$\mathbf{B} = \begin{bmatrix} c_1 & a_1 & b_1 \\ c_2 & a_2 & b_2 \\ c_3 & a_3 & b_3 \\ c_4 & a_4 & b_4 \\ c_5 & a_5 & b_5 \\ c_6 & a_6 & b_6 \end{bmatrix} = \begin{bmatrix} 1.1 & 4.1 & 2.1 \\ 1.2 & 4.2 & 2.2 \\ 1.3 & 4.3 & 2.3 \\ 1.4 & 4.4 & 2.4 \\ 1.5 & 4.5 & 2.5 \\ 1.6 & 4.6 & 2.6 \end{bmatrix} , \quad \mathbf{d} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \end{bmatrix} = \begin{bmatrix} 1.2 \\ 4.5 \\ 5.6 \\ 12.4 \\ 7.8 \\ 6.8 \end{bmatrix} \tag{5.3}$$

output . Finding the solution **x** of tridiagonal system equations in the form of Eq. 5.2

The steps to solve Problem 3 are:

1. Establish a buffer matrix **bufferA** (in Eq. 5.4) from the given matrix **B** (in Eq. 5.3) by using the following function in the class ***LinearSytem***. The function myspdiags(..) creates an m-by-n sparse matrix **bufferA** by taking the columns of B and placing them along the diagonals shown in the following.

$$\textbf{bufferA} = \begin{bmatrix} c_1 & a_1 & b_1 & 0 & & \cdots & 0 & 0 \\ 0 & c_2 & a_2 & b_2 & & \cdots & 0 & 0 \\ 0 & 0 & c_3 & a_3 & b_3 & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & 0 & c_{n-1} & a_{n-1} & b_{n-1} & 0 \\ 0 & 0 & \cdots & 0 & 0 & c_n & a_n & b_n \end{bmatrix} \tag{5.4}$$

2. Obtain the matrix A as in Eq. 5.2 by extracting from the matrix **bufferA**
3. Use the functions in the class ***LinearSytem*** to solve the tridiagonal linear system equations.

The following is the code to solve Problem 3 by using the functions in the library ***LinearSytem***.

Listing code

```
public void TridiagonalSystem()
{
    double [,]B = new double [6,6] ;

    /* columns 1 */
    B[0,0] = 1.1 ;
    B[1,0] = 1.2 ;
    B[2,0] = 1.3 ;
    B[3,0] = 1.4 ;
    B[4,0] = 1.5 ;
    B[5,0] = 1.6 ;

    /* columns 2 */
    B[0,1] = 4.1 ;
    B[1,1] = 4.2 ;
    B[2,1] = 4.3 ;
    B[3,1] = 4.4 ;
    B[4,1] = 4.5 ;
    B[5,1] = 4.6 ;
```

```
/* columns 3 */
B[0,2] = 2.1 ;
B[1,2] = 2.2 ;
B[2,2] = 2.3 ;
B[3,2] = 2.4 ;
B[4,2] = 2.5 ;
B[5,2] = 2.6 ;

double[] db_vectord = new double [6] ;
db_vectord[0] = 1.2  ;
db_vectord[1] = 4.5  ;
db_vectord[2] = 5.6  ;
db_vectord[3] = 12.4 ;
db_vectord[4] = 7.8  ;
db_vectord[5] = 6.8  ;

// The size of the matrix A
int row = 6 ;
int col = 6 ;
int band = 3;  /* tridiagnal, band width */

// Establish the row size=m, and column size=n of the bufferA
int m = row ;
int n = col + (band-1) ;

double[] d = { 0, 1, 2 }; /* for band=3, start from 0 */

int rowB = row  ;
int colB = band ;

/* declare mxArray variables */
MWNumericArray  mw_B        = new MWNumericArray(B) ;
MWNumericArray  mw_bufferA  = null ;
MWNumericArray  mw_A        = null ;

MWNumericArray  mw_d        = new MWNumericArray(d) ;
MWNumericArray  mw_vectord  = new MWNumericArray(db_vectord) ;
MWNumericArray  mw_x        = null ;

/* call an implemental function */
LinearSystem objLinear = new LinearSystem() ;

/* create a sparse matrix mw_bufferA from column-matrix B  */
mw_bufferA = (MWNumericArray) objLinear.myspdiags(mw_B, mw_d, m, n);
mw_bufferA = (MWNumericArray) objLinear.myfull(mw_bufferA) ;

/* plot to see */
Console.WriteLine("The buffer matrix A:") ;
Console.WriteLine(mw_bufferA);

/* extract the need-matrix from the buffter matrix,
  (start at 1 following MATLAB in m.file )
   from row 1 to row 6(=m) and from column 2(=start+1) to column 7(=n-1) */
mw_A = (MWNumericArray) objLinear.myextractmatrix(mw_bufferA, 1, row, 2, 7);

/* plot to see */
Console.WriteLine("The need-matrix A:") ;
Console.WriteLine(mw_A);

/* solve the tridiagnal system equations */
mw_A       = (MWNumericArray) objLinear.mysparse(mw_A)                 ;

mw_vectord = (MWNumericArray) objLinear.mytranspose(mw_vectord)      ;
mw_vectord = (MWNumericArray) objLinear.mysparse(mw_vectord)         ;

mw_x       = (MWNumericArray) objLinear.mymldivide(mw_A, mw_vectord) ;
mw_x       = (MWNumericArray) objLinear.myfull(mw_x)                 ;

/* convert back to Cs double  */
double [] x = (double[]) mw_x.ToVector(MWArrayComponent.Real) ;
```

```
    /*print out */
    Console.WriteLine("Tridiagnal system solution:") ;
    Console.WriteLine(mw_x);

    /* or */
    PrintVector(x) ;

    /* free memory */
    mw_B.Dispose() ;
    mw_bufferA.Dispose() ;
    mw_A.Dispose() ;

    mw_d.Dispose() ;

    mw_vectord.Dispose() ;
    mw_x.Dispose() ;
}
```

end code

5.4 Band Diagonal System Equations

The band diagonal system is a common system in engineering applications. The band diagonal matrix is a matrix with nonzero elements existing only along a few diagonal lines adjacent to the main diagonal (above and below). This section is a study of finding the solution of band diagonal system equations where the $width = 4$. This system is $\mathbf{Ax} = \mathbf{d}$ as follows:

$$\begin{bmatrix} a_1 & b_1 & e_1 & 0 & & \cdots & 0 \\ c_2 & a_2 & b_2 & e_2 & 0 & \cdots & 0 \\ 0 & c_3 & a_3 & b_3 & e_3 & \cdots & 0 \\ \cdots & \cdots & \cdots & & & \cdots & \cdots \\ 0 & \cdots & 0 & c_{n-2} & a_{n-2} & b_{n-2} & e_{n-2} \\ 0 & \cdots & 0 & 0 & c_{n-1} & a_{n-1} & b_{n-1} \\ 0 & \cdots & 0 & 0 & 0 & c_n & a_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \cdots \\ x_{n-2} \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \cdots \\ d_{n-2} \\ d_{n-1} \\ d_n \end{bmatrix} \tag{5.5}$$

The procedure to solve band diagonal system equations is similar to the procedure to solve tridiagonal system equations.

Problem 4

input . Matrix **B** includes columns **c**, **a**, **b**, and **e**.

. Vector **d**

$$\mathbf{B} = \begin{bmatrix} c_1 & a_1 & b_1 & e_1 \\ c_2 & a_2 & b_2 & e_2 \\ c_3 & a_3 & b_3 & e_3 \\ c_4 & a_4 & b_4 & e_4 \\ c_5 & a_5 & b_5 & e_5 \\ c_6 & a_6 & b_6 & e_6 \end{bmatrix} = \begin{bmatrix} 1.1 & 4.1 & 2.1 & 7.1 \\ 1.2 & 4.2 & 2.2 & 7.2 \\ 1.3 & 4.3 & 2.3 & 7.3 \\ 1.4 & 4.4 & 2.4 & 7.4 \\ 1.5 & 4.5 & 2.5 & 7.5 \\ 1.6 & 4.6 & 2.6 & 7.6 \end{bmatrix} \quad , \quad \mathbf{d} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \end{bmatrix} = \begin{bmatrix} 1.2 \\ 4.5 \\ 5.6 \\ 12.4 \\ 7.8 \\ 6.8 \end{bmatrix} \tag{5.6}$$

output .Finding the solution **x** of the band system in the form of Eq. 5.5.

The steps to solve this problem are similar to the steps in the tridiagonal problem. These steps are:

1. Establish a buffer matrix **bufferA** (in Eq. 5.7) from given matrix **B** (in Eq. 5.6) by using a function in the class ***LinearSytem***.

$$\mathbf{bufferA} = \begin{bmatrix} c_1 & a_1 & b_1 & e_1 & 0 & & \cdots & 0 & 0 & 0 \\ 0 & c_2 & a_2 & b_2 & e_2 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & c_3 & a_3 & b_3 & e_3 & \cdots & 0 & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & & & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & 0 & c_{n-2} & a_{n-2} & b_{n-2} & e_{n-2} & 0 & 0 \\ 0 & 0 & \cdots & 0 & 0 & c_{n-1} & a_{n-1} & b_{n-1} & e_{n-1} & 0 \\ 0 & 0 & \cdots & 0 & 0 & 0 & c_n & a_n & b_n & e_n \end{bmatrix} \tag{5.7}$$

2. Obtain the matrix A as in Eq. 5.5 by extracting from the matrix **bufferA**.

3. Use the functions in the class to solve the band system diagonal equations.

The following is the code to solve Problem 4 by using the functions in the library ***LinearSytem***.

Listing code

```
public void BandMatrixSystem()
{
    double[,] B = new double[6,4] ;

    /* columns 1 */
    B[0,0] = 1.1 ;
    B[1,0] = 1.2 ;
    B[2,0] = 1.3 ;
    B[3,0] = 1.4 ;
    B[4,0] = 1.5 ;
    B[5,0] = 1.6 ;

    /* columns 2 */
    B[0,1] = 4.1 ;
    B[1,1] = 4.2 ;
    B[2,1] = 4.3 ;
    B[3,1] = 4.4 ;
    B[4,1] = 4.5 ;
    B[5,1] = 4.6 ;

    /* columns 3 */
    B[0,2] = 2.1 ;
    B[1,2] = 2.2 ;
    B[2,2] = 2.3 ;
    B[3,2] = 2.4 ;
    B[4,2] = 2.5 ;
    B[5,2] = 2.6 ;

    /* columns 4 */
    B[0,3] = 7.1 ;
    B[1,3] = 7.2 ;
```

```
B[2,3] = 7.3 ;
B[3,3] = 7.4 ;
B[4,3] = 7.5 ;
B[5,3] = 7.6 ;

double [] db_vectord = new double [6] ;
db_vectord[0] = 1.2  ;
db_vectord[1] = 4.5  ;
db_vectord[2] = 5.6  ;
db_vectord[3] = 12.4 ;
db_vectord[4] = 7.8  ;
db_vectord[5] = 6.8  ;

// The size of the matrix A
int row = 6 ;
int col = 6 ;

int band = 4 ;  /* band width */

// Establish the row size=m, and column size=n
// of the bufferA

int m = row ;
int n = col + (band-1) ;

double[] d = { 0, 1, 2, 3 }; /* for band = 4; start from 0 */

int rowB = row  ;
int colB = band ;

/* declare mxArray variables */
MWNumericArray  mw_B        = new MWNumericArray(B) ;
MWNumericArray  mw_bufferA  = null ;
MWNumericArray  mw_A        = null ;

MWNumericArray  mw_d        = new MWNumericArray(d) ;
MWNumericArray  mw_vectord  = new MWNumericArray(db_vectord) ;
MWNumericArray  mw_x        = null ;

/* call an implemental function */
/* create a sparse matrix, size mxn, from column-matrix B  */

LinearSystem objLinear = new LinearSystem() ;

mw_bufferA = (MWNumericArray) objLinear.myspdiags(mw_B, mw_d, m, n);
mw_bufferA = (MWNumericArray) objLinear.myfull(mw_bufferA) ;

/* plot to see */
Console.WriteLine("The buffer band-matrix A:") ;
Console.WriteLine(mw_bufferA);

/* extract the need-matrix A from the buffter matrix,
    from row 1 to row 6 and from column 2 to column 7 */

/* extract the need-matrix from the buffter matrix,
  (start at 1 following MATLAB in m.file )
   from row 1 to row 6(=m) and from column 2(=start+1) to column 7(=n-1) */
mw_A = (MWNumericArray) objLinear.myextractmatrix(mw_bufferA, 1, row, 2, 7);

/* plot to see */
Console.WriteLine("The band-need-matrix A:" ) ;
Console.WriteLine(mw_A);

/* solve the system equations */
mw_A       = (MWNumericArray) objLinear.mysparse(mw_A)                  ;

mw_vectord = (MWNumericArray) objLinear.mytranspose(mw_vectord)         ;
mw_vectord = (MWNumericArray) objLinear.mysparse(mw_vectord)            ;

mw_x       = (MWNumericArray) objLinear.mymldivide(mw_A, mw_vectord)    ;
mw_x       = (MWNumericArray) objLinear.myfull(mw_x)                    ;
```

```
    /* convert back to Cs double  */
    double[] solution_x = (double[]) mw_x.ToVector(MWArrayComponent.Real) ;

    /*print out */
    /* convert back to Cs double */
    Console.WriteLine("Band matrix system solution:") ;
    Console.WriteLine(mw_x);

    /* or */
    PrintVector(solution_x) ;

    /* free memory */
    mw_B.Dispose() ;
    mw_bufferA.Dispose() ;
    mw_A.Dispose() ;

    mw_d.Dispose() ;

    mw_vectord.Dispose() ;
    mw_x.Dispose() ;
}
```

end code

The following is the full code for this chapter.

Listing code

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using MathWorks.MATLAB.NET.Arrays;
using MathWorks.MATLAB.NET.Utility;
using LinearSystemNameSpace;
```

```
namespace Example
{
  class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine(" Linear System Equations") ;

      Program objProgram = new Program() ;
      objProgram.LinearSystemEquations() ;

      Console.WriteLine("\n LU_decompression") ;
      objProgram.LU_decompression() ;

      Console.WriteLine("\n Spare System") ;
      objProgram.SparseSystem() ;

      Console.WriteLine() ;
      objProgram.TridiagonalSystem() ;

      Console.WriteLine() ;
      objProgram.BandMatrixSystem() ;
    }

    /* **************************** */
    public void LinearSystemEquations()
    {
      /* Solve general linear system equations Ax = b */
      double[,] A =  {    {1.1,  5.6, 3.3}  ,
                          {4.4, 12.3, 6.6}  ,
                          {7.7,  8.8, 9.9}  };

      double[] vectorb =  { 12.5, 32.2 , 45.6 } ;

      /* declare variables */
      MWNumericArray mw_A         = new MWNumericArray(A) ;
      MWNumericArray mw_vectorb   = new MWNumericArray(vectorb) ;
      MWNumericArray mw_x         = null ;

      /* call an implemental function */
      LinearSystem objLinear = new LinearSystem() ;
      mw_vectorb  = (MWNumericArray) objLinear.mytranspose(mw_vectorb) ;
      mw_x        = (MWNumericArray) objLinear.mymldivide(mw_A, mw_vectorb) ;

      /* convert back to Cs double  */
      double[] x = (double[])mw_x.ToVector(MWArrayComponent.Real) ;

      /* print out */
      Console.WriteLine("result=\n{0}", mw_x);

      /* or */
      PrintVector(x) ;

      /* free memory */
      mw_A.Dispose() ;
      mw_vectorb.Dispose() ;
      mw_x.Dispose() ;
    }

    /* **************************** */
    public void LU_decompression()
    {
      /* find lower and upper matrixes */
      double[,] A =  {    {1.1,  5.6, 3.3}  ,
                          {4.4, 12.3, 6.6}  ,
                          {7.7,  8.8, 9.9}  };

      /* declare variables */
      MWNumericArray mw_A = new MWNumericArray(A) ;

      MWArray [] mw_ArrayOut = null ;
      MWNumericArray mw_L = null ;
      MWNumericArray mw_U = null ;
      MWNumericArray mw_P = null ;

      /* call an implemental function */
      // value of 3 is three of ouput in mylu.m

      LinearSystem objLinear = new LinearSystem() ;
      mw_ArrayOut = objLinear.mylu(3, mw_A) ;

      mw_L = (MWNumericArray) mw_ArrayOut[0] ;
      mw_U = (MWNumericArray) mw_ArrayOut[1] ;
      mw_P = (MWNumericArray) mw_ArrayOut[2] ;

      /* convert back to Cs double  */
      double [,] L = (double[,]) mw_L.ToArray(MWArrayComponent.Real) ;
      double [,] U = (double[,]) mw_U.ToArray(MWArrayComponent.Real) ;
      double [,] P = (double[,]) mw_P.ToArray(MWArrayComponent.Real) ;

      /* print out */
      Console.WriteLine("The lower matrix") ;
      Console.WriteLine("L = \n{0}", mw_L)  ;

      Console.WriteLine("The upper matrix")   ;
      Console.WriteLine("U = \n{0}", mw_U)    ;

      Console.WriteLine("P = \n{0}", mw_P)    ;

      /* or */
      PrintMatrix(L) ;
      PrintMatrix(U) ;
      PrintMatrix(P) ;

      /* free memory */
      mw_A.Dispose() ;
      MWNumericArray.DisposeArray(mw_ArrayOut) ;
      mw_L.Dispose() ;
```

```
  mw_U.Dispose() ;
  mw_P.Dispose() ;
}

/* **************************** */
public void SparseSystem()
{
  /*
  A = 0        0        0        0        1.1
      0        2.2      0        0        0
      3.3      0        0        0        0
      0        0        0        6.6      0
      0        0        5.5      0        0

  b =  11.1    0        22.2     0        33.3
  */

  /* Solve general linear system equations Ax = b */
  double[,] A =    {  {0   ,    0  ,    0  ,   0  ,    1.1 } ,
                      {0   ,    2.2,    0  ,   0  ,    0   } ,
                      {3.3,     0  ,    0  ,   0  ,    0   } ,
                      {0   ,    0  ,    0  ,   6.6,    0   } ,
                      {0   ,    0  ,    5.5,   0  ,    0   } } ;

  double[] vectorb =  {11.1,   0  ,    22.2,  0  ,    33.3} ;

  /* declare variables */
  MWNumericArray mw_A         = new MWNumericArray(A) ;
  MWNumericArray mw_vectorb   = new MWNumericArray(vectorb) ;
  MWNumericArray mw_x         = null ;

  /* call an implemental function */
  LinearSystem objLinear = new LinearSystem() ;
  mw_vectorb = (MWNumericArray) objLinear.mytranspose(mw_vectorb) ;

  mw_A       = (MWNumericArray) objLinear.mysparse(mw_A) ;
```

```
    mw_vectorb = (MWNumericArray) objLinear.mysparse(mw_vectorb) ;

    mw_x       = (MWNumericArray) objLinear.mymldivide(mw_A, mw_vectorb) ;
    mw_x       = (MWNumericArray) objLinear.myfull(mw_x) ;

    /* convert back to Cs double  */
    double [] x = (double[]) mw_x.ToVector(MWArrayComponent.Real) ;

    /* print out */
    Console.WriteLine("Solution x :\n{0}", mw_x);

    /* or */
    PrintVector(x) ;

    /* free memory */
    mw_A.Dispose() ;
    mw_vectorb.Dispose() ;
    mw_x.Dispose() ;
  }

  /* **************************** */
  public void TridiagonalSystem()
  {
    double [,]B = new double [6,6] ;

    /* columns 1 */
    B[0,0] = 1.1 ;
    B[1,0] = 1.2 ;
    B[2,0] = 1.3 ;
    B[3,0] = 1.4 ;
    B[4,0] = 1.5 ;
    B[5,0] = 1.6 ;

    /* columns 2 */
    B[0,1] = 4.1 ;
    B[1,1] = 4.2 ;
```

```
B[2,1] = 4.3 ;
B[3,1] = 4.4 ;
B[4,1] = 4.5 ;
B[5,1] = 4.6 ;

/* columns 3 */
B[0,2] = 2.1 ;
B[1,2] = 2.2 ;
B[2,2] = 2.3 ;
B[3,2] = 2.4 ;
B[4,2] = 2.5 ;
B[5,2] = 2.6 ;

double[] db_vectord = new double [6] ;
db_vectord[0] = 1.2  ;
db_vectord[1] = 4.5  ;
db_vectord[2] = 5.6  ;
db_vectord[3] = 12.4 ;
db_vectord[4] = 7.8  ;
db_vectord[5] = 6.8  ;

// The size of the matrix A
int row = 6 ;
int col = 6 ;
int band = 3;  /* tridiagnal, band width */

// Establish the row size=m, and column size=n of the bufferA
int m = row ;
int n = col + (band-1) ;

double[] d = { 0, 1, 2 }; /* for band=3, start from 0 */

int rowB = row  ;
int colB = band ;

/* declare mxArray variables */
```

```
MWNumericArray  mw_B        = new MWNumericArray(B) ;
MWNumericArray  mw_bufferA  = null ;
MWNumericArray  mw_A        = null ;

MWNumericArray  mw_d        = new MWNumericArray(d) ;

MWNumericArray  mw_vectord  = new MWNumericArray(db_vectord) ;
MWNumericArray  mw_x        = null ;

/* call an implemental function */
LinearSystem objLinear = new LinearSystem() ;

/* create a sparse matrix mw_bufferA from column-matrix B  */
mw_bufferA = (MWNumericArray) objLinear.myspdiags(mw_B, mw_d, m, n);
mw_bufferA = (MWNumericArray) objLinear.myfull(mw_bufferA) ;

/* plot to see */
Console.WriteLine("The buffer matrix A:") ;
Console.WriteLine(mw_bufferA);

/* extract the need-matrix from the buffter matrix,
  (start at 1 following MATLAB in m.file )
   from row 1 to row 6(=m) and from column 2(=start+1) to column 7(=n-1) */
mw_A = (MWNumericArray) objLinear.myextractmatrix(mw_bufferA, 1, row, 2, 7);

/* plot to see */
Console.WriteLine("The need-matrix A:") ;
Console.WriteLine(mw_A);

/* solve the tridiagnal system equations */
mw_A       = (MWNumericArray) objLinear.mysparse(mw_A)              ;

mw_vectord = (MWNumericArray) objLinear.mytranspose(mw_vectord)     ;
mw_vectord = (MWNumericArray) objLinear.mysparse(mw_vectord)        ;

mw_x       = (MWNumericArray) objLinear.mymldivide(mw_A, mw_vectord) ;
```

```
  mw_x       = (MWNumericArray) objLinear.myfull(mw_x)                 ;

  /* convert back to Cs double  */
  double [] x = (double[]) mw_x.ToVector(MWArrayComponent.Real) ;

  /*print out */
  Console.WriteLine("Tridiagnal system solution:") ;
  Console.WriteLine(mw_x);

  /* or */
  PrintVector(x) ;

  /* free memory */
  mw_B.Dispose() ;
  mw_bufferA.Dispose() ;
  mw_A.Dispose() ;

  mw_d.Dispose() ;

  mw_vectord.Dispose() ;
  mw_x.Dispose() ;
}

/* *************************** */
public void BandMatrixSystem()
{
  double[,] B = new double[6,4] ;

  /* columns 1 */
  B[0,0] = 1.1 ;
  B[1,0] = 1.2 ;
  B[2,0] = 1.3 ;
  B[3,0] = 1.4 ;
  B[4,0] = 1.5 ;
  B[5,0] = 1.6 ;
```

```
/* columns 2 */
B[0,1] = 4.1 ;
B[1,1] = 4.2 ;
B[2,1] = 4.3 ;
B[3,1] = 4.4 ;
B[4,1] = 4.5 ;
B[5,1] = 4.6 ;

/* columns 3 */
B[0,2] = 2.1 ;
B[1,2] = 2.2 ;
B[2,2] = 2.3 ;
B[3,2] = 2.4 ;
B[4,2] = 2.5 ;
B[5,2] = 2.6 ;

/* columns 4 */
B[0,3] = 7.1 ;
B[1,3] = 7.2 ;
B[2,3] = 7.3 ;
B[3,3] = 7.4 ;
B[4,3] = 7.5 ;
B[5,3] = 7.6 ;

double [] db_vectord = new double [6] ;
db_vectord[0] = 1.2  ;
db_vectord[1] = 4.5  ;
db_vectord[2] = 5.6  ;
db_vectord[3] = 12.4 ;
db_vectord[4] = 7.8  ;
db_vectord[5] = 6.8  ;

// The size of the matrix A
int row = 6 ;
int col = 6 ;
int band = 4 ;  /* band width */

// Establish the row size=m, and column size=n
// of the bufferA

int m = row ;
int n = col + (band-1) ;

double[] d = { 0, 1, 2, 3 }; /* for band = 4; start from 0 */

int rowB = row  ;
int colB = band ;

/* declare mxArray variables */
MWNumericArray  mw_B        = new MWNumericArray(B) ;
MWNumericArray  mw_bufferA    = null ;
MWNumericArray  mw_A        = null ;

MWNumericArray  mw_d        = new MWNumericArray(d) ;

MWNumericArray  mw_vectord    = new MWNumericArray(db_vectord) ;
MWNumericArray  mw_x        = null ;

/* call an implemental function */
/* create a sparse matrix, size mxn, from column-matrix B  */

LinearSystem objLinear = new LinearSystem() ;

mw_bufferA = (MWNumericArray) objLinear.myspdiags(mw_B, mw_d, m, n);
mw_bufferA = (MWNumericArray) objLinear.myfull(mw_bufferA) ;

/* plot to see */
Console.WriteLine("The buffer band-matrix A:") ;
Console.WriteLine(mw_bufferA);

/* extract the need-matrix A from the buffter matrix,
    from row 1 to row 6 and from column 2 to column 7 */

/* extract the need-matrix from the buffter matrix,
  (start at 1 following MATLAB in m.file )
   from row 1 to row 6(=m) and from column 2(=start+1) to column 7(=n-1) */
mw_A = (MWNumericArray) objLinear.myextractmatrix(mw_bufferA, 1, row, 2, 7);

/* plot to see */
Console.WriteLine("The band-need-matrix A:" ) ;
Console.WriteLine(mw_A);

/* solve the system equations */
mw_A       = (MWNumericArray) objLinear.mysparse(mw_A)                  ;

mw_vectord = (MWNumericArray) objLinear.mytranspose(mw_vectord)         ;
mw_vectord = (MWNumericArray) objLinear.mysparse(mw_vectord)            ;

mw_x       = (MWNumericArray) objLinear.mymldivide(mw_A, mw_vectord)    ;
mw_x       = (MWNumericArray) objLinear.myfull(mw_x)                    ;

/* convert back to Cs double  */
double[] solution_x = (double[]) mw_x.ToVector(MWArrayComponent.Real) ;

/*print out */
/* convert back to Cs double */
Console.WriteLine("Band matrix system solution:") ;
Console.WriteLine(mw_x);

/* or */
PrintVector(solution_x) ;

/* free memory */
mw_B.Dispose() ;
mw_bufferA.Dispose() ;
mw_A.Dispose() ;
```

```
  mw_d.Dispose() ;

  mw_vectord.Dispose() ;
  mw_x.Dispose() ;
}

/* ****************************** */
public static void PrintVector(double[] vector)
{
  int i, row;
  row = vector.GetUpperBound(0) - vector.GetLowerBound(0) + 1;

  for (i = 0; i < row; i++)
  {
    Console.Write("{0} \t", vector.GetValue(i).ToString());
    Console.WriteLine();
  }
}

/* ****************************** */
public static void PrintMatrix(double[,] matrix)
{
  int i, j, row, col ;
  row = matrix.GetUpperBound(0) - matrix.GetLowerBound(0) + 1;
  col = matrix.GetUpperBound(1) - matrix.GetLowerBound(1) + 1;

  for (i=0; i<row; i++)
  {
    for (j=0; j<col; j++)
    {
        Console.Write("{0} \t", matrix.GetValue(i,j).ToString() ) ;
    }
    Console.WriteLine() ;
}

}
```

```
    /* ******************************** */
    /* ******************************** */
    /* ******************************** */
  }
}
```

end code

Chapter 6

Ordinary Differential Equations

In this chapter we'll generate a class ***ODE*** from common M-files working on problems of ordinary differential equations (ODE). The generated functions of this class will be used in MSVC# .Net 2008 project to solve common ODE problems.

The MATLAB function of M-files are used to generate the library to solve ODE problems in this chapter is *ode45(..)*. There are other functions, *ode23(..)*, *ode113(..)*, *ode15s(..)*, and *ode23s(..)*, that can be used to solve ODE problems. Therefore, we can choose a function with options that satisfies problem requirements. For more information on these functions, refer to the manual [6].

We will write the M-files as shown below. These functions will be used to generate the class ***ODE*** . The procedure to create the class ***ODE*** is the same as the procedure in Chapter 2.

```
myode45firstorder.m, yourfunc.m
myode45secondorder.m, yoursecondfunc.m
```

myode45firstorder.m

```
function [t, y] = myode45firstorder(strfunc, tspan, y0)

[t, y] = ode45(@yourfunc, tspan, y0, [], strfunc) ;
```

yourfunc.m

```
function dydt = yourfunc(t, y, strfunc)

%trick for a function with/without t, y
strfunction = strcat(strfunc, '+ 0*t + 0*y') ;

F = inline(strfunction) ;
dydt = feval(F, t, y) ;
```

myode45secondorder.m

```
function [t, y] = myode45secondorder(strfunc, tspan, y0)

[t,y] = ode45(@yoursecondfunc, tspan, y0, [], strfunc) ;
```

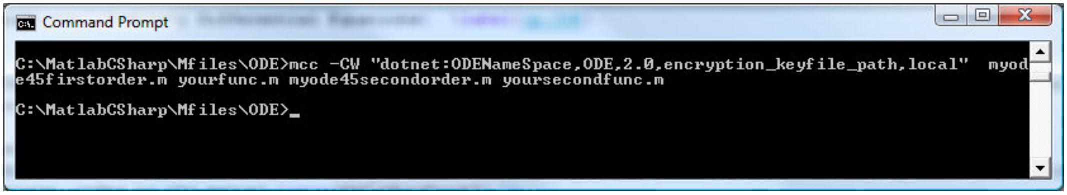
yoursecondfunc.m

```
function dy = yoursecondfunc(t, y, strfunc)

% example:
% y'' - 2y' -6y = cos(3t)
% y'' = cos(3t) + 2y' + 6y
% write an expression string with replace y' by yprime:
%      cos(3*t) + 2*yprime + 6*y



f0 = inline('yy') ;
% it creates a function f(x)=x , as f0(yy) = yy
dy(1,:) = feval( f0, y(2) ) ;

%trick for a function with/without t, yprime, y
strfunction = strcat(strfunc, '+ 0*t + 0*yprime + 0*y') ;

f1 = inline(strfunction) ;
dy(2,:) = feval(  f1, t , y(1), y(2) ) ;
```

The procedure to create the class ***ODE*** is the same as the procedure in Chapter 2 as follows:

1. Write the following command in **Command Prompt** to generate the name space ***ODENameSpace*** that contains the class ***ODE*** (see Fig.6.1).

   ```
   mcc -CW "dotnet:ODENameSpace,ODE,2.0,encryption_keyfile_path,local"
   myode45firstorder.m yourfunc.m myode45secondorder.m yoursecondfunc.m
   ```



```
C:\MatlabCSharp\Mfiles\ODE>mcc -CW "dotnet:ODENameSpace,ODE,2.0,encryption_keyfile_path,local"  myod
e45firstorder.m yourfunc.m myode45secondorder.m yoursecondfunc.m

C:\MatlabCSharp\Mfiles\ODE>_
```

Figure 6.1: Command of creating the class ***ODE***

2. Create a regular C# project in Console Application of Microsoft Visual Studio 2008.
3. Click **Build**, then click **Rebuild** to make sure the program works.
4. In the project menu, click **Project**, **Add Reference**, the project will pop up a dialog to choose a reference.
5. Click on tab **.Net**, **MathWorks, .NET MWArray API** then the project will add MATLAB array wrapper classes for .NET, MWArray into the C# project.
6. Copy the two files `dotnet_mcc_component_data.cs` and `ODE.cs` from the previous steps and put them into the same directory of the `Program.cs` file.
7. Copy the file `ODENameSpace.ctf` and put it into the same directory of the project executable file, `Example.exe`.
8. Add these two files to the C# project by clicking **Project** on the project menu, then click **Add Existing Item**. The project will pop up a dialog to choose the two files `dotnet_mcc_component_data.cs` and `ODE.cs`. This step will add these two files to the project.
9. At the top of the file `Program.cs`, add the following:

```
using MathWorks.MATLAB.NET.Arrays;
using MathWorks.MATLAB.NET.Utility;
using ODENameSpace;
```

The following sections show how to use the functions in the class ***ODE*** to solve the common computation problems. The full code is at the end of this chapter.

6.1 First Order ODE

In this section, we will use MATLAB functions to solve the following problems.

Problem 1 Find the function, $y(t)$, from the ODE function:

$$\frac{dy}{dt} = \cos(t)$$

with initial condition :

$$y_0 = 2.2 \qquad \text{at} \quad t_0 = 0.2$$

Note:

1. When solving first order ODE problems, the MATLAB function *ode45(..)* has an input argument that is an interval `tspan`$=[a, b]$, and the function outputs are two arrays:
   - array **t**[ ] contains the values of time t, $t \in [a, b]$
   - array **y**[ ] contains the values of the function $y(t)$

   The beginning of the interval is given, $a = t_o$. The end of the interval, b, is chosen by users to show the time range in the problem.
2. The time step is set to default if we do not provide a time step.
3. The time step can be set by providing `tspan`$=[t_0, t_2, \ldots, t_n]$ as a vector including the values of time. The output value y will be a column vector. Each row in the solution array y corresponds to the time in the column vector `tspan`.
4. The ODE function is passed as an expression string to the generated function myode45firstorder(..) which is a function-function and has an argument as an expression string. The form of this expression string follows the rule of a MATLAB expression string.

The following is the code to solve Problem 1 by using the function myode45firstorder(..) in the generated ***ODE*** class with the time step set to default.

Listing code

```
public void FirstOrder()
{
    /* Calculating first order ODE */
    //String strfunc = "2+y" ;

    String strfunc = "cos(t)" ;

    double db_y0 = 2.2    ; /* initial condition at t0        */

    double []db_tspan = new double [2]    ;
    db_tspan[0] = 0.2    ; /* begin interval t0 = 0.2        */
    db_tspan[1] = 6.5    ; /* end interval, we choose this    */

    /* declare mxArray variables */
    MWCharArray mw_strfunc  = new MWCharArray(strfunc)      ;
    MWNumericArray mw_tspan = new MWNumericArray(db_tspan)  ;
    MWArray [] mw_ArrayOut  = null ;
    MWNumericArray mw_t     = null ;
    MWNumericArray mw_y     = null ;

    /* call an implemental function */
    ODE objMatlab = new ODE() ;

    /* mx_tspan is a column vector when using in the following function */
    /* call an implemental function */
    mw_ArrayOut = objMatlab.myode45firstorder(2, mw_strfunc, mw_tspan, db_y0) ;
    mw_t        = (MWNumericArray) mw_ArrayOut[0] ;
    mw_y        = (MWNumericArray) mw_ArrayOut[1] ;

    /* convert back to Cs double  */
    double[] db_t = (double[])mw_t.ToVector(MWArrayComponent.Real);
    double[] db_y = (double[])mw_y.ToVector(MWArrayComponent.Real);

    Console.WriteLine("The column of time");
    Console.WriteLine(mw_t) ;

    /* or */
    PrintVector(db_t) ;

    Console.WriteLine("The column of the function values y") ;
    Console.WriteLine(mw_y) ;

    /* or */
    PrintVector(db_y) ;

    /* free memory */
    mw_strfunc.Dispose() ;
    mw_tspan.Dispose() ;
    mw_t.Dispose() ;
    mw_y.Dispose() ;

    MWNumericArray.DisposeArray(mw_ArrayOut) ;
}
```

end code

Problem 2 Find the function, $y(t)$, from the ODE function:

$$\frac{dy}{dt} = 6.4t^2 - 3.8ty$$

with initial condition :

$$y_0 = 1.24 \qquad \text{at} \quad t_0 = 0.15$$

Problem 2 is solved similarly to Problem 1. In the code we just change the expression string:

```
String strfunc = "6.4*t.^2 - 3.8*t*y" ;
```

Remark

To find particular function values we need to set the argument `tspan` as a column vector including the finding time. For example, the following code to find the particular values y at t = 0.15, 0.2, 2.6, and 5.0 in Problem 2.

Listing code

```
public void FirstOrderGetParticularValues()
{
    /* Calculating first order ODE */
    String strfunc = "6.4*t.^2 - 3.8*t*y" ;
    double db_y0 = 1.24  ; /* initial condition at t0            */

    double[] db_tspan = new double [4]  ;
    db_tspan[0] = 0.15  ; /* begin interval t0 = 0.15           */
    db_tspan[1] = 0.2   ; /* choose a particular time t = 0.2   */
    db_tspan[2] = 2.6   ; /* choose a particular time t = 2.6   */
    db_tspan[3] = 5.0   ; /* choose a particular time t = 5.0   */

    MWCharArray mw_strfunc  = new MWCharArray(strfunc)      ;
    MWNumericArray mw_tspan = new MWNumericArray(db_tspan)  ;

    MWArray [] mw_ArrayOut  = null ;
    MWNumericArray mw_t     = null ;
    MWNumericArray mw_y     = null ;

    /* mx_tspan is a column vector when using in the following function */
    /* convert Cs double to mxArray */

    /* call an implemental function */
    ODE objMatlab = new ODE() ;
    mw_ArrayOut = objMatlab.myode45firstorder(2, mw_strfunc, mw_tspan, db_y0) ;
    mw_t        = (MWNumericArray) mw_ArrayOut[0] ;
    mw_y        = (MWNumericArray) mw_ArrayOut[1] ;
    /* convert back to Cs double  */
    double[] db_t = (double[])mw_t.ToVector(MWArrayComponent.Real);
    double[] db_y = (double[])mw_y.ToVector(MWArrayComponent.Real);

    Console.WriteLine("The column of time") ;
    Console.WriteLine(mw_t) ;

    /* or */
    PrintVector(db_t) ;

    Console.WriteLine("The column of the function values y" ) ;
    Console.WriteLine(mw_y) ;

    /* or */
    PrintVector(db_y);

    /* free memory */
    mw_strfunc.Dispose() ;
    mw_tspan.Dispose() ;
    mw_t.Dispose() ;
    mw_y.Dispose() ;

    MWNumericArray.DisposeArray(mw_ArrayOut) ;
}
```

——————————————————————— end code ———————

6.2 Second Order ODE

In this section, we will use MATLAB functions to solve the following problems.

Problem 3 Find the function $y(t)$, and its derivative $y'(t)$ from the ODE function,

$$y'' - 2y' - 6y = \cos(3t)$$

with initial conditions :

$$\text{at} \quad t_0 = 0.12, \quad y_0 = 0.2 \quad \text{and} \quad y'_0 = 1.1$$

6.2.1 Analysis of second order ODE

1. To solve Problem 3 by writing M-files, we can write an M-file mysecondfunc.m as follows:

   ```
   function dy = mysecondfunc(t, y)


   dy = [y(2) ; cos(3*t) + 2*y(2) + 6*y(1)] ;
   ```

   and in MATLAB Command Window write:

   ```
   >> tspan = [1.2 ; 2.5] ;
   >> ybc   = [0.2 ; 1.1] ;
   >> [t,y] = ode45(@mysecondfunc, tspan, ybc)
   ```

2. To explain the code in the M-file mysecondfunc.m, we rewrite and set expressions from the provided equation :

   $$\begin{aligned} y &= y_1 \\ y' &= y_2 \\ y'' &= y'_2 \end{aligned}$$

   Problem 3 then becomes:

   $$\begin{aligned} y &= y_1 \\ y' &= y_2 \\ y'' &= \cos(3t) + 2y' + 6y \\ &= \cos(3t) + 2y_2 + 6y_1 \end{aligned}$$

   This is the second expression in the above M-file mysecondfunc.m.

3. The function `mysecondfunc(..)`, which is passed to the ode function ode45(..), has a return including two arrays:

   - First array is the first derivative of the function y, as `y(2)`

- Second array is the second derivative of the function y, as

  `cos(3*t) + 2*y(2) + 6*y(1) ;`

The M-file yoursecondfunc.m, with which we use to create a class ***ODE*** as shown in above, also has a return including two arrays:

- First array is the first derivative of the function y, as follows:

  ```
  f0 = inline('yy') ;
  dy(1,:) = feval( f0, y(2) ) ;
  ```

- Second array is the second derivative of the function y, as follows:

  `cos(3*t) + 2*y(2) + 6*y(1) ;`

  This second array is represented in the code:

  ```
  f1 = inline(strfunction) ;
  dy(2,:) = feval( f1, t , y(1), y(2) ) ;
  ```

6.2.2 Using a second order ODE function

As explained in above, we have an easy way to use the generated function myode45secondorder(..) in the class ***ODE*** to solve the second order ODE problems by following the steps:

1. Write the ode-function with second derivative in the left-hand-side, for example:

   $y'' = \cos(3t) + 2y' + 6y$

2. Rewrite your ode-function as a MATLAB expression string, for example:

   `y'' = cos(3*t) + 2*y' + 6*y`

3. Replace `y'` by `yprime`, for example:

   `y'' = cos(3*t) + 2*yprime + 6*y`

4. Use the right-hand-side as a string to use in the code, for example:

   `String strfunc = "cos(3*t) + 2*yprime + 6*y" ;`

The following is the code to solve Problem 3 by using the function myode45secondorder(..) in the generated ***ODE*** library .

Listing code

```
public void SecondOrder()
{
    /* Calculating second order ODE */
    String strfunc = "cos(3*t) + 2*yprime + 6*y" ;

    double[] db_ybc = new double [2]    ; /* y boundary conditions            */
    db_ybc[0]  = 0.2    ; /* initial condition of y  at t0    */
    db_ybc[1]  = 1.1    ; /* initial condition of y' at t0    */

    double[] db_tspan = new double [2]    ;
    db_tspan[0] = 1.2    ; /* begin interval t0 = 1.2        */
    db_tspan[1] = 2.5    ; /* end interval, we choose this    */

    /* declare mxArray variables */
    MWCharArray mw_strfunc  = new MWCharArray(strfunc)      ;
    MWNumericArray mw_tspan = new MWNumericArray(db_tspan)  ;
    MWNumericArray mw_ybc   = new MWNumericArray(db_ybc)    ;

    MWArray [] mw_ArrayOut  = null ;
    MWNumericArray mw_t     = null ;
    MWNumericArray mw_y     = null ;

    /* call an implemental function */
    ODE objMatlab = new ODE() ;
    mw_ArrayOut = objMatlab.myode45secondorder(2, mw_strfunc, mw_tspan, mw_ybc) ;
    mw_t        = (MWNumericArray) mw_ArrayOut[0] ;
    mw_y        = (MWNumericArray) mw_ArrayOut[1] ;

    /* convert back to Cs double  */
    double[] db_t  = (double[])  mw_t.ToVector(MWArrayComponent.Real);
    double[,] db_y = (double[,]) mw_y.ToArray(MWArrayComponent.Real) ;

    Console.WriteLine("The column of time") ;
    Console.WriteLine(mw_t) ;

    /* or */
    // PrintVector(db_t) ;

    Console.WriteLine("The column of the function values y :") ;
    /* first column of the matrix y */
    int i ;
    int aLength = db_t.Length ;
    for (i=0; i<aLength; i++)
    {
        Console.Write("{0} ", db_y.GetValue(i,0).ToString() ) ;
        Console.WriteLine() ;
    }

    Console.WriteLine("The column of the first derivative y' :") ;
    /* second column of the matrix y */
    for (i=0; i<aLength; i++)
    {
        Console.Write("{0} ", db_y.GetValue(i,1).ToString() ) ;
        Console.WriteLine() ;
    }

    /* free memory */
    mw_strfunc.Dispose() ;
    mw_tspan.Dispose() ;
    mw_ybc.Dispose() ;
    mw_t.Dispose() ;
    mw_y.Dispose() ;

    MWNumericArray.DisposeArray(mw_ArrayOut) ;
}
```

end code

Note:

1. In solving second order ODE problem, the output matrix **y** includes two columns. The first column is values of the function $y(t)$ and the second column is values of the first derivative $y'(t)$.

2. In this chapter we describe the methods to solve ODE problems by passing ode-functions to the C# code. These methods are useful when your function are changing in the run-time or your function is provided in an application. If your ode-function is known in the design time, you can call directly. For example, the M-file myotherode.m below will directly call the M-file mysecondfunc.m:

mysecondfunc.m

```
function dy = mysecondfunc(t, y)
dy = [y(2) ; cos(3*t) + 2*y(2) + 6*y(1)] ;
```

myotherode.m

```
function [t, y] = myotherode(tspan, y0)

[t,y] = ode45(@mysecondfunc, tspan, y0) ;
```

The following is full code for this chapter.

Listing code

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using MathWorks.MATLAB.NET.Arrays;
using MathWorks.MATLAB.NET.Utility;
using ODENameSpace ;

namespace Example
{
    class Program
    {
        static void Main(string[] args)
        {
```

```
        Program objProgram = new Program() ;
        Console.WriteLine("ODE problems.") ;

        Console.WriteLine("First order ODE.") ;
        objProgram.FirstOrder() ;

        Console.WriteLine("First order ODE. Get particular values") ;
        objProgram.FirstOrderGetParticularValues() ;

        Console.WriteLine("Second order ODE.") ;
        objProgram.SecondOrder()  ;
    }

    /* **************************** */
    public void FirstOrder()
    {
        /* Calculating first order ODE */
        //String strfunc = "2+y" ;

        String strfunc = "cos(t)" ;

        double db_y0 = 2.2    ; /* initial condition at t0        */

        double []db_tspan = new double [2]    ;
        db_tspan[0] = 0.2    ; /* begin interval t0 = 0.2        */
        db_tspan[1] = 6.5    ; /* end interval, we choose this    */

        /* declare mxArray variables */
        MWCharArray mw_strfunc  = new MWCharArray(strfunc)      ;
        MWNumericArray mw_tspan = new MWNumericArray(db_tspan)  ;
        MWArray [] mw_ArrayOut  = null ;
        MWNumericArray mw_t     = null ;
        MWNumericArray mw_y     = null ;

        /* call an implemental function */
        ODE objMatlab = new ODE() ;
        /* mx_tspan is a column vector when using in the following function */
        /* call an implemental function */
        mw_ArrayOut = objMatlab.myode45firstorder(2, mw_strfunc, mw_tspan, db_y0) ;
        mw_t        = (MWNumericArray) mw_ArrayOut[0] ;
        mw_y        = (MWNumericArray) mw_ArrayOut[1] ;

        /* convert back to Cs double  */
        double[] db_t = (double[])mw_t.ToVector(MWArrayComponent.Real);
        double[] db_y = (double[])mw_y.ToVector(MWArrayComponent.Real);

        Console.WriteLine("The column of time");
        Console.WriteLine(mw_t) ;

        /* or */
        PrintVector(db_t) ;

        Console.WriteLine("The column of the function values y") ;
        Console.WriteLine(mw_y) ;

        /* or */
        PrintVector(db_y) ;

        /* free memory */
        mw_strfunc.Dispose() ;
        mw_tspan.Dispose() ;
        mw_t.Dispose() ;
        mw_y.Dispose() ;

        MWNumericArray.DisposeArray(mw_ArrayOut) ;
    }

    /* *************************** */
    public void FirstOrderGetParticularValues()
    {
        /* Calculating first order ODE */
```

```
String strfunc = "6.4*t.^2 - 3.8*t*y" ;
double db_y0 = 1.24  ; /* initial condition at t0            */

double[] db_tspan = new double [4]  ;
db_tspan[0] = 0.15  ; /* begin interval t0 = 0.15           */
db_tspan[1] = 0.2   ; /* choose a particular time t = 0.2   */
db_tspan[2] = 2.6   ; /* choose a particular time t = 2.6   */
db_tspan[3] = 5.0   ; /* choose a particular time t = 5.0   */

MWCharArray mw_strfunc  = new MWCharArray(strfunc)      ;
MWNumericArray mw_tspan = new MWNumericArray(db_tspan)  ;

MWArray [] mw_ArrayOut  = null ;
MWNumericArray mw_t     = null ;
MWNumericArray mw_y     = null ;

/* mx_tspan is a column vector when using in the following function */
/* convert Cs double to mxArray */

/* call an implemental function */
ODE objMatlab = new ODE() ;
mw_ArrayOut = objMatlab.myode45firstorder(2, mw_strfunc, mw_tspan, db_y0) ;
mw_t        = (MWNumericArray) mw_ArrayOut[0] ;
mw_y        = (MWNumericArray) mw_ArrayOut[1] ;

/* convert back to Cs double  */
double[] db_t = (double[])mw_t.ToVector(MWArrayComponent.Real);
double[] db_y = (double[])mw_y.ToVector(MWArrayComponent.Real);

Console.WriteLine("The column of time") ;
Console.WriteLine(mw_t) ;

/* or */
PrintVector(db_t) ;

Console.WriteLine("The column of the function values y" ) ;
```

```
    Console.WriteLine(mw_y) ;

    /* or */
    PrintVector(db_y);

    /* free memory */
    mw_strfunc.Dispose() ;
    mw_tspan.Dispose() ;
    mw_t.Dispose() ;
    mw_y.Dispose() ;

    MWNumericArray.DisposeArray(mw_ArrayOut) ;
}

/* **************************** */
public void SecondOrder()
{
    /* Calculating second order ODE */
    String strfunc = "cos(3*t) + 2*yprime + 6*y" ;

    double[] db_ybc = new double [2]    ; /* y boundary conditions            */
    db_ybc[0]  = 0.2    ; /* initial condition of y  at t0    */
    db_ybc[1]  = 1.1    ; /* initial condition of y' at t0    */

    double[] db_tspan = new double [2]    ;
    db_tspan[0] = 1.2    ; /* begin interval t0 = 1.2        */
    db_tspan[1] = 2.5    ; /* end interval, we choose this     */

    /* declare mxArray variables */
    MWCharArray mw_strfunc  = new MWCharArray(strfunc)      ;
    MWNumericArray mw_tspan = new MWNumericArray(db_tspan)  ;
    MWNumericArray mw_ybc   = new MWNumericArray(db_ybc)    ;

    MWArray [] mw_ArrayOut  = null ;
    MWNumericArray mw_t     = null ;
    MWNumericArray mw_y     = null ;
```

```
/* call an implemental function */
ODE objMatlab = new ODE() ;
mw_ArrayOut = objMatlab.myode45secondorder(2, mw_strfunc, mw_tspan, mw_ybc) ;
mw_t        = (MWNumericArray) mw_ArrayOut[0] ;
mw_y        = (MWNumericArray) mw_ArrayOut[1] ;

/* convert back to Cs double  */
double[] db_t  = (double[])  mw_t.ToVector(MWArrayComponent.Real);
double[,] db_y = (double[,]) mw_y.ToArray(MWArrayComponent.Real) ;

Console.WriteLine("The column of time") ;
Console.WriteLine(mw_t) ;

/* or */
// PrintVector(db_t) ;

Console.WriteLine("The column of the function values y :") ;
/* first column of the matrix y */
int i ;
int aLength = db_t.Length ;
for (i=0; i<aLength; i++)
{
    Console.Write("{0} ", db_y.GetValue(i,0).ToString() ) ;
    Console.WriteLine() ;
}

Console.WriteLine("The column of the first derivative y' :") ;
/* second column of the matrix y */
for (i=0; i<aLength; i++)
{
    Console.Write("{0} ", db_y.GetValue(i,1).ToString() ) ;
    Console.WriteLine() ;
}

/* free memory */
```

```
            mw_strfunc.Dispose() ;
            mw_tspan.Dispose() ;
            mw_ybc.Dispose() ;
            mw_t.Dispose() ;
            mw_y.Dispose() ;

            MWNumericArray.DisposeArray(mw_ArrayOut) ;
        }

        /* ******************************** */
        public static void PrintVector(double[] vector)
        {
            int i, row;
            row = vector.GetUpperBound(0) - vector.GetLowerBound(0) + 1;

            for (i = 0; i < row; i++)
            {
                Console.Write("{0} \t", vector.GetValue(i).ToString());
                Console.WriteLine();
            }
        }

        /* **************************** */
        /* **************************** */
        /* **************************** */
    }
}
```

end code

Chapter 7

Integration

In this chapter we'll generate a class ***Integration*** from common M-files working on problems of single and double integrations. The generated functions of this library will be used in a MSVC# .Net 2008 project to solve the integral problems.
The procedure to create the class ***Integration*** is the same as the procedure in Chapter 2.

We will write the M-files `myquad.m` and `mydblquad.m`. These functions will be used to generate the class ***Integration*** .

myquad.m

```
function y = myquad(strfunc, a, b)


F = inline(strfunc) ;
y = quad(F, a, b) ;
```

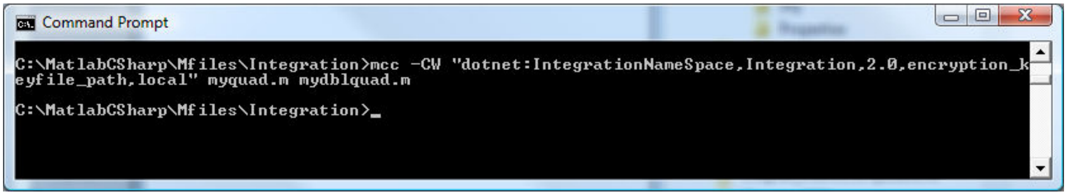
mydblquad.m

```
function dbint = mydblquad(strfunc, x1, x2, y1, y2)


F = inline(strfunc)  ;
dbint = dblquad(F, x1, x2, y1, y2) ;
```

In the following sections, we'll use the following implemental functions in this class to solve the common problems in the integration. The procedure to create the class ***Integration*** is the same as the procedure in Chapter 2 as follows:

1. Write the following command in **Command Prompt** to generate the name space ***IntegrationNameSpace*** that contains the class ***Integration***(see Fig.7.1).

```
mcc -CW "dotnet:IntegrationNameSpace,Integration,2.0,encryption_keyfile_path,
local" myquad.m mydblquad.m
```



```
C:\MatlabCSharp\Mfiles\Integration>mcc -CW "dotnet:IntegrationNameSpace,Integration,2.0,encryption_k
eyfile_path,local" myquad.m mydblquad.m

C:\MatlabCSharp\Mfiles\Integration>_
```

Figure 7.1: Command of creating the class ***Integration***

2. Create a regular C# project in Console Application of Microsoft Visual Studio 2008.

3. Click **Build** then click **Rebuild** to make sure the program works.

4. In the project menu, click **Project**, **Add Reference**. Then the project will pop up a dialog to choose a reference.

5. Click on tab **.Net**, **MathWorks, .NET MWArray API**. Then the project will add MATLAB array wrapper classes for .NET, MWArray into the C# project.

6. Copy the two files `dotnet_mcc_component_data.cs` and `Integration.cs` from previous steps and put them into the same directory of the `Program.cs` file.

7. Copy the file `IntegrationNameSpace.ctf` and put it into the same directory of the project executable file, `Example.exe`.

8. Add these two files to the C# project by clicking **Project** on the project menu, then click **Add Existing Item**. The project will pop up a dialog then choose the two files `dotnet_mcc_component_data.cs` and `Integration.cs`. This step will add these two files to the project.

9. At the top of the file `Program.cs`, add the following:

```
using MathWorks.MATLAB.NET.Arrays;
using MathWorks.MATLAB.NET.Utility;
using IntegrationNameSpace;
```

The following sections show how to use the functions in the class ***Integration*** to solve the common computation problems. The full code is at the end of this chapter.

7.1 Single Integration

In this section, we will use MATLAB functions to solve the following problems.

Problem 1 Calculate the integration :

$$I = \int_0^{3\pi} \Big(\sin(x) + x^2\Big) dx$$

The following is the code to solve Problem 1 by using the functions myquad(..) in the class ***Integration***. The function myquad(..) uses a MATLAB function quad(..) with default as shown in the M-file `myquad.m`. To use more options see this function quad(..) refer to [6].

Listing code

```
public void SingleIntegration()
{
    String strfunc = "sin(x) + x.^2";

    double db_beginInterval = 0;
    double db_endInterval = 3 * Math.PI; // using the value pi in C#

    /* declare mxArray variables */
    MWCharArray mw_strfunc = new MWCharArray(strfunc);
    MWArray[] mw_ArrayOut = null;
    MWNumericArray mw_y = null;

    /* call an implemental function */
    Integration objMatlab = new Integration();
    /* Using Matlab function */
    mw_ArrayOut = objMatlab.myquad(1, mw_strfunc, db_beginInterval, db_endInterval);
    mw_y = (MWNumericArray)mw_ArrayOut[0];
    Console.WriteLine(" I = {0}", mw_ArrayOut);

    /* or */
    Console.WriteLine(" I = {0}", mw_y);

    /* or */
    double db_y = (double)mw_y;
    Console.WriteLine(" I = {0}", db_y);

    /* free memory */
    mw_strfunc.Dispose();
    mw_y.Dispose();

    MWNumericArray.DisposeArray(mw_ArrayOut);
}
```

end code

Note

1. The generated function myquad(..) is a function-function and has an argument as an expression string. The form of this expression string follows the rule of a MATLAB expression string.
2. See the MATLAB function quad(..) for the other method to calculate the single integration.

7.2 Double-Integration

In this section, we will use MATLAB functions to solve the following problems.

Problem 2 Calculate the double-integration:

$$I = \int_0^{\frac{\pi}{2}} \int_0^{\pi} \Big(\sin(x) + x^2 + y^3 \Big) dx\ dy$$

The following is the code to solve Problem 2 using the functions mydblquad(..) in the class ***Integration***. The function mydblquad(..) used a MATLAB function dblquad(..) with an option as shown in the M-file `mydblquad.m`. To use more options see this function dblquad(..) refer to [6].

Listing code

```
public void DoubleIntegration()
{
    String strfunc = "sin(x) + x.^2 + y.^3";

    double db_x1 = 0;
    double db_x2 = 3 * Math.PI; // using the value pi in C#

    double db_y1 = 0;
    double db_y2 = Math.PI;

    /* declare mxArray variables */
    MWCharArray mw_strfunc = new MWCharArray(strfunc);
    MWArray[] mw_ArrayOut = null;
    MWNumericArray mw_II = null;

    /* call an implemental function */
    Integration objMatlab = new Integration();

    /* Using Matlab function */
    mw_ArrayOut = objMatlab.mydblquad(1, mw_strfunc, db_x1, db_x2, db_y1, db_y2);
    mw_II = (MWNumericArray)mw_ArrayOut[0];

    Console.WriteLine(" II = {0}", mw_ArrayOut);

    /* or */
    Console.WriteLine(" II = {0}", mw_II);

    /* or */
    double db_II = (double)mw_II;
    Console.WriteLine(" II = {0}", db_II);

    /* free memory */
    mw_strfunc.Dispose();
    mw_II.Dispose();

    MWNumericArray.DisposeArray(mw_ArrayOut);
}
```

end code

The following is the full code for this chapter.

Listing code

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using MathWorks.MATLAB.NET.Arrays;
using MathWorks.MATLAB.NET.Utility;
using IntegrationNameSpace;

namespace Example
{
  class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine("Single integration:");

      Program objProgram = new Program();
      objProgram.SingleIntegration();

      Console.WriteLine("Double-integration:");
      objProgram.DoubleIntegration();
    }
```

```
/* ******************************************* */
public void SingleIntegration()
{
  String strfunc = "sin(x) + x.^2";

  double db_beginInterval = 0;
  double db_endInterval = 3 * Math.PI; // using the value pi in C#

  /* declare mxArray variables */
  MWCharArray mw_strfunc = new MWCharArray(strfunc);
  MWArray[] mw_ArrayOut = null;
  MWNumericArray mw_y = null;

  /* call an implemental function */
  Integration objMatlab = new Integration();


  /* Using Matlab function */
  mw_ArrayOut = objMatlab.myquad(1, mw_strfunc, db_beginInterval, db_endInterval);
  mw_y = (MWNumericArray)mw_ArrayOut[0];
  Console.WriteLine(" I = {0}", mw_ArrayOut);

  /* or */
  Console.WriteLine(" I = {0}", mw_y);

  /* or */
  double db_y = (double)mw_y;
  Console.WriteLine(" I = {0}", db_y);

  /* free memory */
  mw_strfunc.Dispose();
  mw_y.Dispose();

  MWNumericArray.DisposeArray(mw_ArrayOut);
}
```

```
/* ****************************************** */
public void DoubleIntegration()
{
  String strfunc = "sin(x) + x.^2 + y.^3";

  double db_x1 = 0;
  double db_x2 = 3 * Math.PI; // using the value pi in C#

  double db_y1 = 0;
  double db_y2 = Math.PI;

  /* declare mxArray variables */
  MWCharArray mw_strfunc = new MWCharArray(strfunc);
  MWArray[] mw_ArrayOut = null;
  MWNumericArray mw_II = null;

  /* call an implemental function */
  Integration objMatlab = new Integration();

  /* Using Matlab function */
  mw_ArrayOut = objMatlab.mydblquad(1, mw_strfunc, db_x1, db_x2, db_y1, db_y2);
  mw_II = (MWNumericArray)mw_ArrayOut[0];

  Console.WriteLine(" II = {0}", mw_ArrayOut);

  /* or */
  Console.WriteLine(" II = {0}", mw_II);

  /* or */
  double db_II = (double)mw_II;
  Console.WriteLine(" II = {0}", db_II);

  /* free memory */
  mw_strfunc.Dispose();
  mw_II.Dispose();
```

```
      MWNumericArray.DisposeArray(mw_ArrayOut);
    }

/* ***************************************** */
/* ***************************************** */
/* ***************************************** */
  }
}
```

end code

Chapter 8

Polynomial Fitting and Interpolations

In this chapter we'll generate a class ***PolyInterpolation*** from common M-files working on interpolation curve fitting problems. The generated functions of these libraries will be used in MSVC# .Net 2008 project to solve common polynormial fitting problems. The procedure to create the class ***PolyInterpolation*** is the same as the procedure in Chapter 2.

We will write the M-files as shown below. These functions will be used to generate the class ***PolyInterpolation***.

```
myinterp1.m, myinterp2.m, mypolyfit.m, mypolyval.m,
mymeshgrid.m, mygriddata.m, and myfinemeshgrid.m
```

myinterp1.m

```
function yi = myinterp1(x,y,xi)

yi = interp1(x,y,xi) ;
```

myinterp2.m

```
function ZI = myinterp2(X,Y,Z,XI,YI,method)

ZI = interp2(X,Y,Z,XI,YI,method) ;
```

mypolyfit.m

```
function p = mypolyfit(x,y,n)

p = polyfit(x,y,n) ;
```

mypolyval.m

```
function y = mypolyval(p,x)

y = polyval(p,x) ;
```

mymeshgrid.m

```
function [X,Y] = mymeshgrid(vectorstepx, vectorstepy)

%

[X,Y] = meshgrid( vectorstepx(1):vectorstepx(2):vectorstepx(3), ...
                  vectorstepy(1):vectorstepy(2):vectorstepy(3) ) ;
```

mygriddata.m

```
function ZI = mygriddata(x,y,z,XI,YI)

ZI = griddata(x,y,z,XI,YI) ;
```

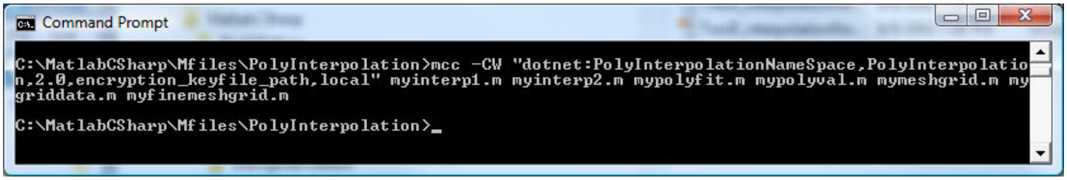
myfinemeshgrid.m

```
function [row,col] = myfinemeshgrid(vectorstepx, vectorstepy)

%Using two colons to create a vector with increments between
%first and end elements.

[X,Y] = meshgrid( vectorstepx(1):vectorstepx(2):vectorstepx(3), ...
                  vectorstepy(1):vectorstepy(2):vectorstepy(3) ) ;

[row, col] = size(X) ;
```

The procedure to create the class ***PolyInterpolation*** is the same as the procedure in Chapter 2 as follows:

1. Write the following command in **Command Prompt** to generate the name space ***PolyInterpolationNameSpace*** that contains the class ***PolyInterpolation*** (see Fig.8.1).

   ```
   mcc -CW "dotnet:PolyInterpolationNameSpace,PolyInterpolation,2.0,encryption_keyfile_path,
   local" myinterp1.m myinterp2.m mypolyfit.m mypolyval.m mymeshgrid.m mygriddata.m
   myfinemeshgrid.m
   ```



```
C:\MatlabCSharp\Mfiles\PolyInterpolation>mcc -CW "dotnet:PolyInterpolationNameSpace,PolyInterpolatio
n,2.0,encryption_keyfile_path,local" myinterp1.m myinterp2.m mypolyfit.m mypolyval.m mymeshgrid.m my
griddata.m myfinemeshgrid.m

C:\MatlabCSharp\Mfiles\PolyInterpolation>_
```

Figure 8.1: Command of creating the class ***PolyInterpolation***

2. Create a regular C# project in Windows Application of Microsoft Visual Studio 2008.
3. Click **Build** then click **Rebuild** to make sure the program works.
4. In the project menu, click **Project**, **Add Reference**. Then the project will pop up a dialog to choose a reference.
5. Click on tab **.Net**, **MathWorks, .NET MWArray API**. Then the project will add MATLAB array wrapper classes for .NET, MWArray into the C# project.
6. Copy the two files `dotnet_mcc_component_data.cs` and `PolyInterpolation.cs` from previous steps and put them into the same directory of the `Program.cs` file.
7. Copy the file `PolyInterpolationNameSpace.ctf` and put it into the same directory of the project executable file, `Example.exe`.
8. Add these two files to the C# project by clicking **Project** on the project menu, then click **Add Existing Item**. The project will pop up a dialog then choose the two files `dotnet_mcc_component_data.cs` and `PolyInterpolation.cs`. This step will add these two files to the project.
9. At the top of the file `Program.cs`, add the following:

```
using MathWorks.MATLAB.NET.Arrays;
using MathWorks.MATLAB.NET.Utility;
using PolyInterpolationNameSpace;
```

8.1 Polynomial Curve Fitting

This section describes how to use the functions in the generated library to to find the coefficients of a polynomial function that fits a set of data in the least-squares sense. An array **c** representing these coefficients is in the polynomial form:

$$f(x) = c_1x^n + c_2x^{n-1} + c_3x^{n-2} + \cdots + c_nx + c_{n+1} \tag{8.1}$$

In this section, we will use MATLAB functions to solve the following problems.

Problem 1

input . There are two arrays **X** and **Y** which have a relationship via a function, $y = f(x)$, $x \in \mathbf{X}$ and $y \in \mathbf{Y}$.

Array **X** = { 1, 2, 3, 4, 5, 6 }

Array **Y** = { 6.8, 50.2, 140.8, 280.5, 321.4, 428.6 }

output . Finding an array **c** of the polynomial function in Eq. 8.1 with degree n=3; since degree n=3, the function $y = f(x)$ has the form:

$$y = c_1x^3 + c_2x^2 + c_3x + c_4$$

. Calculating the interpolation value of the function $y(x)$, at $x = 2.2$

The following is the code to solve Problem 1. In the code, we will use the function *mypolyfit(..)* with degree of $n = 3$ to obtain the coefficient array, and use the function *mypolyval(..)* to calculate the function value.

Listing code

```
public void PolynomialFittingCurve()
{
  double[] db_X = { 1, 2, 3, 4, 5, 6 } ;
  double[] db_Y= { 6.8, 50.2, 140.8, 280.5, 321.4, 428.6 };
```

```
double db_oneValue  = 2.2  ;

/* declare mwArray variables */
MWNumericArray mw_X         = new MWNumericArray(db_X) ;
MWNumericArray mw_Y         = new MWNumericArray(db_Y) ;
MWNumericArray mw_coefs     = null ;
MWNumericArray mw_funcValue = null ;

/* call an implemental function */
/* mw_coefs is a matrix with row = 1 */
PolyInterpolation obj = new PolyInterpolation() ;
mw_coefs = (MWNumericArray) obj.mypolyfit(mw_X, mw_Y, 3) ;

/* convert back to Cs double  */
double[] db_coefs = (double[]) mw_coefs.ToVector(MWArrayComponent.Real) ;

/* print out */
Console.Write("The coefficient value: {0} \n", mw_coefs ) ;

/* print out */
Console.WriteLine("The polynomial:" ) ;
Console.Write( " {0}x^3 + ",  db_coefs[0].ToString() ) ;
Console.Write( " {0}x^2 + ",  db_coefs[1].ToString() ) ;
Console.Write(   " {0}x + ",  db_coefs[2].ToString() ) ;
Console.Write(                db_coefs[3].ToString() ) ;
Console.WriteLine() ;

/* calculate the function value at oneValue */
mw_funcValue = (MWNumericArray) obj.mypolyval(mw_coefs, db_oneValue) ;

double db_funcValue = (double)mw_funcValue ;

Console.Write("The function value at 2.2 is: {0} \n", mw_funcValue      ) ;
Console.Write("The function value at 2.2 is: {0} \n", db_funcValue.ToString() ) ;

/* free memory */
```

```
  mw_X.Dispose()       ;
  mw_Y.Dispose()       ;
  mw_coefs.Dispose()     ;
  mw_funcValue.Dispose()   ;
}
```

end code

8.2 One-Dimensional Polynomial Interpolation

This section describes how to use the functions in the generated class to solve an one-dimensional interpolation problem. This function uses polynomial techniques to evaluate the value of a function at a desired interpolation point.

Problem 2

input . The two two arrays **X** and **Y** have a relationship via a function, $y = f(x)$, $x \in \mathbf{X}$ and $y \in \mathbf{Y}$.

Array **X** = { 1, 2, 3, 4, 5, 6, 7, 8 }

Array **Y** = { 6.8, 24.6, 50.2, 74, 140.8, 280.5, 321.4, 428.6 }

output . Finding the interpolation function value at $x = a$, where a = 2.1

The following is the code to solve Problem 2. In the code, you will use the function *myinterp1(..)* with the default method (liner method) to solve the problem. To learn more about other possible methods, see the MATLAB function *inter1(..)* in [5].

Listing code

```
public void OneDimensionInterpolation()
{
  double[] db_X = { 1, 2, 3, 4, 5, 6, 7, 8 } ;
  double[] db_Y = { 6.8, 24.6, 50.2, 74, 140.8, 280.5, 321.4, 428.6 };

  double db_oneValue  = 2.1  ;

  /* declare mwArray variables */
  MWNumericArray mw_X  = new MWNumericArray(db_X) ;
  MWNumericArray mw_Y  = new MWNumericArray(db_Y) ;
  MWNumericArray mw_funcValue = null ;

  /* call an implemental function */
  PolyInterpolation obj = new PolyInterpolation() ;
  mw_funcValue = (MWNumericArray) obj.myinterp1(mw_X, mw_Y, db_oneValue) ;

  /* print out */
  Console.WriteLine( "The function value at 2.1 is:  {0}", mw_funcValue );

  /* convert back to Cs double  */
  double db_funcValue = (double) mw_funcValue ;
  Console.WriteLine( "The function value at 2.1 is:  {0}", db_funcValue.ToString() );

  /* free memory */
  mw_X.Dispose()       ;
  mw_Y.Dispose()       ;
  mw_funcValue.Dispose()   ;
}
```

end code

8.3 Two-Dimensional Polynomial Interpolation for Grid Points

This section describes how to use the functions in the generated class to solve a two-dimensional interpolation problem. This function uses polynomial techniques to evaluate the value of a function at a desired interpolation point.

Problem 3

input . There are three matrixes, **x**, **y**, and **z**, that have a relationship via a function, $z = f(x, y)$, $x \in \mathbf{x}$, $y \in \mathbf{y}$, and $z \in \mathbf{z}$

. Representation of the grid points (x,y) is two matrixes:

matrix **x** contains the x-direction values of all grid points matrix **y** contains the y-direction values of all grid points

. Matrix **z** contains the function values $z(x, y)$ of all grid points
(the values of matrixes **x**, **y**, and **z** are shown below)

output . Finding the interpolation function value $z(x, y)$ at a particular point,
($x = a = 2.3,\ y = b = 0.7$)

A. Assigning values for a matrix

Suppose that we have grid points as in Fig. 8.2.

The matrix **x**, which contains the x-direction values for all grid points, is:

```
                                              ( from left to right )
 -3    -2    -1     0     1     2     3         from point 1  to 7
 -3    -2    -1     0     1     2     3         from point 8  to 14
 -3    -2    -1     0     1     2     3         from point 15 to 21
 -3    -2    -1     0     1     2     3         from point 22 to 28
 -3    -2    -1     0     1     2     3         from point 29 to 35
 -3    -2    -1     0     1     2     3         from point 36 to 42
 -3    -2    -1     0     1     2     3         from point 43 to 49
```

The matrix **y**, which contains the y-direction values of all grid points, is:

```
                                                ( from left to right )
   -3    -3    -3    -3    -3    -3    -3         from point 1  to 7
   -2    -2    -2    -2    -2    -2    -2         from point 8  to 14
   -1    -1    -1    -1    -1    -1    -1         from point 15 to 21
    0     0     0     0     0     0     0         from point 22 to 28
    1     1     1     1     1     1     1         from point 29 to 35
    2     2     2     2     2     2     2         from point 36 to 42
    3     3     3     3     3     3     3         from point 43 to 49
```

The matrix **z**, which contains the function values $z(x, y)$ for all grid points, is:

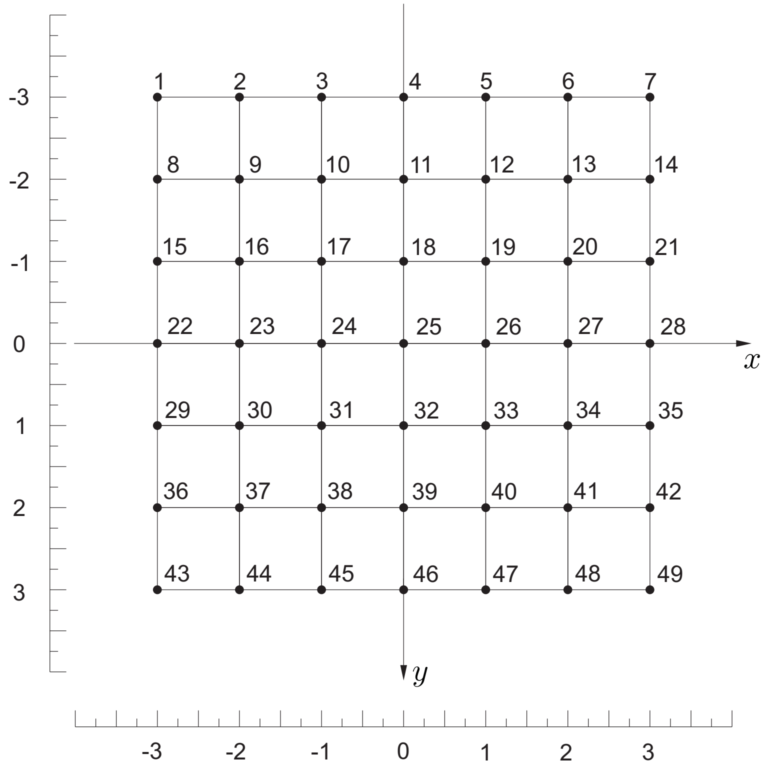


Figure 8.2: Grid points

```
                                                          ( from left to right )
 0.0001   0.0034  -0.0299   -0.2450  -0.1100  -0.0043  -0.0000 (point 1 - 7)
 0.0007   0.0468  -0.5921   -4.7596  -2.1024  -0.0616   0.0004 (point 8 -14)
 0.0088  -0.1301   1.8559   -0.7239  -0.2729   0.4996   0.0130 (point 15-21)
 0.0365  -1.3327  -1.6523    0.9810   2.9369   1.4122   0.0331 (point 22-28)
 0.0137  -0.4808   0.2289    3.6886   2.4338   0.5805   0.0125 (point 29-35)
 0.0000   0.0797   2.0967    5.8591   2.2099   0.1328   0.0013 (point 36-42)
 0.0000   0.0053   0.1099    0.2999   0.1107   0.0057   0.0000 (point 43-49)
```

B. Programming code

The following code is used to solve Problem 3. In the code, you will use the generated function *myinterp2(..)* with the *cubic* method to solve the problem. To learn more about other possible methods see the MATLAB function *inter2(..)* in [5]. The procedure of this code is:

1. Assign values for matrix **x** and matrix **y**
2. Assign values for matrix **z**
3. Use the two-dimensional interpolation function to evaluate the value at the point (a, b)

Listing code

```
public void TwoDimensionsInterpolation()
{
  /* function values z at 47 points, (x_i, y_j) are : */
  /* matrix z values*/
  double[,] z = {
  {  0.0001,  0.0034, -0.0299, -0.2450, -0.1100, -0.0043,  0.0000  }  ,
  {  0.0007,  0.0468, -0.5921, -4.7596, -2.1024, -0.0616,  0.0004  }  ,
  { -0.0088, -0.1301,  1.8559, -0.7239, -0.2729,  0.4996,  0.0130  }  ,
  { -0.0365, -1.3327, -1.6523,  0.9810,  2.9369,  1.4122,  0.0331  }  ,
  { -0.0137, -0.4808,  0.2289,  3.6886,  2.4338,  0.5805,  0.0125  }  ,
  {  0.0000,  0.0797,  2.0967,  5.8591,  2.2099,  0.1328,  0.0013  }  ,
  {  0.0000,  0.0053,  0.1099,  0.2999,  0.1107,  0.0057,  0.0000  }  };

  double[] db_vectorstep = {-3, 1, 3} ;
  double db_a = 2.3 ;
  double db_b = 0.7 ;

  /* declare mwArray variables */
  String interp2method = "cubic";

  MWNumericArray  mw_interp2z  = null  ;
  MWNumericArray  mw_z  = new MWNumericArray(z)  ;
  MWNumericArray  mw_x  = null  ;
  MWNumericArray  mw_y  = null  ;

  /* same for two step-vectors */
  MWNumericArray  mw_vectorstepx  = new MWNumericArray(db_vectorstep) ;
  MWNumericArray  mw_vectorstepy  = new MWNumericArray(db_vectorstep) ;

  MWCharArray   mw_method = new MWCharArray(interp2method)  ;
  MWArray [] mw_ArrayGrid = null ;

  /* call implemental functions */
  PolyInterpolation obj = new PolyInterpolation() ;

  /* create values for the matrix x and matrix y */
  mw_ArrayGrid = obj.mymeshgrid(2, mw_vectorstepx, mw_vectorstepy) ;

  mw_x = (MWNumericArray) mw_ArrayGrid[0] ;
  mw_y = (MWNumericArray) mw_ArrayGrid[1] ;

  mw_interp2z = (MWNumericArray) obj.myinterp2(mw_x, mw_y, mw_z, db_a, db_b, mw_method);

  /* convert back to Cs double  */
  double[,] db_x = (double[,]) mw_x.ToArray(MWArrayComponent.Real) ;
  double[,] db_y = (double[,]) mw_y.ToArray(MWArrayComponent.Real) ;

  /* print out */
  Console.WriteLine("Matrix x {0}: \n", mw_x ) ;

  /* or */
```

```
  PrintMatrix(db_x) ;

  Console.WriteLine("Matrix y {0}: \n", mw_y ) ;

  /* or */
  PrintMatrix(db_y) ;

  Console.WriteLine("Interpolation in two dimensions with cubic method " ) ;
  Console.WriteLine(" z = {0}", mw_interp2z );

  double db_interp2z = (double) mw_interp2z ;
  Console.WriteLine(" z = {0}", db_interp2z.ToString() );

  /* free memory */
  mw_interp2z  = null  ;
  mw_z.Dispose() ;
  mw_x.Dispose() ;
  mw_y.Dispose() ;
  mw_vectorstepx.Dispose() ;
  mw_vectorstepy.Dispose() ;
  mw_method.Dispose() ;

  MWNumericArray.DisposeArray(mw_ArrayGrid) ;
}
```

end code

Remarks

1. The M-file, `mymeshgrid.m`, uses the MATLAB function *meshgrid(..)*, which assigns values for matrixes **x** and **y**. These values are assigned from left to right, and from top to bottom.

   In Fig. 8.2, the y axis direction is from top to bottom, therefore pay attention when assigning your data into the matrix **y**.

2. The matrix **z** values will be assigned according to the same rules (left to right, top to bottom) as matrixes **x** and **y** (Fig. 8.3). Therefore to avoid mistakes and to have the

convenience of visibility, you can set up your matrix data as in Fig. 8.2 (**y axis direction from top to bottom**).

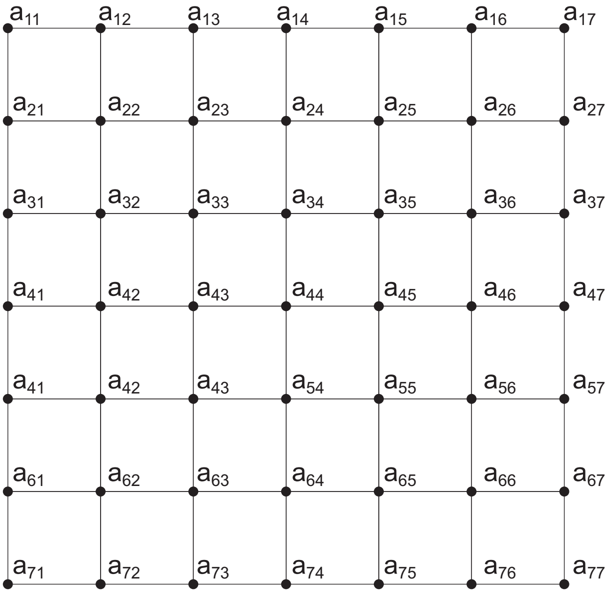


Figure 8.3: A matrix form for grid points in two-dimensional interpolation

3. To receive a better solution in the two-dimensional interpolation you can create a fine grid by using the function mygriddata(..) (see the M-file mygriddata.m in the beginning of this chapter). This functions uses the MATLAB griddata(..) function which fits a surface of the form $z = f(x, y)$ to the data in the spaced vectors (x, y, z). For more information on this function, refer to the MATLAB manual [5]. The function `TwoDimensionsInterpolationFineSolution()` in the end of this chapter finds the fine solution of Problem 3 by using the function mygriddata(..).

The following is the full code for this chapter.

Listing code

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using MathWorks.MATLAB.NET.Arrays;
using MathWorks.MATLAB.NET.Utility;
using PolyInterpolationNameSpace;

namespace Example
{
  class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine("Curve Fitting") ;
      Program objProgram = new Program() ;

      Console.WriteLine() ;
      Console.WriteLine("1. Polynomial") ;
      objProgram.PolynomialFittingCurve() ;

      Console.WriteLine() ;
      Console.WriteLine("2. One-dimensional interpolation" ) ;
      objProgram.OneDimensionInterpolation() ;

      Console.WriteLine() ;
      Console.WriteLine("3. Two-dimensional interpolation") ;
      objProgram.TwoDimensionsInterpolation() ;

      Console.WriteLine() ;
      Console.WriteLine("4. Two-dimensional interpolation with fine grid") ;
      objProgram.TwoDimensionsInterpolationFineSolution() ;
    }

    /* ******************* */
    public void PolynomialFittingCurve()
    {
      double[] db_X = { 1, 2, 3, 4, 5, 6 } ;
```

```
double[] db_Y= { 6.8, 50.2, 140.8, 280.5, 321.4, 428.6 };

double db_oneValue  = 2.2  ;

/* declare mwArray variables */
MWNumericArray mw_X         = new MWNumericArray(db_X) ;
MWNumericArray mw_Y         = new MWNumericArray(db_Y) ;
MWNumericArray mw_coefs     = null ;
MWNumericArray mw_funcValue = null ;

/* call an implemental function */
/* mw_coefs is a matrix with row = 1 */
PolyInterpolation obj = new PolyInterpolation() ;
mw_coefs = (MWNumericArray) obj.mypolyfit(mw_X, mw_Y, 3) ;

/* convert back to Cs double  */
double[] db_coefs = (double[]) mw_coefs.ToVector(MWArrayComponent.Real) ;

/* print out */
Console.Write("The coefficient value: {0} \n", mw_coefs ) ;

/* print out */
Console.WriteLine("The polynomial:" ) ;
Console.Write( " {0}x^3 + ",  db_coefs[0].ToString() ) ;
Console.Write( " {0}x^2 + ",  db_coefs[1].ToString() ) ;
Console.Write( " {0}x + "  ,  db_coefs[2].ToString() ) ;
Console.Write(                db_coefs[3].ToString() ) ;
Console.WriteLine() ;

/* calculate the function value at oneValue */
mw_funcValue = (MWNumericArray) obj.mypolyval(mw_coefs, db_oneValue) ;

double db_funcValue = (double)mw_funcValue ;

Console.Write("The function value at 2.2 is: {0} \n", mw_funcValue ) ;
Console.Write("The function value at 2.2 is: {0} \n",
                                             db_funcValue.ToString() ) ;
      /* free memory */
      mw_X.Dispose()       ;
      mw_Y.Dispose()       ;
      mw_coefs.Dispose()     ;
      mw_funcValue.Dispose()   ;
    }

    /* ************************** */
    public void OneDimensionInterpolation()
    {
      double[] db_X = { 1, 2, 3, 4, 5, 6, 7, 8 } ;
      double[] db_Y = { 6.8, 24.6, 50.2, 74, 140.8, 280.5, 321.4, 428.6 };

      double db_oneValue  = 2.1  ;

      /* declare mwArray variables */
      MWNumericArray mw_X  = new MWNumericArray(db_X) ;
      MWNumericArray mw_Y  = new MWNumericArray(db_Y) ;
      MWNumericArray mw_funcValue = null ;

      /* call an implemental function */
      PolyInterpolation obj = new PolyInterpolation() ;
      mw_funcValue = (MWNumericArray) obj.myinterp1(mw_X, mw_Y, db_oneValue) ;

      /* print out */
      Console.WriteLine( "The function value at 2.1 is: {0}", mw_funcValue );

      /* convert back to Cs double  */
      double db_funcValue = (double) mw_funcValue ;
      Console.WriteLine( "The function value at 2.1 is: {0}",
                                          db_funcValue.ToString() );
      /* free memory */
      mw_X.Dispose()       ;
      mw_Y.Dispose()       ;
      mw_funcValue.Dispose()   ;
```

```
}

/* *************************** */
public void TwoDimensionsInterpolation()
{
  /* function values z at 47 points, (x_i, y_j) are : */
  /* matrix z values*/
double[,] z = {
{  0.0001,  0.0034, -0.0299, -0.2450, -0.1100, -0.0043,  0.0000  }  ,
{  0.0007,  0.0468, -0.5921, -4.7596, -2.1024, -0.0616,  0.0004  }  ,
{ -0.0088, -0.1301,  1.8559, -0.7239, -0.2729,  0.4996,  0.0130  }  ,
{ -0.0365, -1.3327, -1.6523,  0.9810,  2.9369,  1.4122,  0.0331  }  ,
{ -0.0137, -0.4808,  0.2289,  3.6886,  2.4338,  0.5805,  0.0125  }  ,
{  0.0000,  0.0797,  2.0967,  5.8591,  2.2099,  0.1328,  0.0013  }  ,
{  0.0000,  0.0053,  0.1099,  0.2999,  0.1107,  0.0057,  0.0000  }  };

  double[] db_vectorstep = {-3, 1, 3} ;
  double db_a = 2.3 ;
  double db_b = 0.7 ;

  /* declare mwArray variables */
  String interp2method = "cubic";

  MWNumericArray  mw_interp2z  = null  ;
  MWNumericArray  mw_z  = new MWNumericArray(z)  ;
  MWNumericArray  mw_x  = null  ;
  MWNumericArray  mw_y  = null  ;

  /* same for two step-vectors */
  MWNumericArray  mw_vectorstepx  = new MWNumericArray(db_vectorstep) ;
  MWNumericArray  mw_vectorstepy  = new MWNumericArray(db_vectorstep) ;

  MWCharArray   mw_method = new MWCharArray(interp2method)  ;
  MWArray [] mw_ArrayGrid = null ;

  /* call implemental functions */
```

```
PolyInterpolation obj = new PolyInterpolation() ;

/* create values for the matrix x and matrix y */
mw_ArrayGrid = obj.mymeshgrid(2, mw_vectorstepx, mw_vectorstepy) ;

mw_x = (MWNumericArray) mw_ArrayGrid[0] ;
mw_y = (MWNumericArray) mw_ArrayGrid[1] ;

mw_interp2z = (MWNumericArray) obj.myinterp2(mw_x, mw_y, mw_z, db_a, db_b, mw_method);

/* convert back to Cs double  */
double[,] db_x = (double[,]) mw_x.ToArray(MWArrayComponent.Real) ;
double[,] db_y = (double[,]) mw_y.ToArray(MWArrayComponent.Real) ;

/* print out */
Console.WriteLine("Matrix x {0}: \n", mw_x ) ;

/* or */
PrintMatrix(db_x) ;

Console.WriteLine("Matrix y {0}: \n", mw_y ) ;

/* or */
PrintMatrix(db_y) ;

Console.WriteLine("Interpolation in two dimensions with cubic method " ) ;
Console.WriteLine(" z = {0}", mw_interp2z );

double db_interp2z = (double) mw_interp2z ;
Console.WriteLine(" z = {0}", db_interp2z.ToString() );

/* free memory */
mw_interp2z  = null  ;
mw_z.Dispose() ;
mw_x.Dispose() ;
mw_y.Dispose() ;
```

```
    mw_vectorstepx.Dispose() ;
    mw_vectorstepy.Dispose() ;
    mw_method.Dispose() ;

    MWNumericArray.DisposeArray(mw_ArrayGrid) ;
  }

  /* **************************** */
  public void TwoDimensionsInterpolationFineSolution()
  {
    /* function values z, (x_i, y_j) are : */
    /* matrix z values*/

    double[,] z = {
{  0.0001,  0.0034, -0.0299, -0.2450, -0.1100, -0.0043,  0.0000  }  ,
{  0.0007,  0.0468, -0.5921, -4.7596, -2.1024, -0.0616,  0.0004  }  ,
{ -0.0088, -0.1301,  1.8559, -0.7239, -0.2729,  0.4996,  0.0130  }  ,
{ -0.0365, -1.3327, -1.6523,  0.9810,  2.9369,  1.4122,  0.0331  }  ,
{ -0.0137, -0.4808,  0.2289,  3.6886,  2.4338,  0.5805,  0.0125  }  ,
{  0.0000,  0.0797,  2.0967,  5.8591,  2.2099,  0.1328,  0.0013  }  ,
{  0.0000,  0.0053,  0.1099,  0.2999,  0.1107,  0.0057,  0.0000  }  };

    double[] db_vectorstep     = {-3,  1 , 3} ; // interval = 1
    double[] db_finevectorstep = {-3, 0.2, 3} ; // interval = 0.2

    double db_a = 2.3 ;
    double db_b = 0.7 ;
    String interp2method = "cubic";

    /* declare mwArray variables */
    MWNumericArray  mw_interp2z  = null  ;
    MWNumericArray  mw_z    = new MWNumericArray(z)  ;
    MWNumericArray  mw_finez  = new MWNumericArray(z)  ;

    MWNumericArray  mw_x    = null  ;
    MWNumericArray  mw_y    = null  ;
    MWNumericArray  mw_finex  = null  ;
    MWNumericArray  mw_finey  = null  ;

    /* same for two step-vectors */
    MWNumericArray  mw_vectorstepx  = new MWNumericArray(db_vectorstep) ;
    MWNumericArray  mw_vectorstepy  = new MWNumericArray(db_vectorstep) ;

    MWCharArray   mw_method     = new MWCharArray(interp2method)  ;

    MWNumericArray  mw_finevectorstepx = new MWNumericArray(db_finevectorstep)  ;
    MWNumericArray  mw_finevectorstepy = new MWNumericArray(db_finevectorstep)  ;

    MWArray [] mw_ArrayGridXY     = null ;

    MWArray [] mw_ArrayMatrixXY     = null ;
    MWArray [] mw_ArrayMatrixXY_fine  = null ;

    MWArray [] mw_ArrayGridXY_fine  = null ;
    MWArray [] mw_ArrayFineVector   = null ;

    /* call implemental functions */
    /* get size for fine matrixes */
    PolyInterpolation obj = new PolyInterpolation() ;

    /* create values for the matrix x and matrix y */
    mw_ArrayMatrixXY = obj.mymeshgrid(2, mw_vectorstepx, mw_vectorstepy) ;
    mw_x = (MWNumericArray) mw_ArrayMatrixXY[0] ;
    mw_y = (MWNumericArray) mw_ArrayMatrixXY[1] ;

    //Console.WriteLine(" see normal x = {0} \n",  mw_x ) ;

    /* create values for the fine matrix x and fine matrix y */
    mw_ArrayMatrixXY_fine = obj.mymeshgrid(2, mw_finevectorstepx, mw_finevectorstepy) ;
    mw_finex = (MWNumericArray) mw_ArrayMatrixXY_fine[0] ;
    mw_finey = (MWNumericArray) mw_ArrayMatrixXY_fine[1] ;
```

```
//Console.WriteLine(" see fine x = {0}",  mw_finex ) ;

/* get a fine matrix mx_finez from mx_z */
mw_finez = (MWNumericArray) obj.mygriddata(mw_x, mw_y, mw_z, mw_finex, mw_finey);

//Console.WriteLine(" see fine z = {0} \n",  mw_finez ) ;

mw_interp2z = (MWNumericArray) obj.myinterp2(mw_finex, mw_finey, mw_finez, db_a,
                                                                 db_b, mw_method);
/* convert back to Cs double  */
double db_interp2finez = (double) mw_interp2z ;

Console.Write(@"Interpolation in two dimensions with cubic method ");
Console.WriteLine(@"and a fine grid");
Console.WriteLine(" z = {0}",  mw_interp2z ) ;
Console.WriteLine(" z = {0}",  db_interp2finez.ToString() ) ;

/* free memory */
mw_interp2z.Dispose() ;
mw_z.Dispose() ;
mw_finez.Dispose() ;

mw_x.Dispose() ;
mw_y.Dispose() ;

mw_vectorstepx.Dispose() ;
mw_vectorstepy.Dispose() ;
mw_method.Dispose() ;

mw_finevectorstepx.Dispose() ;
mw_finevectorstepy.Dispose() ;

MWNumericArray.DisposeArray(mw_ArrayGridXY)     ;
MWNumericArray.DisposeArray(mw_ArrayGridXY_fine)  ;
MWNumericArray.DisposeArray(mw_ArrayFineVector)   ;
```

```
    }

    /* **************************************** */
    public static void PrintMatrix(double[,] matrix)
    {
      int i, j, row, col ;
      row = matrix.GetUpperBound(0) - matrix.GetLowerBound(0) + 1;
      col = matrix.GetUpperBound(1) - matrix.GetLowerBound(1) + 1;

      for (i=0; i<row; i++)
      {
        for (j=0; j<col; j++)
        {
          Console.Write("{0} \t", matrix.GetValue(i,j).ToString() ) ;
        }
        Console.WriteLine() ;
      }
    }

/* **************************************** */
/* **************************************** */
/* **************************************** */
  }
}
```

end code

Chapter 9

Using MATLAB Curve Fitting Toolbox In C# Functions

In this chapter we'll generate a class ***CurveFittings*** from M-files to use functions of Curve Fitting Toolbox in C# functions. The generated functions of this class will be used in a MSVC# .Net 2008 project to plot curves and show information of a curve fitting.

Note that, in order to run applications in this chapter you need to have MATLAB Curve Fitting Toolbox in your computer.

We will write the M-files as shown below to generate that class.

`CurveFitting.m, mytextread.m, mytranspose.m, CurveFittingWithPlots.m`

CurveFitting.m

```
function varargout = CurveFitting(x,y,cftLib)


[cft_data,cft_info] = fit(x,y,cftLib);
formula_a    = formula(cft_data)     ;
varargout{1} = formula_a ;


coeff_names  = coeffnames(cft_data)  ;
varargout{2} = coeff_names ;
```

```
coeff_values = coeffvalues(cft_data) ;
varargout{3} = coeff_values ;

conf_int     = confint(cft_data)     ;
varargout{4} = conf_int ;

varargout{5} = cft_info ;
```

CurveFittingWithPlots.m

```
function  varargout = CurveFittingWithPlots(x,y,cftLib)

%%%%%%% Curve fit data %%%%%%%%%
[cft_data,cft_info] = fit(x,y,cftLib);
formula_a    = formula(cft_data)     ;
varargout{1} = formula_a ;

coeff_names  = coeffnames(cft_data)  ;
varargout{2} = coeff_names ;

coeff_values = coeffvalues(cft_data) ;
varargout{3} = coeff_values ;

conf_int     = confint(cft_data)     ;
varargout{4} = conf_int ;

varargout{5} = cft_info ;
%%%%%%%%%%%%%%%%%

close all ;
cftResult = fit(x,y, cftLib);

Hcfit = plot(cftResult) ;
hold on ; % require here: after plot result
```

```
set(Hcfit,'Color','red','Marker','.','LineStyle','-') ;

Hcfit_legend = 'Fitting curve';

Hdata = plot(x,y) ;
set(Hdata, 'Color', 'blue','Marker','*','LineStyle','none') ;
Hdata_legend = 'data' ;

Hlegend = legend(Hcfit_legend, Hdata_legend) ;
set(Hlegend, 'FontWeight', 'bold') ;

hold off ;

grid on ;
```

CurveFittingWithPlotsAdvance.m

```
function  varargout = CurveFittingWithPlotsAdvance(x,y,cftLib, ...
                                          graTitle, xlab, ylab, ...
                       cftColor, dataColor, cftLegend, dataLengend)

%%%%%%% Curve fit data %%%%%%%%%
[cft_data,cft_info] = fit(x,y,cftLib);
formula_a    = formula(cft_data)     ;
varargout{1} = formula_a ;

coeff_names  = coeffnames(cft_data)  ;
varargout{2} = coeff_names ;

coeff_values = coeffvalues(cft_data) ;
varargout{3} = coeff_values ;

conf_int     = confint(cft_data)     ;
varargout{4} = conf_int ;
```

```
varargout{5} = cft_info ;
%%%%%%%%%%%%%%%%%

close all ;
cftResult = fit(x,y, cftLib);

Hcfit = plot(cftResult) ;
hold on ; % require here: after plot result

set(Hcfit,'Color',cftColor,'Marker','.','LineStyle','-') ;

Hcfit_legend = cftLegend ;

Hdata = plot(x,y) ;
set(Hdata,'Color',dataColor,'Marker','*','LineStyle','none') ;
Hdata_legend = dataLengend ;

Hlegend = legend(Hcfit_legend, Hdata_legend) ;
set(Hlegend, 'FontWeight', 'bold') ;

title(graTitle) ;
xlabel(xlab) ;
ylabel(ylab) ;



hold off ;

grid on ;
```

mytextread.m

```
function varargout = mytextread(fileName, colNum, mydelimiter)

  switch colNum
```

```
    case 2
    [A1, A2] = textread(fileName, '%f%f','delimiter', mydelimiter) ;
    varargout{1} = A1 ; varargout{2} = A2 ;

    case 3
    [A1, A2, A3] = textread(fileName, '%f%f%f', 'delimiter', mydelimiter) ;
    varargout{1} = A1 ; varargout{2} = A2 ; varargout{3} = A3 ;

    case 4
    [A1, A2, A3, A4] = textread(fileName, '%f%f%f%f', 'delimiter', mydelimiter) ;
    varargout{1} = A1 ; varargout{2} = A2 ; varargout{3} = A3 ;
    varargout{4} = A4 ;

    case 5
    [A1, A2, A3, A4, A5] = textread(fileName, '%f%f%f%f%f', 'delimiter', mydelimiter) ;
    varargout{1} = A1 ; varargout{2} = A2 ; varargout{3} = A3 ;
    varargout{4} = A4 ; varargout{5} = A5 ;

    otherwise
        disp('This function reads files with max colums = 5.');

  end
```

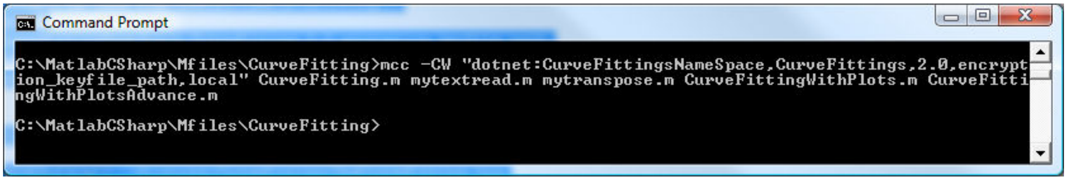
mytranspose.m

```
function y = mytranspose( x )

y = x' ;
```

The following procedure to create the class ***CurveFitting*** is the same as the procedure in Chapter 2 as follows:

1. Write the following command in **Command Prompt** to generate the name space ***CurveFittingNameSpace*** that contains the class ***CurveFitting*** (see Fig.9.2).

```
mcc -CW "dotnet:CurveFittingsNameSpace,CurveFittings,2.0,encryption_keyfile_path,
local" CurveFitting.m mytextread.m mytranspose.m CurveFittingWithPlots.m
CurveFittingWithPlotsAdvance.m
```



```
C:\MatlabCSharp\Mfiles\CurveFitting>mcc -CW "dotnet:CurveFittingsNameSpace,CurveFittings,2.0,encrypt
ion_keyfile_path,local" CurveFitting.m mytextread.m mytranspose.m CurveFittingWithPlots.m CurveFitti
ngWithPlotsAdvance.m

C:\MatlabCSharp\Mfiles\CurveFitting>
```

Figure 9.1: Command of creating the class ***CurveFitting***

2. Create a regular C# project in Windows Application of Microsoft Visual Studio 2008.
3. Click **Build**, then click **Rebuild** to make sure the program works.
4. In the project menu, click **Project**, **Add Reference**. Then the project will pop up a dialog to choose a reference.
5. Click on tab **.Net**, **MathWorks, .NET MWArray API**. Then the project will add MATLAB array wrapper classes for .NET, MWArray into the C# project.
6. Copy the two files `dotnet_mcc_component_data.cs` and `CurveFitting.cs` from the previous steps and put them into the same directory of the `Program.cs` file.
7. Copy the file `CurveFittingNameSpace.ctf` and put it into the same directory of the project executable file, `Example.exe`.
8. Add these two files to the C# project by click **Project** on the project menu, then click **Add Existing Item**. The project will pop up a dialog then choose the two files `dotnet_mcc_component_data.cs` and `CurveFitting.cs`. This step will add these two files to the project.
9. At the top of the file `Program.cs`, add the following:

```
using MathWorks.MATLAB.NET.Arrays;
using MathWorks.MATLAB.NET.Utility;
using CurveFittingNameSpace;
```

9.1 Using Curve Fitting Toolbox functions in C#

Problem 1

input . There are two data arrays:

```
X = {0, 50, 100, 150, 200, 250, 300, 350, 400, 450, 500, 550, 600} ;
Y = {0.4000, 0.2426, 0.1472, 0.0893, 0.0541, 0.0328, 0.0199,
                 0.0121, 0.0073, 0.0044, 0.0027, 0.0016,  0.0010 } ;
```

output . Find a Curve Fitting exponent function and its information, with the formula of exponent function:

$$a * exp(b * x) + c * exp(d * x)$$

The code below of the subroutine *CurveFittingInfo()* is for this problem. The result output is:

```
Curve fitting formula :
a*exp(b*x) + c*exp(d*x)

Coefficient names:
    'a'
    'b'
    'c'
    'd'

Coefficient values:
-3.04017687611051E-06
0.0031266697523144
0.400008836352447
-0.00999952268115858

Confidence interval:
-4.84852291943982E-05 -0.0248717641096469 0.399931839884673  -0.0100046533755882
4.24048754421772E-05   0.0311251036142757 0.400085832820221  -0.00999439198672901
```

```
Error info:
           sse: 1.0092e-008
       rsquare: 1.0000
           dfe: 9
    adjrsquare: 1.0000
          rmse: 3.3486e-005
```

The output has five variables as described in the following paragraphs.

1. The first variable is a string of function formula:

$$a * exp(b * x) + c * exp(d * x)$$

2. The second variable is a string of coefficient names. In this function, the coefficient names are a, b, c, and d.

3. The third variable is the value of the coefficient names above:

$$a = -3.04017687611051E - 06$$
$$b = 0.0031266697523144$$
$$c = 0.400008836352447$$
$$d = -0.00999952268115858$$

4. The fourth variable is value of confidence bounds at the confidence level specified by level(level must be between 0 and 1). In this work, we choose the default value of level as 0.95 (95%).

   In this work, the confidence bounds is a matrix in which each column is bound of each coefficient value. The first column is bound for the coefficient a. Second column is bound for coefficient b and so on.

5. The fifth variable is error information of this curve fitting function for the original data.

```
sse          sum of squares due to error
rsquare      coefficient of determination or R^2
```

```
dfe          error Degrees of Freedom.
  The DFE is equal to the number of data points minus the number
  of fitted coefficients.
adjrsquare   degree of freedom adjusted R^2
rmse         fit standard error or root mean square error
```

If we solve this problem in MATLAB Command Prompt we will have the following solution.

```
General model Exp2:
    a*exp(b*x) + c*exp(d*x)
Coefficients (with 95% confidence bounds):
  a =  -3.04e-006  (-4.849e-005, 4.24e-005)
  b =    0.003127  (-0.02487, 0.03113)
  c =         0.4  (0.3999, 0.4001)
  d =       -0.01  (-0.01, -0.009994)
```

The following code is used to solve Problem 1 by using the function in MATLAB Curve Fitting Toolbox.

Listing code

```
public void CurveFittingInfo()
{
    double[]X = {0, 50, 100, 150, 200, 250, 300, 350, 400, 450, 500, 550, 600} ;

    double[]Y = {0.4000, 0.2426, 0.1472, 0.0893, 0.0541, 0.0328, 0.0199,
                 0.0121, 0.0073, 0.0044, 0.0027, 0.0016,  0.0010 } ;

    MWNumericArray mw_X = new MWNumericArray(X) ;
    MWNumericArray mw_Y = new MWNumericArray(Y) ;

    MWArray [] mw_ArrayOut              = null ;

    MWCharArray mwChar_formula          = null ;
    MWCellArray mwCell_coeffnames       = null ;
    MWNumericArray mwDouble_coeffvalues = null ;
    MWNumericArray mwDouble_confint     = null ;

    MWStructArray mwStructure_info      = null ;

    String curveFitLib = "exp2" ;

    CurveFittings objMatlab = new CurveFittings() ;
    mw_X = (MWNumericArray)objMatlab.mytranspose(mw_X) ;
    mw_Y = (MWNumericArray)objMatlab.mytranspose(mw_Y) ;

    mw_ArrayOut = objMatlab.CurveFitting(5,mw_X, mw_Y, curveFitLib) ;

    mwChar_formula       = (MWCharArray) mw_ArrayOut[0] ;
    mwCell_coeffnames    = (MWCellArray) mw_ArrayOut[1] ;
    mwDouble_coeffvalues = (MWNumericArray) mw_ArrayOut[2] ;
    mwDouble_confint     = (MWNumericArray) mw_ArrayOut[3] ;
    mwStructure_info     = (MWStructArray)  mw_ArrayOut[4] ;

    // This output works as well
    /*
    Console.WriteLine(mwChar_formula      ) ;
    Console.WriteLine(mwCell_coeffnames   ) ;
    Console.WriteLine(mwDouble_coeffvalues) ;
    Console.WriteLine(mwDouble_confint    ) ;
    Console.WriteLine(mwStructure_info   ) ;
    */

    String str_formula = mwChar_formula.ToString() ;
    Console.WriteLine("Curve fitting formula :") ;
    Console.WriteLine(str_formula) ;
    Console.WriteLine() ;

    String str_coeffnames = mwCell_coeffnames.ToString() ;
    Console.WriteLine("Coefficient names:") ;
    Console.WriteLine(str_coeffnames) ;
    Console.WriteLine() ;
```

```
    double[] db_coeffvalues = (double[])mwDouble_coeffvalues.ToVector(
                                                    MWArrayComponent.Real) ;
    Console.WriteLine("Coefficient values:") ;
    PrintVector(db_coeffvalues);
    Console.WriteLine() ;

    double[,] confidentInterval = (double[,])mwDouble_confint.ToArray(
                                                    MWArrayComponent.Real) ;
    Console.WriteLine("Confidence interval:");
    PrintMatrix(confidentInterval) ;
    Console.WriteLine() ;

    Console.WriteLine("Error info:") ;
    String str_info = mwStructure_info.ToString() ;
    Console.WriteLine(str_info) ;

    // Free memory
    mw_X.Dispose() ;
    mw_Y.Dispose() ;

    MWNumericArray.DisposeArray(mw_ArrayOut) ;

    mwChar_formula.Dispose() ;
    mwCell_coeffnames.Dispose() ;
    mwDouble_coeffvalues.Dispose() ;
    mwDouble_confint.Dispose() ;

    mwStructure_info.Dispose() ;
}
```

end code

9.2 Plot Graphics from Curve Fitting Toolbox in C#

Problem 2

input

. There are two data arrays:

```
X = {0, 50, 100, 150, 200, 250, 300, 350, 400, 450, 500, 550, 600} ;
Y = {0.4000, 0.2426, 0.1472, 0.0893, 0.0541, 0.0328, 0.0199,
                 0.0121, 0.0073, 0.0044, 0.0027, 0.0016,  0.0010 } ;
```

output :

. Find a Curve Fitting polynomial function and its information, with the formula:

$$p1 * x^3 + p2 * x^2 + p3 * x + p4$$

. Plot curves of (X,Y) and curve fitting data

The code of the subroutine, *CurveFittingWithPlots()*, at the end of this chapter is used for this problem. The result output is:

```
Curve fitting formula :
p1*x^3 + p2*x^2 + p3*x + p4

Coefficient names:
    'p1'
    'p2'
    'p3'
    'p4'

Coefficient values:
-5.66993006993008E-09
7.0736063936064E-06
-0.00285847252747253
0.383108791208791
```

```
Confidence interval:
-7.23782835968991E-09 5.64027438515616E-06 -0.00321949974326486 0.359112207949607
-4.10203178017025E-09 8.50693840205665E-06 -0.0024974453116802  0.407105374467976

Error info:
           sse: 0.0014
       rsquare: 0.9920
           dfe: 9
    adjrsquare: 0.9893
          rmse: 0.0124
```

The subroutine *CurveFittingWithPlots()* generates two curves. One is a curve of (X,Y) and other is a curve of fitting data shown in the following.

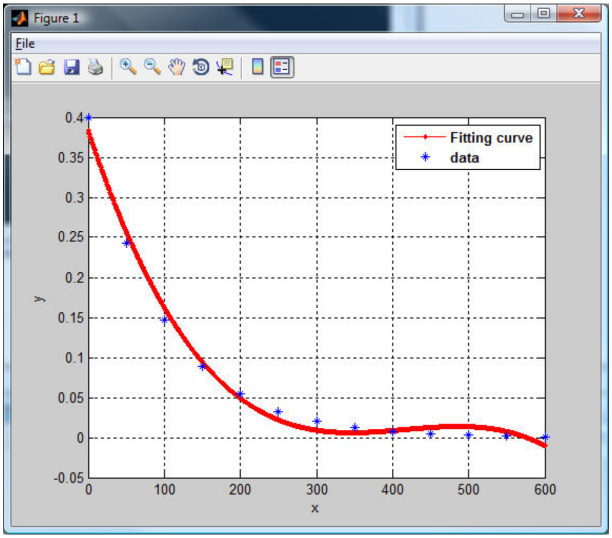


Figure 9.2: Graphics of curve fitting

9.3 Choosing functions in MATLAB Curve Fitting Toolbox in C# functions

In the previous sections, the code we chose were the two functions ***exp2*** for $a * exp(b * x) + c * exp(d * x)$ and ***poly3*** for $p1 * x^3 + p2 * x^2 + p3 * x + p4$. The following are common models of library curves in Curve Fitting Toolbox (these get from command ***cflibhelp*** in MATLAB Commnd Prompt).

1. Polynomial models

   ```
    poly1    Y = p1*x+p2
    poly2    Y = p1*x^2+p2*x+p3
    poly3    Y = p1*x^3+p2*x^2+...+p4
    ...
    poly9    Y = p1*x^9+p2*x^8+...+p10
   ```

2. Exponential models

   ```
   exp1      Y = a*exp(b*x)
   exp2      Y = a*exp(b*x)+c*exp(d*x)
   ```

3. Sinusoidal models

   ```
   sin1     Y = a1*sin(b1*x+c1)
   sin2     Y = a1*sin(b1*x+c1)+a2*sin(b2*x+c2)
   sin3     Y = a1*sin(b1*x+c1)+...+a3*sin(b3*x+c3)
   ...
   sin8     Y = a1*sin(b1*x+c1)+...+a8*sin(b8*x+c8)
   ```

4. Power models

   ```
   power1    Y = a*x^b
   power2    Y = a*x^b+c
   ```

5. Rational models

   ```
   rat02      Y = (p1)/(x^2+q1*x+q2)
   rat21      Y = (p1*x^2+p2*x+p3)/(x+q1)
   rat55      Y = (p1*x^5+...+p6)/(x^5+...+q5)
   ```

6. Gaussian models

```
gauss1     Y = a1*exp(-((x-b1)/c1)^2)
gauss2     Y = a1*exp(-((x-b1)/c1)^2)+a2*exp(-((x-b2)/c2)^2)
gauss3     Y = a1*exp(-((x-b1)/c1)^2)+...+a3*exp(-((x-b3)/c3)^2)
...
gauss8     Y = a1*exp(-((x-b1)/c1)^2)+...+a8*exp(-((x-b8)/c8)^2)
```

7. Fourier models

```
fourier1   Y = a0+a1*cos(x*p)+b1*sin(x*p)
fourier2   Y = a0+a1*cos(x*p)+b1*sin(x*p)+a2*cos(2*x*p)+b2*sin(2*x*p)
fourier3   Y = a0+a1*cos(x*p)+b1*sin(x*p)+...+a3*cos(3*x*p)+b3*sin(3*x*p)
 ...
fourier8   Y = a0+a1*cos(x*p)+b1*sin(x*p)+...+a8*cos(8*x*p)+b8*sin(8*x*p)

where p = 2*pi/(max(xdata)-min(xdata)).
```

9.4 Advance of Setting Plot Graphics from Curve Fitting Toolbox in C# functions

In the code of this chapter, there is a subroutine *CurveFittingPlotsAdvance()* that allows users to set curve properties of *title*, *xlabel*, *ylabel*, curve colors, and legends. Based on this setting, users can write similar functions M-files to create their own way of using graphics in Curve Fitting Toolbox. To know more information of properties of MATLAB , see the MATLAB function *Plot*.

These are some syntax characters for colors of MATLAB graphics:

`b`	`blue`
`g`	`green`
`r`	`red`
`c`	`cyan`
`m`	`magenta`
`y`	`yellow`
`k`	`black`
`w`	`white`

The following is the full code of this chapter.

Listing code

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using MathWorks.MATLAB.NET.Arrays;
using MathWorks.MATLAB.NET.Utility;
using CurveFittingsNameSpace ;

namespace Example
{
  class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine(" Matlab C# for Curve Fititng Toolbox. ") ;
      Program objProgram = new Program() ;

      //objProgram.CurveFittingInfo() ;
      //objProgram.CurveFittingPlots() ;
      objProgram.CurveFittingPlotsAdvance() ;
    }

    /* **************************** */
    public void CurveFittingInfo()
    {
      double[]X = {0, 50, 100, 150, 200, 250, 300, 350, 400, 450, 500, 550, 600} ;

      double[]Y = {0.4000, 0.2426, 0.1472, 0.0893, 0.0541, 0.0328, 0.0199,
            0.0121, 0.0073, 0.0044, 0.0027, 0.0016,  0.0010 } ;

      MWNumericArray mw_X = new MWNumericArray(X) ;
      MWNumericArray mw_Y = new MWNumericArray(Y) ;
```

```
MWArray [] mw_ArrayOut              = null ;

MWCharArray mwChar_formula          = null ;
MWCellArray mwCell_coeffnames       = null ;
MWNumericArray mwDouble_coeffvalues = null ;
MWNumericArray mwDouble_confint     = null ;

MWStructArray mwStructure_info      = null ;

String curveFitLib = "exp2" ;

CurveFittings objMatlab = new CurveFittings() ;
mw_X = (MWNumericArray)objMatlab.mytranspose(mw_X) ;
mw_Y = (MWNumericArray)objMatlab.mytranspose(mw_Y) ;

mw_ArrayOut = objMatlab.CurveFitting(5,mw_X, mw_Y, curveFitLib) ;

mwChar_formula       = (MWCharArray) mw_ArrayOut[0] ;
mwCell_coeffnames    = (MWCellArray) mw_ArrayOut[1] ;
mwDouble_coeffvalues = (MWNumericArray) mw_ArrayOut[2] ;
mwDouble_confint     = (MWNumericArray) mw_ArrayOut[3] ;
mwStructure_info     = (MWStructArray)  mw_ArrayOut[4] ;

// This output works as well
/*
Console.WriteLine(mwChar_formula    ) ;
Console.WriteLine(mwCell_coeffnames   ) ;
Console.WriteLine(mwDouble_coeffvalues) ;
Console.WriteLine(mwDouble_confint  ) ;
Console.WriteLine(mwStructure_info   ) ;
*/

String str_formula = mwChar_formula.ToString() ;
Console.WriteLine("Curve fitting formula :") ;
Console.WriteLine(str_formula) ;
```

```
        Console.WriteLine() ;

        String str_coeffnames = mwCell_coeffnames.ToString() ;
        Console.WriteLine("Coefficient names:") ;
        Console.WriteLine(str_coeffnames) ;
        Console.WriteLine() ;

        double[] db_coeffvalues = (double[])mwDouble_coeffvalues.ToVector(MWArrayComponent.Real) ;
        Console.WriteLine("Coefficient values:") ;
        PrintVector(db_coeffvalues);
        Console.WriteLine() ;

        double[,] confidentInterval = (double[,])mwDouble_confint.ToArray(MWArrayComponent.Real) ;
        Console.WriteLine("Confidence interval:");
        PrintMatrix(confidentInterval) ;
        Console.WriteLine() ;

        Console.WriteLine("Error info:") ;
        String str_info = mwStructure_info.ToString() ;
        Console.WriteLine(str_info) ;

        // Free memory
        mw_X.Dispose() ;
        mw_Y.Dispose() ;

        MWNumericArray.DisposeArray(mw_ArrayOut) ;

        mwChar_formula.Dispose() ;
        mwCell_coeffnames.Dispose() ;
        mwDouble_coeffvalues.Dispose() ;
        mwDouble_confint.Dispose() ;

        mwStructure_info.Dispose() ;
      }

    /* **************************** */
```

```
public void CurveFittingPlots()
{
  double[]X = {0, 50, 100, 150, 200, 250, 300, 350, 400, 450, 500, 550, 600} ;

  double[]Y = {0.4000, 0.2426, 0.1472, 0.0893, 0.0541, 0.0328, 0.0199,
        0.0121, 0.0073, 0.0044, 0.0027, 0.0016,  0.0010 } ;

  MWNumericArray mw_X = new MWNumericArray(X) ;
  MWNumericArray mw_Y = new MWNumericArray(Y) ;

  MWArray [] mw_ArrayOut              = null ;

  MWCharArray mwChar_formula          = null ;
  MWCellArray mwCell_coeffnames       = null ;
  MWNumericArray mwDouble_coeffvalues = null ;
  MWNumericArray mwDouble_confint     = null ;

  MWStructArray mwStructure_info      = null ;

  String curveFitLib = "poly3" ;

  CurveFittings objMatlab = new CurveFittings() ;
  mw_X = (MWNumericArray)objMatlab.mytranspose(mw_X) ;
  mw_Y = (MWNumericArray)objMatlab.mytranspose(mw_Y) ;

  mw_ArrayOut = objMatlab.CurveFittingWithPlots(5,mw_X, mw_Y, curveFitLib) ;

  mwChar_formula       = (MWCharArray) mw_ArrayOut[0] ;
  mwCell_coeffnames    = (MWCellArray) mw_ArrayOut[1] ;
  mwDouble_coeffvalues = (MWNumericArray) mw_ArrayOut[2] ;
  mwDouble_confint     = (MWNumericArray) mw_ArrayOut[3] ;
  mwStructure_info     = (MWStructArray)  mw_ArrayOut[4] ;

  // This output works as well
  /*
  Console.WriteLine(mwChar_formula    ) ;
  Console.WriteLine(mwCell_coeffnames   ) ;
  Console.WriteLine(mwDouble_coeffvalues) ;
  Console.WriteLine(mwDouble_confint  ) ;
  Console.WriteLine(mwStructure_info   ) ;
  */

  String str_formula = mwChar_formula.ToString() ;
  Console.WriteLine("Curve fitting formula :") ;
  Console.WriteLine(str_formula) ;
  Console.WriteLine() ;

  String str_coeffnames = mwCell_coeffnames.ToString() ;
  Console.WriteLine("Coefficient names:") ;
  Console.WriteLine(str_coeffnames) ;
  Console.WriteLine() ;

  double[] db_coeffvalues = (double[])mwDouble_coeffvalues.ToVector(MWArrayComponent.Real) ;
  Console.WriteLine("Coefficient values:") ;
  PrintVector(db_coeffvalues);
  Console.WriteLine() ;

  double[,] confidentInterval = (double[,])mwDouble_confint.ToArray(MWArrayComponent.Real) ;
  Console.WriteLine("Confidence interval:");
  PrintMatrix(confidentInterval) ;
  Console.WriteLine() ;

  Console.WriteLine("Error info:") ;
  String str_info = mwStructure_info.ToString() ;
  Console.WriteLine(str_info) ;

  objMatlab.WaitForFiguresToDie() ;

  // Free memory
  mw_X.Dispose() ;
  mw_Y.Dispose() ;
```

```
  MWNumericArray.DisposeArray(mw_ArrayOut) ;

  mwChar_formula.Dispose() ;
  mwCell_coeffnames.Dispose() ;
  mwDouble_coeffvalues.Dispose() ;
  mwDouble_confint.Dispose() ;

  mwStructure_info.Dispose() ;
}

/* **************************** */
public void CurveFittingPlotsAdvance()
{
  double[]X = {0, 50, 100, 150, 200, 250, 300, 350, 400, 450, 500, 550, 600} ;

  double[]Y = {0.4000, 0.2426, 0.1472, 0.0893, 0.0541, 0.0328, 0.0199,
        0.0121, 0.0073, 0.0044, 0.0027, 0.0016,  0.0010 } ;

  MWNumericArray mw_X = new MWNumericArray(X) ;
  MWNumericArray mw_Y = new MWNumericArray(Y) ;

  MWArray [] mw_ArrayOut             = null ;

  MWCharArray mwChar_formula          = null ;
  MWCellArray mwCell_coeffnames       = null ;
  MWNumericArray mwDouble_coeffvalues = null ;
  MWNumericArray mwDouble_confint     = null ;

  MWStructArray mwStructure_info      = null ;

  String curveFitLib = "poly3" ;

  CurveFittings objMatlab = new CurveFittings() ;
  mw_X = (MWNumericArray)objMatlab.mytranspose(mw_X) ;
  mw_Y = (MWNumericArray)objMatlab.mytranspose(mw_Y) ;
```

```
String graphicTitle = @"Curve Fitting Graphics" ;
String xlabel = "Force" ;
String ylabel = "Voltage" ;

String curveFittingColor = "r" ;
String curveDataColor  = "g";

String curveFittingLegend = "My curve fitting" ;
String dataLegend     = "My data" ;

mw_ArrayOut = objMatlab.CurveFittingWithPlotsAdvance(5,mw_X, mw_Y, curveFitLib,
        graphicTitle,xlabel,ylabel,
        curveFittingColor,curveDataColor,curveFittingLegend,dataLegend) ;

mwChar_formula       = (MWCharArray) mw_ArrayOut[0] ;
mwCell_coeffnames    = (MWCellArray) mw_ArrayOut[1] ;
mwDouble_coeffvalues = (MWNumericArray) mw_ArrayOut[2] ;
mwDouble_confint     = (MWNumericArray) mw_ArrayOut[3] ;
mwStructure_info     = (MWStructArray)  mw_ArrayOut[4] ;

// This output works as well
/*
Console.WriteLine(mwChar_formula    ) ;
Console.WriteLine(mwCell_coeffnames   ) ;
Console.WriteLine(mwDouble_coeffvalues) ;
Console.WriteLine(mwDouble_confint  ) ;
Console.WriteLine(mwStructure_info   ) ;
*/

String str_formula = mwChar_formula.ToString() ;
Console.WriteLine("Curve fitting formula :") ;
Console.WriteLine(str_formula) ;
Console.WriteLine() ;

String str_coeffnames = mwCell_coeffnames.ToString() ;
Console.WriteLine("Coefficient names:") ;
```

```
  Console.WriteLine(str_coeffnames) ;
  Console.WriteLine() ;

  double[] db_coeffvalues = (double[])mwDouble_coeffvalues.ToVector(
                                              MWArrayComponent.Real) ;
  Console.WriteLine("Coefficient values:") ;
  PrintVector(db_coeffvalues);
  Console.WriteLine() ;

  double[,] confidentInterval = (double[,])mwDouble_confint.ToArray(
                                              MWArrayComponent.Real) ;
  Console.WriteLine("Confidence interval:");
  PrintMatrix(confidentInterval) ;
  Console.WriteLine() ;

  Console.WriteLine("Error info:") ;
  String str_info = mwStructure_info.ToString() ;
  Console.WriteLine(str_info) ;
  objMatlab.WaitForFiguresToDie() ;

  // Free memory
  mw_X.Dispose() ;
  mw_Y.Dispose() ;

  MWNumericArray.DisposeArray(mw_ArrayOut) ;

  mwChar_formula.Dispose() ;
  mwCell_coeffnames.Dispose() ;
  mwDouble_coeffvalues.Dispose() ;
  mwDouble_confint.Dispose() ;

  mwStructure_info.Dispose() ;
}

/* ****************************** */
public static void PrintVector(double[] vector)
```

```
    {
      int i, row;
      row = vector.GetUpperBound(0) - vector.GetLowerBound(0) + 1;

      for (i = 0; i < row; i++)
      {
        Console.Write("{0} \t", vector.GetValue(i).ToString());
        Console.WriteLine();
      }
    }

    /* ******************************** */
    public static void PrintMatrix(double[,] matrix)
    {
      int i, j, row, col ;
      row = matrix.GetUpperBound(0) - matrix.GetLowerBound(0) + 1;
      col = matrix.GetUpperBound(1) - matrix.GetLowerBound(1) + 1;

      for (i=0; i<row; i++)
      {
        for (j=0; j<col; j++)
        {
          Console.Write("{0} \t", matrix.GetValue(i,j).ToString() ) ;
        }
        Console.WriteLine() ;
      }
    }
    /* ******************************** */
    /* ******************************** */
    /* ******************************** */
  }
}
```

end code

Chapter 10

Roots of Equations

In this chapter we'll generate a class ***FindingRoots*** from common M-files to find roots of a polynomial function and a nonlinear function. The generated functions of this library will be used in a MSVC# .Net 2008 project to find the roots of functions. The procedure to create the class ***FindingRoots*** is the same as the procedure in Chapter 2.

We will write the M-files as shown below. These functions will be used to generate the class ***FindingRoots***.

`myfzero.m and myroots.m`

myroots.m

```
function r = (c)
r = roots(c) ;
```

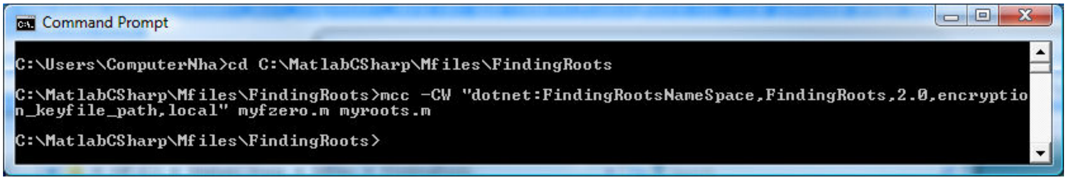
myfzero.m

```
function x = myfzero(strfunc, x0)

F = inline(strfunc) ;
x = fzero(F, x0) ;
```

The procedure to create the class ***FindingRoots*** is the same as the procedure in Chapter 2 as follows:

1. Write the following command in **Command Prompt** to generate the name space ***FindingRootsNameSpace*** that contains the class ***FindingRoots*** (see Fig.10.1).

   ```
   mcc -CW "dotnet:FindingRootsNameSpace,FindingRoots,2.0,encryption_keyfile_path,
   local" myfzero.m myroots.m
   ```



```
C:\Users\ComputerNha>cd C:\MatlabCSharp\Mfiles\FindingRoots

C:\MatlabCSharp\Mfiles\FindingRoots>mcc -CW "dotnet:FindingRootsNameSpace,FindingRoots,2.0,encryptio
n_keyfile_path,local" myfzero.m myroots.m

C:\MatlabCSharp\Mfiles\FindingRoots>
```

Figure 10.1: Command of creating the class ***FindingRoots***

2. Create a regular C# project in Console Application of Microsoft Visual Studio 2008.

3. Click **Build**, then click **Rebuild** to make sure the program works.

4. In the project menu, click **Project**, **Add Reference**. Then the project will pop up a dialog to choose a reference.

5. Click on tab **.Net**, **MathWorks, .NET MWArray API**. Then the project will add MATLAB array wrapper classes for .NET, MWArray into the C# project.

6. Copy the two files `dotnet_mcc_component_data.cs` and `FindingRoots.cs` from the previous steps and put them into the same directory of the `Program.cs` file.

7. Copy the file `FindingRootsNameSpace.ctf` and put it into the same directory of the project executable file, `Example.exe`.

8. Add these two files to the C# project by click **Project** on the project menu, then click **Add Existing Item**. The project will pop up a dialog then choose the two files `dotnet_mcc_component_data.cs` and `FindingRoots.cs`. This step will add these two files to the project.

9. At the top of the file `Program.cs`, add the following:

```
using MathWorks.MATLAB.NET.Arrays;
using MathWorks.MATLAB.NET.Utility;
using FindingRootsNameSpace;
```

The following sections show how to use the functions in the class ***FindingRoots*** to solve the common computation problems. The full code is at the end of this chapter.

10.1 Roots of Polynomials

This section describes how to use the functions in the generated class to find the roots of a polynomial function.

In this section, we will use MATLAB functions to solve the following problems.

Problem 1

Find the root of the polynomial function:

$$f(x) = -x^3 + 7.2x^2 - 21x - 5$$

In calculation, the values of these coefficients will be assigned to an array **c**[ ] in the function (pay attention to its order):

$$f(x) = c_1x^3 + c_2x^2 + c_3x + c_4$$

where:

$$c_1 = -1\ , \quad c_2 = 7.2\ , \quad c_3 = -21\ , \text{ and } \quad c_4 = -5$$

The following is the code to solve Problem 1 by using the function *myroots(..)*. See the MATLAB manual [2] for more information on the MATLAB function *roots(..)*.

Listing code

```
public void FindingRootsPolynormial()
{
    /*
    Find the solutions of polynomial function:
        f(x) = -x^3 + 7.2x^2 -21x -5
    */
    int order = 3 ;
    double[] db_coefs = { -1, 7.2, -21, -5 } ;

    /* declare mxArray variables */
    MWNumericArray mw_coefs = new MWNumericArray(db_coefs) ;
    MWNumericArray mw_x     = null ;

    /* call an implemental function */
    FindingRoots objMatlab = new FindingRoots() ;
    mw_x = (MWNumericArray) objMatlab.myroots(mw_coefs) ;

    Console.WriteLine("Solutions of the polynomial function: \n {0}", mw_x) ;

    /* convert back to C/C++ double  */
    double []db_xReal = new double [order] ;
    double []db_xImag = new double [order] ;

    db_xReal = (double[]) mw_x.ToVector(MWArrayComponent.Real)      ;

    try
    {
        db_xImag = (double[])mw_x.ToVector(MWArrayComponent.Imaginary);
    }
    catch
    {
        // nothing, db_xImag[] will be assigned value of zero
    }

    /* print out */
    Console.WriteLine("Solutions of the polynomial function : ")    ;
    int i ;
    for (i=0; i<order; i++)
    {
        Console.Write( db_xReal[i].ToString() +  " + " ) ;
        Console.Write( db_xImag[i].ToString() + "i \n" ) ;
    }

    /* free memory */
    mw_coefs.Dispose()  ;
    mw_x.Dispose()      ;
}
```

end code

Note
The roots of a polynomial function may have imaginary terms. Therefore we need to use another variable (say db_xImag) to receive the imaginary value of a MWNumericArray variable. Or we can use the property `IsComplex` (for example mw_x.IsComplex) in this case.

10.2 The Root of a Nonlinear-Equation

This section describes how to use the function myfzero(..) in the generated ***FindingRoot*** class to find a root of a nonlinear function. These functions use a MATLAB function fzero(..) which returns ONLY ONE SOLUTION near the initial guess value. For more information of the fzero(..) function, refer to MATLAB manual [5].

Problem 2

Find the root of the function:

$$f(x) = \sin(2x) + \cos(x) + 1$$

The following is the code to solve Problem 2 by using the function myfzero(..).

Listing code

```
public void FindingZeroFunction ( )
{
    /* Find the solution of the function f(x) = sin(2*x) + cos(x) + 1
        fzero(..) returns ONLY ONE SOLUTION near a initial guess value
        If your problem is complicated, please look at functions
        in Optimization Tool Box
    */
    double db_initialGuess = 0.9 ;
    String strfunc = "sin(2*x) + cos(x) + 1" ;

    /* declare mxArray variables */
    MWCharArray mw_strfunc  = new MWCharArray(strfunc)      ;
    MWArray [] mw_ArrayOut = null ;
    MWNumericArray mw_x     = null ;

    /* call an implemental function */
    FindingRoots objMatlab = new FindingRoots() ;

    mw_ArrayOut = objMatlab.myfzero(1, mw_strfunc, db_initialGuess) ;
    Console.WriteLine("Solutions of the given function :\n {0}", mw_ArrayOut ) ;

    mw_x = (MWNumericArray) mw_ArrayOut[0] ;
    Console.WriteLine("Solutions of the given function :\n {0}", mw_x ) ;

    /* convert back to Cs double  */
    double db_xReal = (double) mw_x.ToScalarDouble() ;

    Console.WriteLine("Solutions of the given function : " ) ;
    Console.WriteLine( db_xReal.ToString() ) ;

    /* free memory */
    mw_strfunc.Dispose()    ;
    mw_x.Dispose()          ;
    MWNumericArray.DisposeArray(mw_ArrayOut) ;
}
```

end code

Note

The generated function myfzero(..) is a function-function and has an argument as an expression string. The form of this expression string follows the rule of a MATLAB expression string.

The following is full code for this chapter.

Listing code

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using MathWorks.MATLAB.NET.Arrays;
using MathWorks.MATLAB.NET.Utility;
using FindingRootsNameSpace ;

namespace Example
{
  class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine("Roots of functions.") ;

      Program objProgram = new Program() ;
      objProgram.FindingRootsPolynormial() ;

      Console.WriteLine("\n") ;
      objProgram.FindingZeroFunction() ;
    }

    /* *************************** */
    public void FindingRootsPolynormial()
    {
      /*
```

```
Find the solutions of polynomial function:
    f(x) = -x^3 + 7.2x^2 -21x -5
*/
int order = 3 ;
double[] db_coefs = { -1, 7.2, -21, -5 } ;

/* declare mxArray variables */
MWNumericArray mw_coefs = new MWNumericArray(db_coefs) ;
MWNumericArray mw_x     = null ;

/* call an implemental function */
FindingRoots objMatlab = new FindingRoots() ;
mw_x = (MWNumericArray) objMatlab.myroots(mw_coefs) ;

Console.WriteLine("Solutions of the polynomial function: \n {0}", mw_x) ;

/* convert back to C/C++ double  */
double []db_xReal = new double [order] ;
double []db_xImag = new double [order] ;

db_xReal = (double[]) mw_x.ToVector(MWArrayComponent.Real)      ;

try
{
    db_xImag = (double[])mw_x.ToVector(MWArrayComponent.Imaginary);
}
catch
{
    // nothing, db_xImag[] will be assigned value of zero
}

/* print out */
Console.WriteLine("Solutions of the polynomial function : ")    ;
int i ;
for (i=0; i<order; i++)
{
        Console.Write( db_xReal[i].ToString() +  " + " ) ;
        Console.Write( db_xImag[i].ToString() + "i \n" ) ;
    }

    /* free memory */
    mw_coefs.Dispose()  ;
    mw_x.Dispose()      ;
  }

  /* *************************** */
  public void FindingZeroFunction ( )
  {
    /* Find the solution of the function f(x) = sin(2*x) + cos(x) + 1
        fzero(..) returns ONLY ONE SOLUTION near a initial guess value
        If your problem is complicated, please look at functions
        in Optimization Tool Box
    */
    double db_initialGuess = 0.9 ;
    String strfunc = "sin(2*x) + cos(x) + 1" ;

    /* declare mxArray variables */
    MWCharArray mw_strfunc  = new MWCharArray(strfunc)      ;
    MWArray [] mw_ArrayOut = null ;
    MWNumericArray mw_x     = null ;

    /* call an implemental function */
    FindingRoots objMatlab = new FindingRoots() ;

    mw_ArrayOut = objMatlab.myfzero(1, mw_strfunc, db_initialGuess) ;
    Console.WriteLine("Solutions of the given function :\n {0}", mw_ArrayOut ) ;

    mw_x = (MWNumericArray) mw_ArrayOut[0] ;
    Console.WriteLine("Solutions of the given function :\n {0}", mw_x ) ;

    /* convert back to Cs double  */
    double db_xReal = (double) mw_x.ToScalarDouble() ;
```

```
      Console.WriteLine("Solutions of the given function : " ) ;
      Console.WriteLine( db_xReal.ToString() ) ;

      /* free memory */
      mw_strfunc.Dispose()    ;
      mw_x.Dispose()          ;

      MWNumericArray.DisposeArray(mw_ArrayOut) ;
    }

    /* ******************************** */
    /* ******************************** */
    /* ******************************** */
  }
}
```

end code

Chapter 11

Fast Fourier Transform

The Fourier analysis is very useful for data analysis in applications. The Fourier transform divides a function into constituent sinusoids of different frequencies.
The Fourier transform of a function $f(x)$ is defined as:

$$F(s) = \int_{-\infty}^{+\infty} f(x) e^{-i(2\pi s)x} dx \tag{11.1a}$$

and its inverse

$$f(x) = \int_{-\infty}^{+\infty} F(s) e^{\ i(2\pi x)s} ds \tag{11.1b}$$

The Fast Fourier Transform (FFT) is an efficient algorithm for computing the discrete Fourier transform [1]. MATLAB provides functions for working on the Fast Fourier Transform and its inverse. These functions will be used to generate a class in the following sections.

In this chapter we'll generate a class ***FFT*** from common M-files working on FFT problems. The generated functions of this class will be used in a MSVC# .Net 2008 project to solve the FFT problems.

We will write the M-files as shown below. These functions will be used to generate the class ***FFT***.

`myfft.m, myifft.m, myfft2.m, and myifft2.m`

myfft.m

```
function Y = myfft(X)

Y = fft(X) ;
```

myifft.m

```
function Y = myifft(X)

Y = ifft(X) ;
```

myfft2.m

```
function Y = myfft2(X)

Y = fft2(X) ;
```

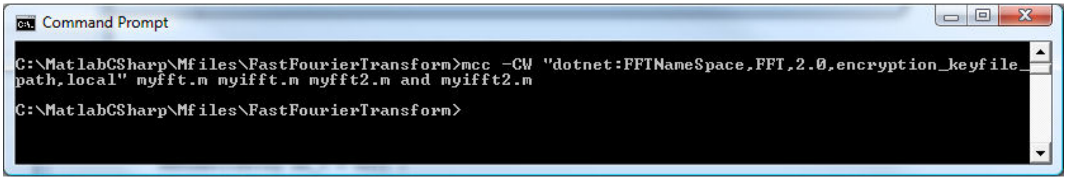
myifft2.m

```
function Y = myifft2(X)

Y = ifft2(X) ;
```

The procedure to create the class ***FFT*** is the same as the procedure in Chapter 2 as follows:

1. Write the following command in **Command Prompt** to generate the name space ***FFTNameSpace*** that contains the class ***FFT*** (see Fig.11.1).

   ```
   mcc -CW "dotnet:FFTNameSpace,FFT,2.0,encryption_keyfile_path,
   local" myfft.m myifft.m myfft2.m and myifft2.m
   ```



```
C:\MatlabCSharp\Mfiles\FastFourierTransform>mcc -CW "dotnet:FFTNameSpace,FFT,2.0,encryption_keyfile_
path,local" myfft.m myifft.m myfft2.m and myifft2.m

C:\MatlabCSharp\Mfiles\FastFourierTransform>
```

Figure 11.1: Command of creating the class ***FFT***

2. Create a regular C# project in Console Application of Microsoft Visual Studio 2008.
3. Click **Build**, then click **Rebuild** to make sure the program works.
4. In the project menu, click **Project**, **Add Reference**. Then the project will pop up a dialog to choose a reference.
5. Click on tab **.Net**, **MathWorks, .NET MWArray API**. Then the project will add MATLAB array wrapper classes for .NET, **MWArray** into the C# project.
6. Copy the two files `dotnet_mcc_component_data.cs` and `FFT.cs` from previous steps and put them into the same directory of the `Program.cs` file.
7. Copy the file `FFTNameSpace.ctf` and put it into the same directory of the project executable file, `Example.exe`.
8. Add these two files to the C# project by clicking **Project** on the project menu, then click **Add Existing Item**. The project will pop up a dialog then choose the two files `dotnet_mcc_component_data.cs` and `FFT.cs`. This step will add these two files to the project.
9. At the top of the file `Program.cs`, add the following:

```
using MathWorks.MATLAB.NET.Arrays;
using MathWorks.MATLAB.NET.Utility;
using FFTNameSpace;
```

The following sections show how to use the functions in the class ***FFT*** to solve the common computation problems. The full code is at the end of this chapter.

11.1 One-Dimensional Fast Fourier Transform

The MATLAB functions implement FFT and its inverse for a vector **X** with length N as follows:

$$\mathbf{Y}_k = \sum_{n=1}^{N} \mathbf{X}_n \; e^{(-i2\pi)\left(\frac{n-1}{N}\right)(k-1)} \qquad 1 \le k \le N \tag{11.2a}$$

and its inverse

$$\mathbf{X}_n = \frac{1}{N} \sum_{k=1}^{N} \mathbf{Y}_k \; e^{(i2\pi)\left(\frac{k-1}{N}\right)(n-1)} \qquad 1 \le n \le N \tag{11.2b}$$

Remarks

1. The vectors **X** and **Y** in equation 11.2 are represented by functions $f(x)$ and $F(s)$ in equation 11.1, respectively.
2. The first index of a vector in MATLAB starts at 1, therefore in the equation 11.2 we have term $(n-1)$ and $(k-1)$. This produces identical results as the traditional Fourier equations from 0 to $(N-1)$.

In this section, we will use MATLAB functions to solve the following problems.

Problem 1

input . Vector **X**,
X = { 6, 3, 7, -9, 0, 3, -2, 1 }

output . Finding the FFT vector **Y** in Eq. 11.2a

The following is the code to solve Problem 1 by using the function myfft(..).

Listing code

```
public void FastFourierTrans1D()
{
    double[] db_X  = { 6, 3, 7, -9, 0, 3, -2, 1 } ;
    int vectorSize = 8 ;

    /* declare mxArray variables */
    MWNumericArray mw_X = new MWNumericArray(db_X) ;
    MWNumericArray mw_Y = null ;

    /* call an implemental function */
    FFT objMatlab = new FFT() ;

    mw_Y = (MWNumericArray)objMatlab.myfft(mw_X) ;
    Console.WriteLine("Fast Fourier Transform of X : \n{0}", mw_Y) ;
    ///* convert back to Cs double  */
    double[] db_YReal = new double [vectorSize] ;
    double[] db_YImag = new double [vectorSize] ;

        db_YReal = (double[]) mw_Y.ToVector(MWArrayComponent.Real) ;

    try
    {
        db_YImag = (double[])mw_Y.ToVector(MWArrayComponent.Imaginary);
    }

    catch
    {
         // nothing, db_YImag[] will be assigned value of zero
    }

    /* print out */
    Console.WriteLine("Fast Fourier Transform of X : \n") ;
    int i ;
    for (i=0; i<vectorSize; i++)
    {
        Console.Write( db_YReal[i].ToString() +  " + " ) ;
        Console.Write( db_YImag[i].ToString() +  "i"  + "\n" ) ;
    }

    Console.WriteLine("\n") ;

    mw_X.Dispose() ;
    mw_Y.Dispose() ;
}
```

end code

Problem 1B

input . A vector **Y**,

Y real numbers = {9.00, 13.0711, 1.0, -1.0711, 13.0, -1.0711 , 1.0 , 13.0711 }

Y imaginary numbers = {0, -1.9289, -14.0, 16.0711, 0, -16.0711, 14.0, 1.9289 }

output . Finding an inverse FFT vector **X** in Eq. 11.2b

The following is the code to solve Problem 1B by using the function myifft(..).

Listing code

```
public void InverseFastFourierTrans1D()
{
    double[] db_YReal = { 9.00, 13.0711,   1.00, -1.0711, 13.00, -1.0711 , 1.00 , 13.0711 } ;
    double[] db_YImag = { 0    , -1.9289, -14.00, 16.0711, 0    , -16.0711, 14.00, 1.9289  } ;
    int vectorSize = 8 ;

    /* declare mxArray variables */
    MWNumericArray mw_Y = new MWNumericArray(db_YReal, db_YImag) ;
    MWNumericArray mw_X = new MWNumericArray(MWArrayComplexity.Complex, MWNumericType.Double, 8) ;

    /* call an implemental function */
    FFT objMatlab = new FFT() ;
    mw_X = (MWNumericArray) objMatlab.myifft(mw_Y) ;
    Console.WriteLine("Inverse Fast Fourier Transform of Y : \n{0}", mw_X ) ;

    /* convert back to Cs double  */
    double[] db_XReal = (double[]) mw_X.ToVector(MWArrayComponent.Real)      ;
    double[] db_XImag = new double [vectorSize] ;

    try
    {
        db_XImag = (double[])mw_X.ToVector(MWArrayComponent.Imaginary);
    }
    catch
    {
        // nothing, db_XImag[] will be assigned value of zero
    }

    /* print out */
    Console.WriteLine("Inverse Fast Fourier Transform of Y : " ) ;
    int i ;
    for (i=0; i<vectorSize; i++)
    {
        Console.Write( db_XReal[i].ToString() +  " + " ) ;
        Console.Write( db_XImag[i].ToString() +  "i" + "\n" ) ;
    }

    Console.WriteLine("\n") ;
    mw_X.Dispose() ;
    mw_Y.Dispose() ;
}
```

end code

11.2 Two-Dimensional Fast Fourier Transform

The MATLAB function fft2(..) ($\mathbf{Y} = fft2(\mathbf{X})$) computes the one-dimensional FFT of each column of a matrix **X**, and the size of the result matrix **Y** is the same size of **X**. If you want to get a different size , use the function $\mathbf{Y} = fft2(\mathbf{X}, m, n)$, refer to the MATLAB manual [5].

In this section, we will use MATLAB functions to solve the following problems.

Problem 2

input . Matrix **X**,

$$\mathbf{X} = \begin{bmatrix} 4 & 3.2 & 6.8 & 9.1 \\ -4 & 1.2 & 4.3 & 5.4 \\ 2.2 & -6.7 & 8 & 12 \end{bmatrix}$$

output . Finding the matrix **Y**, which is a FFT matrix of **X**

The following is the code to solve Problem 2 by using the function myfft2(..) in the generated class ***FFT***.

Listing code

```
public void FastFourierTrans2D()
{
    double[,] X = { {4  ,  3.2,  6.8,  9.1  }  ,
                    {-4 ,  1.2,  4.3,  5.4  }  ,
                    {2.2, -6.7,  8  ,  12.2 }  } ;
    int row = 3 ;
    int col = 4 ;
    int i, j ;

    /* declare mxArray variables */
    MWNumericArray mw_X = new MWNumericArray (X) ;
    MWNumericArray mw_Y = null ;

    /* call an implemental function */
    FFT objMatlab = new FFT() ;
    mw_Y = (MWNumericArray) objMatlab.myfft2(mw_X) ;
    Console.WriteLine("2-D Fast Fourier Transform of X : \n{0}", mw_Y) ;

    /* convert back to Cs double  */
    double [,] db_YReal = new double [row, col] ;
    double [,] db_YImag = new double [row, col] ;

        db_YReal = (double[,]) mw_Y.ToArray(MWArrayComponent.Real)      ;

    try
    {
        db_YImag = (double[,])mw_Y.ToArray(MWArrayComponent.Imaginary);
    }
    catch
    {
        // nothing, db_YImag will be assigned value of zero
    }
    /* print out */
    Console.WriteLine("2-D Fast Fourier Transform of X : ") ;

    for (i=0; i<row; i++)
    {
        for (j=0; j<col; j++ )
        {
            Console.Write( db_YReal.GetValue(i,j).ToString() + " + " );
            Console.Write( db_YImag.GetValue(i,j).ToString() + "i" + "\t\t" ) ;
        }
        Console.WriteLine("\n") ;
    }
    Console.WriteLine("\n") ;

    /* free memory */
    mw_X.Dispose() ;
    mw_Y.Dispose() ;
}
```

end code

<u>**Problem 2B**</u>

input . Matrix $\mathbf{Y}$,

$$\mathbf{Y} = \begin{bmatrix} (45.70 + 0i) & (-16.9000 + 29.0000i) & (-3.10 + 0i) & (-16.9000 - 29.0000i) \\ (11.80 + 7.621i) & (-8.4806 - 3.4849i) & (-0.70 + 9.5263i) & (16.9806 + 7.8151i) \\ (11.80 - 7.621i) & (16.9806 - 7.8151i) & (-0.70 - 9.5263i) & (-8.4806 + 3.4849i) \end{bmatrix}$$

output . Finding the matrix $\mathbf{X}$ which is an inverse FFT matrix of $\mathbf{Y}$.

The following is the code to solve Problem 2B by using the function myifft2(..).

Listing code

```
public void InverseFastFourierTrans2D()
{
    double[,] YReal = { { 45.7000,  -16.9000,  -3.1000,  -16.9000 }    ,
                        { 11.8000,   -8.4806,  -0.7000,   16.9806 }    ,
                        { 11.8000,   16.9806,  -0.7000,  -8.4806  }    };

    double[,] YImag =  {{ 0      ,  29.0000,   0      , -29.0000 }    ,
                        { 7.6210,  -3.4849,   9.5263,   7.8151 }    ,
                        {-7.6210,  -7.8151,  -9.5263,   3.4849 }    };
    int row = 3 ;
    int col = 4 ;
    int i, j ;

    /* declare mxArray variables */
    MWNumericArray mw_X = null ;
    MWNumericArray mw_Y = new MWNumericArray(YReal, YImag) ;

    /* call an implemental function */
    FFT objMatlab = new FFT() ;
    mw_X = (MWNumericArray) objMatlab.myifft2(mw_Y) ;
    Console.WriteLine("Inverse 2-D Fast Fourier Transform of Y : \n{0}", mw_X ) ;

    /* convert back to Cs double  */
    double[,] db_XReal = new double [row, col] ;
    double[,] db_XImag = new double [row, col] ;

        db_XReal = (double[,]) mw_X.ToArray(MWArrayComponent.Real)      ;

    try
    {
        db_XImag = (double[,])mw_X.ToArray(MWArrayComponent.Imaginary);
    }
    catch
    {
        // nothing, db_XImag will be assigned value of zero
    }

    /* print out */
    Console.WriteLine("Inverse 2-D Fast Fourier Transform of Y : " ) ;

    for (i=0; i<row; i++)
    {
        for (j=0; j<col; j++ )
        {
            Console.Write( db_XReal.GetValue(i,j).ToString() +  " + " ) ;
            Console.Write( db_XImag.GetValue(i,j).ToString() +  "i" + "\t\t" );
        }
        Console.WriteLine("\n") ;
    }
    Console.WriteLine("\n") ;

    /* free memory */
    mw_X.Dispose() ;
    mw_Y.Dispose() ;
}
```

end code

The following is full code for this chapter.

Listing code

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using MathWorks.MATLAB.NET.Arrays;
using MathWorks.MATLAB.NET.Utility;
using FFTNameSpace;

namespace Example
{
```

```
class Program
{
  static void Main(string[] args)
  {
    Program objProgram = new Program() ;
    Console.WriteLine("Fast Fourier Transform.") ;

    Console.WriteLine("Fast Fourier Transform in 1D.") ;
    objProgram.FastFourierTrans1D() ;

    Console.WriteLine("Fast Fourier Transform in inverse of 1D.") ;
    objProgram.InverseFastFourierTrans1D() ;

    Console.WriteLine("Fast Fourier Transform in 2D.") ;
    objProgram.FastFourierTrans2D() ;

    Console.WriteLine("Fast Fourier Transform in inverse of 2D.") ;
    objProgram.InverseFastFourierTrans2D() ;
  }

  /* **************************** */
  public void FastFourierTrans1D()
  {
    double[] db_X  = { 6, 3, 7, -9, 0, 3, -2, 1 } ;
    int vectorSize = 8 ;

    /* declare mxArray variables */
    MWNumericArray mw_X = new MWNumericArray(db_X) ;
    MWNumericArray mw_Y = null ;

    /* call an implemental function */
    FFT objMatlab = new FFT() ;

    mw_Y = (MWNumericArray)objMatlab.myfft(mw_X) ;
    Console.WriteLine("Fast Fourier Transform of X : \n{0}", mw_Y) ;
    ///* convert back to Cs double  */
    double[] db_YReal = new double [vectorSize] ;
    double[] db_YImag = new double [vectorSize] ;

      db_YReal = (double[]) mw_Y.ToVector(MWArrayComponent.Real) ;

    try
    {
      db_YImag = (double[])mw_Y.ToVector(MWArrayComponent.Imaginary);
    }

    catch
    {
       // nothing, db_YImag[] will be assigned value of zero
    }

    /* print out */
    Console.WriteLine("Fast Fourier Transform of X : \n") ;
    int i ;
    for (i=0; i<vectorSize; i++)
    {
      Console.Write( db_YReal[i].ToString() +  " + " ) ;
      Console.Write( db_YImag[i].ToString() +  "i"  + "\n" ) ;
    }

    Console.WriteLine("\n") ;

    mw_X.Dispose() ;
    mw_Y.Dispose() ;
  }

  /* **************************** */
  public void InverseFastFourierTrans1D()
  {
    double[] db_YReal={ 9.00, 13.0711,   1.00, -1.0711, 13.00, -1.0711 , 1.00 , 13.0711};
    double[] db_YImag={ 0   , -1.9289, -14.00, 16.0711, 0    , -16.0711, 14.00, 1.9289 };
```

```
int vectorSize = 8 ;

/* declare mxArray variables */
MWNumericArray mw_Y = new MWNumericArray(db_YReal, db_YImag) ;
MWNumericArray mw_X = new MWNumericArray(MWArrayComplexity.Complex,
                                                    MWNumericType.Double, 8) ;

/* call an implemental function */
FFT objMatlab = new FFT() ;
mw_X = (MWNumericArray) objMatlab.myifft(mw_Y) ;
Console.WriteLine("Inverse Fast Fourier Transform of Y : \n{0}", mw_X ) ;

/* convert back to Cs double  */
double[] db_XReal = (double[]) mw_X.ToVector(MWArrayComponent.Real)    ;
double[] db_XImag = new double [vectorSize] ;

try
{
  db_XImag = (double[])mw_X.ToVector(MWArrayComponent.Imaginary);
}
catch
{
  // nothing, db_XImag[] will be assigned value of zero
}

/* print out */
Console.WriteLine("Inverse Fast Fourier Transform of Y : " ) ;
int i ;
for (i=0; i<vectorSize; i++)
{
  Console.Write( db_XReal[i].ToString() +  " + " ) ;
  Console.Write( db_XImag[i].ToString() +  "i" + "\n" ) ;
}

Console.WriteLine("\n") ;
mw_X.Dispose() ;
```

```
  mw_Y.Dispose() ;
}

/* **************************** */
public void FastFourierTrans2D()
{
  double[,] X = { {4  ,  3.2,  6.8,  9.1  }  ,
                  {-4 ,  1.2,  4.3,  5.4  }  ,
                  {2.2, -6.7,  8  ,  12.2 }  } ;
  int row = 3 ;
  int col = 4 ;
  int i, j ;

  /* declare mxArray variables */
  MWNumericArray mw_X = new MWNumericArray (X) ;
  MWNumericArray mw_Y = null ;

  /* call an implemental function */
  FFT objMatlab = new FFT() ;
  mw_Y = (MWNumericArray) objMatlab.myfft2(mw_X) ;
  Console.WriteLine("2-D Fast Fourier Transform of X : \n{0}", mw_Y) ;

  /* convert back to Cs double  */
  double [,] db_YReal = new double [row, col] ;
  double [,] db_YImag = new double [row, col] ;

    db_YReal = (double[,]) mw_Y.ToArray(MWArrayComponent.Real)    ;

  try
  {
    db_YImag = (double[,])mw_Y.ToArray(MWArrayComponent.Imaginary);
  }
  catch
  {
    // nothing, db_YImag will be assigned value of zero
  }
```

```
    /* print out */
    Console.WriteLine("2-D Fast Fourier Transform of X : ") ;

    for (i=0; i<row; i++)
    {
      for (j=0; j<col; j++ )
      {
        Console.Write( db_YReal.GetValue(i,j).ToString() + " + " );
        Console.Write( db_YImag.GetValue(i,j).ToString() + "i" + "\t\t" ) ;
      }
      Console.WriteLine("\n") ;
    }
    Console.WriteLine("\n") ;

    /* free memory */
    mw_X.Dispose() ;
    mw_Y.Dispose() ;
  }

  /* **************************** */
  public void InverseFastFourierTrans2D()
  {
    double[,] YReal = { { 45.7000,  -16.9000,  -3.1000,  -16.9000 }  ,
                        { 11.8000,   -8.4806,  -0.7000,   16.9806 }  ,
                        { 11.8000,   16.9806,  -0.7000,  -8.4806  }  };

    double[,] YImag =  {{ 0     ,  29.0000,   0     , -29.0000 }  ,
                        { 7.6210,  -3.4849,   9.5263,   7.8151 }  ,
                        {-7.6210,  -7.8151,  -9.5263,   3.4849 }  };
    int row = 3 ;
    int col = 4 ;
    int i, j ;

    /* declare mxArray variables */
    MWNumericArray mw_X = null ;
```

```
MWNumericArray mw_Y = new MWNumericArray(YReal, YImag) ;

/* call an implemental function */
FFT objMatlab = new FFT() ;
mw_X = (MWNumericArray) objMatlab.myifft2(mw_Y) ;
Console.WriteLine("Inverse 2-D Fast Fourier Transform of Y : \n{0}", mw_X ) ;

/* convert back to Cs double  */
double[,] db_XReal = new double [row, col] ;
double[,] db_XImag = new double [row, col] ;

  db_XReal = (double[,]) mw_X.ToArray(MWArrayComponent.Real)    ;

try
{
  db_XImag = (double[,])mw_X.ToArray(MWArrayComponent.Imaginary);
}
catch
{
  // nothing, db_XImag will be assigned value of zero
}

/* print out */
Console.WriteLine("Inverse 2-D Fast Fourier Transform of Y : " ) ;

for (i=0; i<row; i++)
{
  for (j=0; j<col; j++ )
  {
    Console.Write( db_XReal.GetValue(i,j).ToString() +  " + " ) ;
    Console.Write( db_XImag.GetValue(i,j).ToString() +  "i" + "\t\t" );
  }
  Console.WriteLine("\n") ;
}
Console.WriteLine("\n") ;
```

```
      /* free memory */
      mw_X.Dispose() ;
      mw_Y.Dispose() ;
    }

/* **************************** */
/* **************************** */
/* **************************** */
  }
}
```

end code

Chapter 12

Eigenvalues and Eigenvectors

In this chapter we'll generate a class ***Eigen*** from common M-files working on problems of eigenvectors and eigenvalues. The generated functions of this class will be used in a MSVC# .Net 2008 project to solve the eigen problems. The procedure to create the class ***Eigen*** is the same as the procedure in Chapter 2.

This chapter focus on using the MATLAB function eig(..) to find the eigenvalues and eigenvectors of a square matrix. For more information of this function, refer to the MATLAB manual [4].

We will write the M-file myeig.m as shown below. This function will be used to generate the class ***Eigen***.

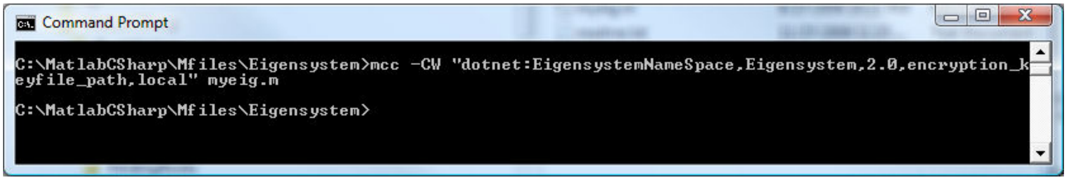
myeig.m

```
function [V,D] = myeig(A)

[V,D] = eig(A) ;
```

The procedure to create the class ***Eigensystem*** is the same as the procedure in Chapter 2 as follows:

1. The command in `Command Prompt` to generate the name space ***EigensystemNameSpace*** that contains the class ***Eigensystem*** is (see Fig.12.1):

```
mcc -CW "dotnet:EigensystemNameSpace,Eigensystem,2.0,encryption_keyfile_path,
local" myeig.m
```



```
C:\MatlabCSharp\Mfiles\Eigensystem>mcc -CW "dotnet:EigensystemNameSpace,Eigensystem,2.0,encryption_k
eyfile_path,local" myeig.m

C:\MatlabCSharp\Mfiles\Eigensystem>
```

Figure 12.1: Command of creating the class ***Eigensystem***

2. Create a regular C# project in Console Application of Microsoft Visual Studio 2008.
3. Click `Build` then click `Rebuild` to make sure the program works.
4. In the project menu, click *Project*, *Add Reference*, the project will pop up a dialog to choose a reference.
5. Click on tab `.Net`, `MathWorks, .NET MWArray API` then the project will add MATLAB array wrapper classes for .NET, *MWArray* into the C# project.
6. Copy two files `dotnet_mcc_component_data.cs` and `Eigensystem.cs` from previous steps and put into the same directory of the `Program.cs` file.
7. Copy the file `EigensystemNameSpace.ctf` and put into the same directory of the project executable file, `Example.exe`.
8. Add these two files to the C# project by click `Project` on the project menu, then click `Add Existing Item`. The project will pop up a dialog then choose the two files `dotnet_mcc_component_data.cs` and `Eigensystem.cs`. This step will add these two files to the project.
9. In the file `Program.cs`, at the top add *using* system

```
using MathWorks.MATLAB.NET.Arrays;
using MathWorks.MATLAB.NET.Utility;
using EigensystemNameSpace;
```

The following sections show how to use the functions in the class ***Eigensystem*** to solve the common computation problems. The full of code is at the end of this chapter.

12.1 Eigenvalues and Eigenvectors

In this section, we will use MATLAB functions to solve the following problems.

Problem 1

input A square matrix **A**

$$\mathbf{A} = \begin{bmatrix} 0 & -6 & -1 \\ 6 & 2 & -16 \\ -5 & 20 & -10 \end{bmatrix}$$

output Finding eigenvalues and eigenvectors of **A**

The following is the code to solve Problem 1 by using function myeig(..) in the generated ***Eigen*** class to find eigenvalues and eigenvectors of a square matrix.

Listing code

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using MathWorks.MATLAB.NET.Arrays;
using MathWorks.MATLAB.NET.Utility;
using EigensystemNameSpace ;

namespace Example
{
  class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine(" ") ;
      Program objProgram = new Program() ;
      Console.WriteLine("Eigenvalues and eigenvectors" ) ;
      objProgram.EigValueVector() ;
    }

    /* **************************** */
    public void EigValueVector()
    {
      double[,] A = {{ 0, -6, -1} , {6, 2, -16} ,  {-5, 20, -10} } ;
      int row = 3 ;
      int col = 3 ;
      int i, j ;

      // declare mx_Array variables //
      MWNumericArray mw_A = new MWNumericArray(A) ;
      MWArray [] mw_ArrayOut = null ;

      MWNumericArray mw_eigenvectors = null ;
      MWNumericArray mw_eigenvalues  = null ;

      /* call an implemental function */
      Eigensystem obj = new Eigensystem() ;
      mw_ArrayOut = obj.myeig(2, mw_A) ;
      Console.WriteLine("Eigen of the matrix A : \n {0}", mw_ArrayOut) ;

      mw_eigenvectors = (MWNumericArray) mw_ArrayOut[0] ;
      mw_eigenvalues  = (MWNumericArray) mw_ArrayOut[1] ;

      /* convert back to Cs double  */
      double[,] db_eigenvectorsReal = new double [row, col] ;
      double[,] db_eigenvectorsImag = new double [row, col] ;

      double[,] db_eigenvaluesReal = new double [row, col] ;
      double[,] db_eigenvaluesImag = new double [row, col] ;

      db_eigenvectorsReal = (double[,]) mw_eigenvectors.ToArray(MWArrayComponent.Real) ;
```

```
db_eigenvaluesReal  = (double[,]) mw_eigenvalues.ToArray(MWArrayComponent.Real)  ;

try
{
  db_eigenvectorsImag = (double[,])mw_eigenvectors.ToArray(MWArrayComponent.Imaginary);
  db_eigenvaluesImag  = (double[,])mw_eigenvalues.ToArray (MWArrayComponent.Imaginary);
}

catch
{
  // nothing, db_eigenvectorsImag and db_eigenvaluesImag will be assigned value of zero
}

/* print out */
Console.WriteLine("Eigenvalues of the matrix A : " ) ;

for (i=0; i<row; i++)
{
  Console.Write( db_eigenvaluesReal.GetValue(i,i) +  " + "  ) ;
  Console.Write( db_eigenvaluesImag.GetValue(i,i) +  "i"  + "\n"  ) ;
}

Console.Write("\n") ;
Console.WriteLine("Eigenvectors of the matrix A : " ) ;

for (j=0; j<col; j++ )
{
  Console.WriteLine("Eigenvector {0} is : ", (j+1).ToString()  ) ;

  for (i=0; i<row; i++)
  {
    Console.Write( db_eigenvectorsReal.GetValue(i,j) + " + "  ) ;
    Console.Write( db_eigenvectorsImag.GetValue(i,j) + "i" + "\n") ;
  }

  Console.Write("\n") ;
```

```
      }

      Console.Write("\n") ;
      /* free memory */
      mw_A.Dispose() ;
      MWNumericArray.DisposeArray(mw_ArrayOut) ;

      mw_eigenvectors.Dispose() ;
      mw_eigenvalues.Dispose() ;
    }

/* **************************** */
/* **************************** */
/* **************************** */
  }
}
```

end code

Remarks

1. The MATLAB function `[V,D] = eig(A)` assigns eigenvalues `D` as a diagonal matrix, in which the eigenvalues are diagonal terms. The above programming gives the eigenvalues in the matrix form as follow:

$$eigenvalues = \begin{bmatrix} -0.30710 & 0.00000 & 0.00000 \\ 0.00000 & -0.24645 + 1.76008i & 0.00000 \\ 0.00000 & 0.00000 & -0.24645 - 1.76008i \end{bmatrix}$$

   then the eigenvalues of the matrix **A** are:

$$eigenvalue_1 = -0.30710$$

$$eigenvalue_2 = -0.24645 + 1.76008i$$

$$eigenvalue_3 = -0.24645 - 1.76008i$$

2. The MATLAB function `[V,D] = eig(A)` also assigns eigenvectors `V` as a matrix, in which

the eigenvectors are matrix columns. The above programming gives the value as follows:

$$eigenvectors = \begin{bmatrix} -0.83261 & 0.20027 - 0.13936i & 0.20027 + 0.13936i \\ -0.35534 & -0.21104 - 0.64472i & -0.21104 + 0.64472i \\ -0.42485 & -0.69301 & -0.69301 \end{bmatrix}$$

then the eigenvectors of the matrix **A** are:

$$eigenvector_1 = \begin{bmatrix} -0.83261 \\ -0.35534 \\ -0.42485 \end{bmatrix}$$

$$eigenvector_2 = \begin{bmatrix} 0.20027 - 0.13936i \\ -0.21104 - 0.64472i \\ -0.69301 \end{bmatrix}$$

$$eigenvector_3 = \begin{bmatrix} 0.20027 + 0.13936i \\ -0.21104 + 0.64472i \\ -0.69301 \end{bmatrix}$$

Chapter 13

Random Numbers

In this chapter we'll generate a class ***RandonNumber*** from common M-files working on matrix computation problems. The generated functions of this class will be used in a MSVC# .Net 2008 project to solve random number problems.

We will write two M-files, `myrand.m` and `myrandn.m` as shown below to generate the class ***RandonNumber***.

myrand.m

```
function Y = myrand(m,n)

Y = rand(m,n) ;
```

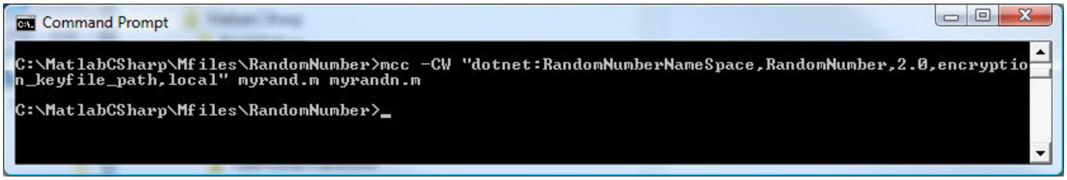
myrandn.m

```
function Y = myrandn(m,n)

Y = randn(m,n) ;
%
```

The following procedure to create the class ***RandomNumber*** is the same as the procedure in Chapter 2 as follows:

1. Write the following command in **Command Prompt** to generate the name space ***RandomNumberNameSpace*** that contains the class ***RandomNumber*** is (see Fig.13.1):

```
mcc -CW "dotnet:RandomNumberNameSpace,RandomNumber,2.0,encryption_keyfile_path,
local" myrand.m myrandn.m
```



```
C:\MatlabCSharp\Mfiles\RandomNumber>mcc -CW "dotnet:RandomNumberNameSpace,RandomNumber,2.0,encryptio
n_keyfile_path,local" myrand.m myrandn.m

C:\MatlabCSharp\Mfiles\RandomNumber>_
```

Figure 13.1: Command of creating the class ***RandomNumber***

2. Create a regular C# project in Windows Application of Microsoft Visual Studio 2008.
3. Click **Build**, then click **Rebuild** to make sure the program works.
4. In the project menu, click **Project**, **Add Reference**. Then the project will pop up a dialog to choose a reference.
5. Click on tab **.Net**, **MathWorks, .NET MWArray API**. Then the project will add MATLAB array wrapper classes for .NET, **MWArray** into the C# project.
6. Copy the two files `dotnet_mcc_component_data.cs` and `RandomNumber.cs` from previous steps and put them into the same directory of the `Program.cs` file.
7. Copy the file `RandomNumberNameSpace.ctf` and put it into the same directory of the project executable file, `Example.exe`.
8. Add the two files to the C# project by click **Project** on the project menu, then click **Add Existing Item**. The project will pop up a dialog then choose the two files `dotnet_mcc_component_data.cs` and `RandomNumber.cs`. This step will add these two files to the project.
9. At the top of the file `Program.cs`, add the following:

```
using MathWorks.MATLAB.NET.Arrays;
using MathWorks.MATLAB.NET.Utility;
using RandomNumberNameSpace;
```

13.1 Uniform Random Numbers

Uniform random numbers are random numbers that lie within a specific range. This section describes how to use the MATLAB function myrand(..) in the generated class to solve uniform random number problems. The functions of this class uses the MATLAB function $rand(m, n)$ to generate a matrix (size $m \times n$) of random numbers. These random numbers are uniformly distributed in the interval (0,1). For more information of this function $rand(m, n)$, refer to the MATLAB manual [6].

13.1.1 Generating Uniform Random Numbers in Range [0,1]

Problem 1

input . A number, $N = 5$

output . Generating N uniform random numbers in range $[0, 1]$

The following subroutine uses a function in the class ***RandomNumber*** to solve Problem 1.

Listing code

```
public void UniformRandom_vector()
{
  int row = 1 ;
  int col = 5 ;

  MWNumericArray uniformRandVector = null ;
  MWNumericArray mw_row = new MWNumericArray(row) ;
  MWNumericArray mw_col = new MWNumericArray(col) ;

  RandomNumber objMatlab = new RandomNumber() ;
  uniformRandVector = (MWNumericArray) objMatlab.myrand(mw_row, mw_col);

  Console.WriteLine("Uniform random numbers from 0 to 1:");
  Console.WriteLine(uniformRandVector);
  // Or convert back to double
  double[] db_uniformRandVector = (double[])uniformRandVector.ToVector(MWArrayComponent.Real) ;
  Console.WriteLine() ;
  PrintVector(db_uniformRandVector) ;

  uniformRandVector.Dispose() ;
}
```

end code

13.1.2 Generating Uniform Random Numbers in Range [a,b]

Problem 2

input . A number, $N = 6$
. A range $[a, b]$, where a=2 and b=18

output . Generating N uniform random numbers in the range $[a, b]$

The following subroutine uses a function in the class ***RandomNumber*** to solve Problem 2.

Listing code

```
public void UniformRandom_vector2()
{
  double a = 2.0    ;
  double b = 18.0   ;

  int row = 1 ;
  int col = 6 ;
  int i ;

  // Get random numbers in [0, 1]
  MWNumericArray uniformRandVector = null ;

  MWNumericArray mw_row = new MWNumericArray(row) ;
  MWNumericArray mw_col = new MWNumericArray(col) ;
  RandomNumber objMatlab = new RandomNumber() ;
  uniformRandVector = (MWNumericArray) objMatlab.myrand(mw_row, mw_col);

  // Convert back to double
  double[] db_uniformRandVector = (double[])uniformRandVector.ToVector(MWArrayComponent.Real) ;

  Console.WriteLine() ;
  Console.WriteLine("Uniform random numbers from a=2 to b=18 : ") ;

  for (i=0; i<db_uniformRandVector.Length; i++)  {
    double aVal = a + (b-a)*db_uniformRandVector[i] ;
    Console.WriteLine( aVal.ToString() ) ;
  }

  // Free memories
  uniformRandVector.Dispose() ;
}
```

end code

13.1.3 Generating a Matrix of Uniform Random Numbers in Range [0,1]

In this section, we will use MATLAB functions to solve the following problems.

Problem 3

input . A row number $m = 8$ and a column number $n = 5$

output . Generating a matrix (size $m \times n$) of uniform random numbers in the range [0,1]

The following subroutine uses a function in the class ***RandomNumber*** to solve Problem 3.

Listing code

```
public void UniformRandom_matrix()
{
  int row = 8 ;
  int col = 5 ;

  MWNumericArray mw_uniformRandMatrix = null ;
  MWNumericArray mw_row = new MWNumericArray(row) ;
  MWNumericArray mw_col = new MWNumericArray(col) ;

  RandomNumber objMatlab = new RandomNumber() ;
  mw_uniformRandMatrix = (MWNumericArray)objMatlab.myrand(mw_row, mw_col);

  Console.WriteLine("The matrix of uniform random numbers from 0 to 1 :" ) ;
  Console.WriteLine(mw_uniformRandMatrix);

  // Or convert back to  double
  double [,] db_uniformRandMatrix = (double[,])mw_uniformRandMatrix.ToArray(
                                                      MWArrayComponent.Real) ;
  Console.WriteLine() ;
  PrintMatrix(db_uniformRandMatrix) ;

  mw_uniformRandMatrix.Dispose() ;
}
```

end code

13.1.4 Generating a Matrix of Uniform Random Numbers in Range [a,b]

In this section, we will use MATLAB functions to solve the following problems.

Problem 4

input . A row number $m = 8$ and a column number $n = 5$

. Range $[a, b]$, where a=4 and b=17

output . Generating a matrix (size $m \times n$) of uniform random numbers in the range $[a, b]$

The following subroutine uses a function in the class ***RandomNumber*** to solve Problem 4.

Listing code

```
public void UniformRandom_matrix2()
{
  int row = 8 ;
  int col = 5 ;
  int i, j ;

  double a = 4.0  ;
  double b = 17.0 ;

  MWNumericArray mw_uniformRandMatrix = null ;
  MWNumericArray mw_row = new MWNumericArray(row) ;
  MWNumericArray mw_col = new MWNumericArray(col) ;

  // Get matrix of random number from 0 to 1
  RandomNumber objMatlab = new RandomNumber() ;
  mw_uniformRandMatrix = (MWNumericArray)objMatlab.myrand(mw_row, mw_col);

  // Convert to double
  double[,] db_uniformRandMatrix = (double[,])mw_uniformRandMatrix.ToArray(
                                                       MWArrayComponent.Real) ;
  // Print out
  Console.WriteLine() ;
  Console.WriteLine("The matrix of uniform random numbers from a=4.0 to b=17.0:") ;

  for (i=0; i<row; i++)  {
    for (j=0; j<col; j++)  {
      Console.WriteLine( a + (b-a)*db_uniformRandMatrix[i,j] + "\t");
    }
    Console.WriteLine() ;
  }

  Console.WriteLine() ;

  // Free memories
  mw_uniformRandMatrix.Dispose() ;
}
```

end code

13.2 Normal Random Numbers

Normal random numbers are random numbers that establish the normal distribution (Gaussian distribution). This section describe how to use the function in the generated class ***RandomNumber*** to solve normal random number problems. These functions use the MATLAB function $randn(m, n)$ to generate a matrix (size n by m) of random numbers. These random numbers are normally distributed with specified properties, $\mu = 0$, variance $\sigma^2 = 1$.

13.2.1 Generating Normal Random Numbers with mean=0 and variance=1

In this section, we will use MATLAB functions to solve the following problems.

Problem 5

input . A number, $N = 5$

output . Generating (N) normal random numbers with :
mean $\mu = 0$
variance $\sigma^2 = 1$

The following subroutine uses a function in the class ***RandomNumber*** to solve Problem 5 .

Listing code

```
public void NormalRandom_vector()
{
  int row = 1 ;
  int col = 5 ;

  MWNumericArray mw_normalRandVector = null ;
  MWNumericArray mw_row = new MWNumericArray(row) ;
  MWNumericArray mw_col = new MWNumericArray(col) ;

  RandomNumber objMatlab = new RandomNumber() ;
  mw_normalRandVector = (MWNumericArray)objMatlab.myrandn(mw_row, mw_col) ;

  Console.WriteLine("Normal random numbers :") ;
  Console.WriteLine(mw_normalRandVector) ;

  // Or convert back to double
  double [] db_normalRandVector = (double[])mw_normalRandVector.ToVector(
                                                      MWArrayComponent.Real) ;
  Console.WriteLine() ;
  PrintVector(db_normalRandVector) ;

  // Free memories
  mw_normalRandVector.Dispose() ;
}
```

end code

13.2.2 Generating Normal Random Numbers with mean=a and variance=b

Problem 6

input . A number, $N = 5$

output . Generating N normal random numbers with specified properties:

mean $\mu = 0.56$

variance $\sigma^2 = 0.12$

The following subroutine uses a function in the class ***RandomNumber*** to solve Problem 6.

Listing code

```
public void NormalRandom_vector2()
{
  /* Generate a vector of normal random numbers at
   particular mean and variance */
  int row = 1 ;
  int col = 5 ;

  double mean_mu  = 0.56 ;
  double variance = 0.12 ;
  int i ;

  MWNumericArray mw_normalRandVector = null ;
  MWNumericArray mw_row = new MWNumericArray(row) ;
  MWNumericArray mw_col = new MWNumericArray(col) ;

  RandomNumber objMatlab = new RandomNumber() ;
  mw_normalRandVector = (MWNumericArray)objMatlab.myrandn(mw_row, mw_col) ;

  // Convert to double
  double[] db_normalRandVector = (double[])mw_normalRandVector.ToVector(
                                                         MWArrayComponent.Real) ;
  double standard_deviation = Math.Sqrt(variance) ;

  // Print out
  Console.WriteLine("Normal random numbers with mean = 0.56 and variance = 0.12 :") ;

  for (i=0; i<col; i++)  {
    double aVal = mean_mu + standard_deviation*db_normalRandVector[i] ;
    Console.WriteLine(aVal.ToString() + "\t") ;
  }

  Console.WriteLine() ;

  // Free memories
  mw_normalRandVector.Dispose() ;
}
```

end code

13.2.3 Generating a Matrix of Normal Random Numbers with mean=0 and variance=1

Problem 7

input . A row number $m = 8$ and a column number $n = 5$

output . Generating a matrix (size $m \times n$) of normal random numbers
with specified properties:
mean $\mu = 0$
variance $\sigma^2 = 1$

The following subroutine uses a function in the class ***RandomNumber*** to solve Problem 7.

Listing code

```
public void NormalRandom_matrix()
{
  int row = 8 ;
  int col = 5 ;

  RandomNumber objMatlab = new RandomNumber() ;

  MWNumericArray mw_normalRandMatrix = null ;
  MWNumericArray mw_row = new MWNumericArray(row) ;
  MWNumericArray mw_col = new MWNumericArray(col) ;

  // Get matrix of random number from 0 to 1
  mw_normalRandMatrix = (MWNumericArray)objMatlab.myrandn(mw_row, mw_col);

  Console.WriteLine(mw_normalRandMatrix) ;

  // Or convert to double
  double[,] db_normalRandMatrix = (double[,])mw_normalRandMatrix.ToArray(
                                                      MWArrayComponent.Real) ;
  Console.WriteLine() ;
  PrintMatrix(db_normalRandMatrix) ;

  // Free memories
  mw_normalRandMatrix.Dispose() ;
}
```

end code

13.2.4 Generating a Matrix of Normal Random Numbers with mean=a and variance=b

Problem 8

input . A row number $m = 8$ and a column number $n = 5$

output . Generating a matrix (size $m \times n$) of normal random numbers with specified properties:

mean $\mu = 0.56$

variance $\sigma^2 = 0.12$

The following subroutine uses a function in the class ***RandomNumber*** to solve Problem 8.

Listing code

```
public void NormalRandom_matrix2()
{
  int row = 8 ;
  int col = 5 ;
  int i, j ;

  double mean_mu  = 0.56 ;
  double variance = 0.12 ;

  double standard_deviation = Math.Sqrt(variance) ;
  RandomNumber objMatlab = new RandomNumber() ;

  MWNumericArray mw_normalRandMatrix = null ;
  MWNumericArray mw_row = new MWNumericArray(row) ;
  MWNumericArray mw_col = new MWNumericArray(col) ;

  // Get matrix of normal random number
  mw_normalRandMatrix = (MWNumericArray)objMatlab.myrandn(mw_row, mw_col);

  // Convert to double
  double[,] db_normalRandMatrix = (double[,])mw_normalRandMatrix.ToArray(
                                                       MWArrayComponent.Real);
  // Print out
  Console.WriteLine() ;
  Console.WriteLine(@"The matrix of normal random numbers ");
  Console.WriteLine("at specified mean and variance" );

  for (i=0; i<row; i++)  {
    for (j=0; j<col; j++)  {

      double aVal = mean_mu + standard_deviation*db_normalRandMatrix[i,j] ;
      Console.WriteLine( aVal.ToString() ) ;
    }
    Console.WriteLine() ;
  }

  Console.WriteLine() ;

  // Free memories
  mw_normalRandMatrix.Dispose() ;
}
```

end code

The following is the full code for this chapter.

Listing code

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using MathWorks.MATLAB.NET.Arrays;
using MathWorks.MATLAB.NET.Utility;
using RandomNumberNameSpace ;

namespace Example
{
  class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine("Random Number") ;
      Program objProgram = new Program() ;

      Console.WriteLine() ;
      Console.WriteLine("1. Vector of uniform random number from 0 to 1") ;
      objProgram.UniformRandom_vector() ;

      Console.WriteLine() ;
      Console.WriteLine("2. Vector of uniform random numbers from a to b ") ;
      objProgram.UniformRandom_vector2() ;

      Console.WriteLine() ;
      Console.WriteLine("3. Matrix of random numbers") ;
      objProgram.UniformRandom_matrix() ;

      Console.WriteLine() ;
      Console.WriteLine("4. Matrix of random numbers from a to b") ;
      objProgram.UniformRandom_matrix2() ;

      Console.WriteLine() ;
      Console.WriteLine("5. Vector of normal random number") ;
      objProgram.NormalRandom_vector() ;

      Console.WriteLine() ;
      Console.WriteLine("6. Vector of normal random number with mean and variance") ;
      objProgram.NormalRandom_vector2() ;

      Console.WriteLine() ;
      Console.WriteLine("7. Matrix of normal random number") ;
      objProgram.NormalRandom_matrix() ;

      Console.WriteLine() ;
      Console.WriteLine("8. Matrix of normal random number with mean and variance") ;
      objProgram.NormalRandom_matrix2() ;
    }

    /* **************************** */
    public void UniformRandom_vector()
    {
      int row = 1 ;
      int col = 5 ;

      MWNumericArray uniformRandVector = null ;
      MWNumericArray mw_row = new MWNumericArray(row) ;
      MWNumericArray mw_col = new MWNumericArray(col) ;

      RandomNumber objMatlab = new RandomNumber() ;
      uniformRandVector = (MWNumericArray) objMatlab.myrand(mw_row, mw_col);

      Console.WriteLine("Uniform random numbers from 0 to 1:");
      Console.WriteLine(uniformRandVector);

      // Or convert back to double
      double[] db_uniformRandVector = (double[])uniformRandVector.ToVector(
                                                          MWArrayComponent.Real) ;
      Console.WriteLine() ;
```

```
      PrintVector(db_uniformRandVector) ;

      uniformRandVector.Dispose() ;
    }

    /*  ********************************** */
    public void UniformRandom_vector2()
    {
      double a = 2.0 ;
      double b = 18.0 ;

      int row = 1 ;
      int col = 6 ;
      int i ;

      // Get random numbers in [0, 1]
      MWNumericArray uniformRandVector = null ;

      MWNumericArray mw_row = new MWNumericArray(row) ;
      MWNumericArray mw_col = new MWNumericArray(col) ;

      RandomNumber objMatlab = new RandomNumber() ;
      uniformRandVector = (MWNumericArray) objMatlab.myrand(mw_row, mw_col);

      // Convert back to double
      double[] db_uniformRandVector = (double[])uniformRandVector.ToVector(
                                                           MWArrayComponent.Real) ;

      Console.WriteLine() ;
      Console.WriteLine("Uniform random numbers from a=2 to b=18 : ") ;

      for (i=0; i<db_uniformRandVector.Length; i++)  {
        double aVal = a + (b-a)*db_uniformRandVector[i] ;
        Console.WriteLine( aVal.ToString() ) ;
      }
      // Free memories
      uniformRandVector.Dispose() ;
    }

    /* ************************************* */
    public void UniformRandom_matrix()
    {
      int row = 8 ;
      int col = 5 ;

      MWNumericArray mw_uniformRandMatrix = null ;
      MWNumericArray mw_row = new MWNumericArray(row) ;
      MWNumericArray mw_col = new MWNumericArray(col) ;

      RandomNumber objMatlab = new RandomNumber() ;
      mw_uniformRandMatrix = (MWNumericArray)objMatlab.myrand(mw_row, mw_col);

      Console.WriteLine("The matrix of uniform random numbers from 0 to 1 :" ) ;
      Console.WriteLine(mw_uniformRandMatrix);

      // Or convert back to  double
      double [,] db_uniformRandMatrix = (double[,])mw_uniformRandMatrix.ToArray(
                                                                MWArrayComponent.Real) ;
      Console.WriteLine() ;
      PrintMatrix(db_uniformRandMatrix) ;

      mw_uniformRandMatrix.Dispose() ;
    }

    /* ********************************* */
    public void UniformRandom_matrix2()
    {
      int row = 8 ;
      int col = 5 ;
      int i, j ;
      double a = 4.0  ;
      double b = 17.0 ;

      MWNumericArray mw_uniformRandMatrix = null ;
      MWNumericArray mw_row = new MWNumericArray(row) ;
      MWNumericArray mw_col = new MWNumericArray(col) ;

      // Get matrix of random number from 0 to 1
      RandomNumber objMatlab = new RandomNumber() ;
      mw_uniformRandMatrix = (MWNumericArray)objMatlab.myrand(mw_row, mw_col);

      // Convert to double
      double[,] db_uniformRandMatrix = (double[,])mw_uniformRandMatrix.ToArray(
                                                                MWArrayComponent.Real) ;

      // Print out
      Console.WriteLine() ;
      Console.WriteLine("The matrix of uniform random numbers from a=4.0 to b=17.0:") ;

      for (i=0; i<row; i++)  {
        for (j=0; j<col; j++)  {
          Console.WriteLine( a + (b-a)*db_uniformRandMatrix[i,j] + "\t");
        }
        Console.WriteLine() ;
      }

      Console.WriteLine() ;

      // Free memories
      mw_uniformRandMatrix.Dispose() ;
    }

    /* ********************************* */
    public void NormalRandom_vector()
    {
      int row = 1 ;
      int col = 5 ;

      MWNumericArray mw_normalRandVector = null ;
      MWNumericArray mw_row = new MWNumericArray(row) ;
      MWNumericArray mw_col = new MWNumericArray(col) ;

      RandomNumber objMatlab = new RandomNumber() ;
      mw_normalRandVector = (MWNumericArray)objMatlab.myrandn(mw_row, mw_col) ;

      Console.WriteLine("Normal random numbers :") ;
      Console.WriteLine(mw_normalRandVector) ;

      // Or convert back to double
      double [] db_normalRandVector = (double[])mw_normalRandVector.ToVector(
                                                                 MWArrayComponent.Real) ;
      Console.WriteLine() ;
      PrintVector(db_normalRandVector) ;

      // Free memories
      mw_normalRandVector.Dispose() ;
    }

    /* **************************** */
    public void NormalRandom_vector2()
    {
      /* Generate a vector of normal random numbers at
       particular mean and variance */
      int row = 1 ;
      int col = 5 ;

      double mean_mu  = 0.56 ;
      double variance = 0.12 ;
      int i ;

      MWNumericArray mw_normalRandVector = null ;
      MWNumericArray mw_row = new MWNumericArray(row) ;
```

```
    MWNumericArray mw_col = new MWNumericArray(col) ;

    RandomNumber objMatlab = new RandomNumber() ;
    mw_normalRandVector = (MWNumericArray)objMatlab.myrandn(mw_row, mw_col) ;

    // Convert to double
    double[] db_normalRandVector = (double[])mw_normalRandVector.ToVector(
                                                        MWArrayComponent.Real) ;

    double standard_deviation = Math.Sqrt(variance) ;

    // Print out
    Console.WriteLine("Normal random numbers with mean = 0.56 and variance = 0.12 :") ;

    for (i=0; i<col; i++)  {
      double aVal = mean_mu + standard_deviation*db_normalRandVector[i] ;
      Console.WriteLine(aVal.ToString() + "\t") ;
    }

    Console.WriteLine() ;

    // Free memories
    mw_normalRandVector.Dispose() ;
  }

  /* ************************************* */
  public void NormalRandom_matrix()
  {
    int row = 8 ;
    int col = 5 ;

    RandomNumber objMatlab = new RandomNumber() ;

    MWNumericArray mw_normalRandMatrix = null ;
    MWNumericArray mw_row = new MWNumericArray(row) ;
    MWNumericArray mw_col = new MWNumericArray(col) ;
    // Get matrix of random number from 0 to 1
    mw_normalRandMatrix = (MWNumericArray)objMatlab.myrandn(mw_row, mw_col);

    Console.WriteLine(mw_normalRandMatrix) ;

    // Or convert to double
    double[,] db_normalRandMatrix = (double[,])mw_normalRandMatrix.ToArray(
                                                              MWArrayComponent.Real) ;
    Console.WriteLine() ;
    PrintMatrix(db_normalRandMatrix) ;

    // Free memories
    mw_normalRandMatrix.Dispose() ;
  }

  /* **************************** */
  public void NormalRandom_matrix2()
  {
    int row = 8 ;
    int col = 5 ;
    int i, j ;

    double mean_mu  = 0.56 ;
    double variance = 0.12 ;

    double standard_deviation = Math.Sqrt(variance) ;

    RandomNumber objMatlab = new RandomNumber() ;

    MWNumericArray mw_normalRandMatrix = null ;
    MWNumericArray mw_row = new MWNumericArray(row) ;
    MWNumericArray mw_col = new MWNumericArray(col) ;

    // Get matrix of normal random number
    mw_normalRandMatrix = (MWNumericArray)objMatlab.myrandn(mw_row, mw_col);
```

```
    // Convert to double
    double[,] db_normalRandMatrix = (double[,])mw_normalRandMatrix.ToArray(
                                                         MWArrayComponent.Real);

    // Print out
    Console.WriteLine() ;
    Console.WriteLine(@"The matrix of normal random numbers ");
    Console.WriteLine("at specified mean and variance" );

    for (i=0; i<row; i++)  {
      for (j=0; j<col; j++)  {

        double aVal = mean_mu + standard_deviation*db_normalRandMatrix[i,j] ;
        Console.WriteLine( aVal.ToString() ) ;
      }
      Console.WriteLine() ;
    }

    Console.WriteLine() ;

    // Free memories
    mw_normalRandMatrix.Dispose() ;
  }

  /* ******************************** */
  public static void PrintVector(double[] vector)
  {
    int i, row;
    row = vector.GetUpperBound(0) - vector.GetLowerBound(0) + 1;

    for (i = 0; i < row; i++)
    {
      Console.Write("{0} \t", vector.GetValue(i).ToString());
      Console.WriteLine();
    }
    }

    /* ******************************** */
    public static void PrintMatrix(double[,] matrix)
    {
      int i, j, row, col ;
      row = matrix.GetUpperBound(0) - matrix.GetLowerBound(0) + 1;
      col = matrix.GetUpperBound(1) - matrix.GetLowerBound(1) + 1;

      for (i=0; i<row; i++)
      {
        for (j=0; j<col; j++)
        {
          Console.Write("{0} \t", matrix.GetValue(i,j).ToString() ) ;
        }
        Console.WriteLine() ;
      }

    }

/* ******************************** */
/* ******************************** */
/* ******************************** */
  }
}
```

end code

Part II:

Using MATLAB functions in C# Windows Form Applications

Chapter 14

Using MATLAB Functions In C# Windows Forms Applications

14.1 Using MATLAB Built-in Functions to Calculate a Addition Matrix

This chapter describes how to use MATLAB functions in a C# Windows Form application. The steps of this application are:

1. Get input data from a Windows Form application.
2. Assign input data to variables in C# *double* type.
3. Use MATLAB built-in functions in the generated class to perform the task.
4. Transfer back result values to the C# *double* type.
5. Put the result to the Windows Form application.

An example of the Windows Forms application in this section is a simple application of the matrix addition.
We will write the M-files as shown below to generate a class for a Windows Form application .

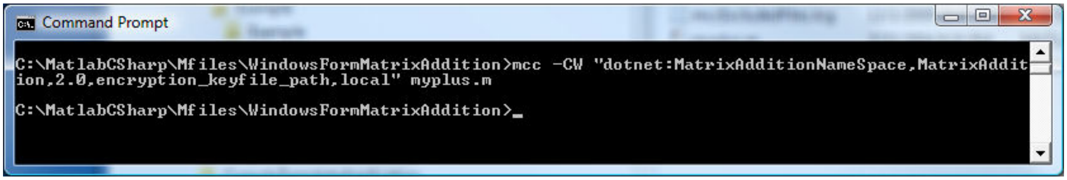
myplus.m

```
function y = myplus(a, b)


y = a + b ;
```

The following procedure to create the class ***MatrixAddition*** is the same as the procedure in Chapter 2 as follows:

1. Write the following command in **Command Prompt** to generate the name space ***MatrixAdditionNameSpace*** that contains the class ***MatrixAddition*** is (see Fig.14.1):

   ```
   mcc -CW "dotnet:MatrixAdditionNameSpace,MatrixAddition,2.0,encryption_keyfile_path,
   local" myplus.m
   ```



```
C:\MatlabCSharp\Mfiles\WindowsFormMatrixAddition>mcc -CW "dotnet:MatrixAdditionNameSpace,MatrixAddit
ion,2.0,encryption_keyfile_path,local" myplus.m

C:\MatlabCSharp\Mfiles\WindowsFormMatrixAddition>_
```

Figure 14.1: Command of creating the class ***MatrixAddition***

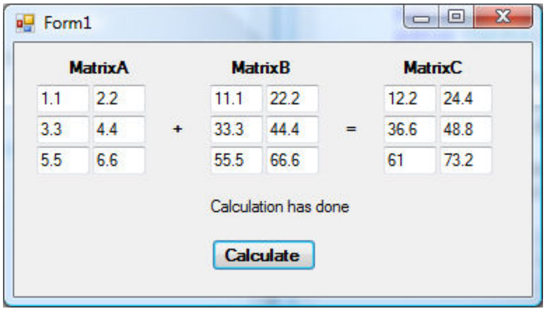
2. Create a regular C# project in Windows Application of Microsoft Visual Studio 2008.
3. Click **Build**, then click **Rebuild** to make sure the program works.
4. In the project menu, click **Project**, **Add Reference**. Then the project will pop up a dialog to choose a reference.
5. Click on tab **.Net**, **MathWorks, .NET MWArray API**. Then the project will add MATLAB array wrapper classes for .NET, **MWArray** into the C# project.
6. Copy the two files `dotnet_mcc_component_data.cs` and `MatrixAddition.cs` from previous steps and put them into the same directory of the `Program.cs` file.
7. Copy the file `MatrixAdditionNameSpace.ctf` and put it into the same directory of the project executable file, `Example.exe`.

8. Add the two files to the C# project by click **Project** on the project menu, then click **Add Existing Item**. The project will pop up a dialog then choose the two files `dotnet_mcc_component_data.cs` and `MatrixAddition.cs`. This step will add these two files to the project.

9. At the top of the file `Program.cs`, add the following:

```
using MathWorks.MATLAB.NET.Arrays;
using MathWorks.MATLAB.NET.Utility;
using MatrixAdditionNameSpace;
```

The form of this Windows Form application shows in Fig.14.2



MatrixA			MatrixB			MatrixC	
1.1	2.2		11.1	22.2		12.2	24.4
3.3	4.4	+	33.3	44.4	=	36.6	48.8
5.5	6.6		55.5	66.6		61	73.2

Calculation has done

Figure 14.2: Matrix Addition in Windows Forms Application

The following is the code of this example for the Windows Form application.

Listing code

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

using MathWorks.MATLAB.NET.Utility;
using MathWorks.MATLAB.NET.Arrays ;
using MatrixAdditionNameSpace ;

namespace Example
{
  public partial class Form1 : Form
  {
    public TextBox[,] TextBoxMatrixA ;
    public TextBox[,] TextBoxMatrixB ;
    public TextBox[,] TextBoxMatrixC ;

    public Point StartPoint = new Point() ;
    int TextboxLength ;
    int TextboxHeight ;

    public int NumRow = 3 ;
    public int NumCol = 2 ;

    public double [,] MatrixA_data ;
    public double [,] MatrixB_data ;
    public double [,] MatrixC_data ;

    public Form1()
    {
      InitializeComponent();
      InitMatrixes() ;
    }

    /* *************************** */
    public void InitMatrixes()
    {
      TextBoxMatrixA = (TextBox[,])Array.CreateInstance( typeof(TextBox), NumRow, NumCol ) ;
      TextBoxMatrixB = (TextBox[,])Array.CreateInstance( typeof(TextBox), NumRow, NumCol ) ;
      TextBoxMatrixC = (TextBox[,])Array.CreateInstance( typeof(TextBox), NumRow, NumCol ) ;

      MatrixA_data = (double[,])Array.CreateInstance( typeof(double), NumRow, NumCol ) ;
      MatrixB_data = (double[,])Array.CreateInstance( typeof(double), NumRow, NumCol ) ;
      MatrixC_data = (double[,])Array.CreateInstance( typeof(double), NumRow, NumCol ) ;

      StartPoint.X = 16 ;
      StartPoint.Y = 30 ;

      TextboxLength = 38 ;
      TextboxHeight = 20 ;

      SetTextBoxPositionMatrixA() ;
      SetTextBoxPositionMatrixB() ;

      labelPlusSign.Location = new System.Drawing.Point(StartPoint.X - 30,
                                                       (StartPoint.Y + 24) );
      SetTextBoxPositionMatrixC() ;
      labelEqualSign.Location = new System.Drawing.Point(StartPoint.X - 30,
                                                        (StartPoint.Y + 24) );
    }

    /* *************************** */
    public void SetTextBoxPositionMatrixA()
    {
      for(int i=0; i<NumRow; i++)
      {
        for(int j=0; j<NumCol; j++)
        {
          TextBoxMatrixA[i,j] = new System.Windows.Forms.TextBox();
          TextBoxMatrixA[i,j].Location = new System.Drawing.Point(
                 StartPoint.X + (TextboxLength+2)*j, StartPoint.Y + (TextboxHeight+2)*i );
          TextBoxMatrixA[i,j].Name = "textBox1";
          TextBoxMatrixA[i,j].Size = new System.Drawing.Size(TextboxLength, TextboxHeight);
          TextBoxMatrixA[i,j].TabIndex = i;
          TextBoxMatrixA[i,j].Text = "0.0";
          this.Controls.Add(TextBoxMatrixA[i,j]) ;
        }
      }
    }
```

```
/* ************************** */
public void SetTextBoxPositionMatrixB()
{
  StartPoint.X = StartPoint.X + (NumRow*TextboxLength + 10) ;

  for(int i=0; i<NumRow; i++)
  {
    for(int j=0; j<NumCol; j++)
    {
      TextBoxMatrixB[i,j] = new System.Windows.Forms.TextBox();
      TextBoxMatrixB[i,j].Location = new System.Drawing.Point(
              StartPoint.X + (TextboxLength+2)*j, StartPoint.Y + (TextboxHeight+2)*i );
      TextBoxMatrixB[i,j].Name = "textBox1";
      TextBoxMatrixB[i,j].Size = new System.Drawing.Size(TextboxLength, TextboxHeight);
      TextBoxMatrixB[i,j].TabIndex = i;
      TextBoxMatrixB[i,j].Text = "0.0";
      this.Controls.Add(TextBoxMatrixB[i,j]) ;
    }
  }
}

/* ************************** */
public void SetTextBoxPositionMatrixC()
{
  StartPoint.X = StartPoint.X + (NumRow*TextboxLength + 10) ;
  for(int i=0; i<NumRow; i++)
  {
    for(int j=0; j<NumCol; j++)
    {
      TextBoxMatrixC[i,j] = new System.Windows.Forms.TextBox();
      TextBoxMatrixC[i,j].Location = new System.Drawing.Point(
                   StartPoint.X + (TextboxLength+2)*j, StartPoint.Y + (TextboxHeight+2)*i );
      TextBoxMatrixC[i,j].Name = "textBox1";
      TextBoxMatrixC[i,j].Size = new System.Drawing.Size(TextboxLength, TextboxHeight);
      TextBoxMatrixC[i,j].TabIndex = i;
      TextBoxMatrixC[i,j].Text = "0.0";
      this.Controls.Add(TextBoxMatrixC[i,j]) ;
    }
  }
}

/* ************************** */
private void buttonMatrixAddition_Click(object sender, EventArgs e)
{
  ////////// get matrix A /////////////
  try
  {
    GetMatrixA() ;
  }
  catch
  {
    MessageBox.Show("Data in matrix A is not valid") ;
    return ;
  }

  ////////// get matrix B /////////////
  try
  {
    GetMatrixB() ;
  }
  catch
  {
    MessageBox.Show("Data in matrix B is not valid") ;
    return ;
  }

  this.labelCalcInfo.Text = @"Calculation is in process. Please wait.";
  this.Invalidate() ;
  this.Refresh() ;

  /////////////// calculate matrix C ////////////
  GetMatrixC() ;
}

/* ************************** */
public void GetMatrixA()
{
  int i, j;
  for( i=0; i<NumRow; i++)
  {
    for( j=0; j<NumCol; j++)
    {
      MatrixA_data[i,j] = Convert.ToDouble(TextBoxMatrixA[i,j].Text) ;
    }
  }
}

/* ************************** */
public void GetMatrixB()
{
  int i, j;
  for( i=0; i<NumRow; i++)
  {
    for( j=0; j<NumCol; j++)
    {
      MatrixB_data[i,j] = Convert.ToDouble(TextBoxMatrixB[i,j].Text) ;
    }
  }
}

/* ************************** */
public void GetMatrixC()
{
  // calculate matrix multiplication
  MWNumericArray mw_A = new MWNumericArray(MatrixA_data) ;
  MWNumericArray mw_B = new MWNumericArray(MatrixB_data) ;
  MWNumericArray mw_C = null ;
```

```
      /* call an implemental function */
      MatrixAddition objMatlab = new MatrixAddition() ;
      mw_C = (MWNumericArray) objMatlab.myplus(mw_A, mw_B) ;

      /* convert back to C# double  */
      MatrixC_data = (double[,]) mw_C.ToArray(MWArrayComponent.Real) ;

      // display to GUI
      int i, j ;
      for( i=0; i<NumRow; i++)
      {
        for( j=0; j<NumCol; j++)
        {
          TextBoxMatrixC[i,j].Text = MatrixC_data[i,j].ToString() ;
        }
      }

      labelCalcInfo.Text = @"Calculation has done";

      mw_A.Dispose() ;
      mw_B.Dispose() ;
      mw_C.Dispose() ;
    }

/* *************************** */
/* *************************** */
/* *************************** */
  }
}
```

end code

14.2 Using MATLAB Built-in Functions to Calculate a Multiplication Matrix

This section describes how to use MATLAB functions in a C# Windows Form application. The tasks of this application are:

1. Get input data from the Windows Form application.
2. Assign input data to variables in the C# *double* type.
3. Use MATLAB built-in functions in the generated class to perform the task.
4. Transfer back result values to C# *double* type.
5. Put the result to the Windows Form application.

An example of the Windows Forms application in this section is a simple application of the matrix multiplication.

We will write the M-files as shown below to generate a class for a Windows Form application.

mymtimes.m

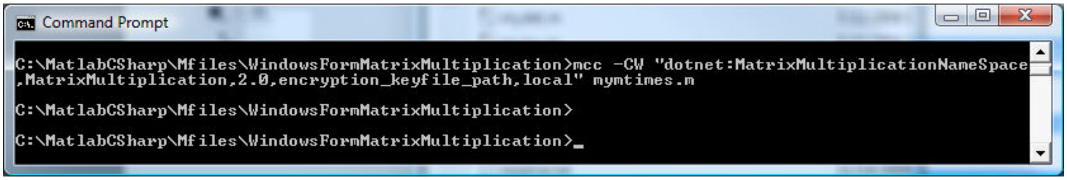
```
function y = mymtimes(a, b)


y = a*b ;
```

The following procedure to create the class ***MatrixMultiplication*** is the same as the procedure in Chapter 2 as follows:

1. Write the following command in **Command Prompt** to generate the name space ***MatrixMultiplicationNameSpace*** that contains the class ***MatrixMultiplication*** is (see Fig.14.3):

   ```
   mcc -CW "dotnet:MatrixMultiplicationNameSpace,MatrixMultiplication,2.0,
   encryption_keyfile_path,local" mymtimes.m
   ```

2. Create a regular C# project in Windows Application of Microsoft Visual Studio 2008.
3. Click **Build**, then click **Rebuild** to make sure the program works.



```
C:\MatlabCSharp\Mfiles\WindowsFormMatrixMultiplication>mcc -CW "dotnet:MatrixMultiplicationNameSpace
,MatrixMultiplication,2.0,encryption_keyfile_path,local" mymtimes.m

C:\MatlabCSharp\Mfiles\WindowsFormMatrixMultiplication>

C:\MatlabCSharp\Mfiles\WindowsFormMatrixMultiplication>_
```

Figure 14.3: Command of creating the class ***MatrixMultiplication***

4. In the project menu, click **Project**, **Add Reference**. Then the project will pop up a dialog to choose a reference.

5. Click on tab **.Net**, **MathWorks, .NET MWArray API**. Then the project will add MATLAB array wrapper classes for .NET, **MWArray** into the C# project.

6. Copy the two files `dotnet_mcc_component_data.cs` and `MatrixMultiplication.cs` from previous steps and put them into the same directory of the `Program.cs` file.

7. Copy the file `MatrixMultiplicationNameSpace.ctf` and put it into the same directory of the project executable file, `Example.exe`.

8. Add the two files to the C# project by click **Project** on the project menu, then click **Add Existing Item**. The project will pop up a dialog then choose the two files `dotnet_mcc_component_data.cs` and `MatrixMultiplication.cs`. This step will add these two files to the project.

9. At the top of the file `Program.cs`, add the following:

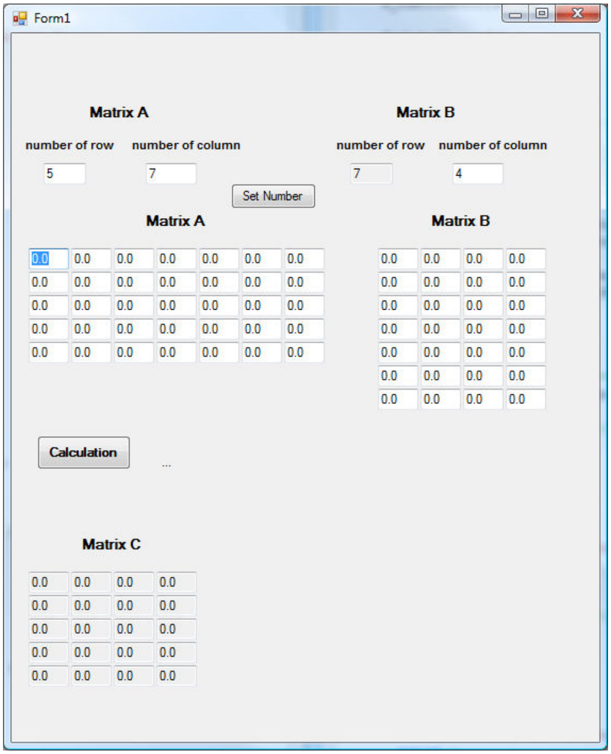
```
using MathWorks.MATLAB.NET.Arrays;
using MathWorks.MATLAB.NET.Utility;
using MatrixMultiplicationNameSpace;
```

The form of this Windows Form application shows in Fig.14.4

The following is code of this example for Windows Form application.

Listing code

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
```



Matrix A

number of row: 5

number of column: 7

Matrix B

number of row: 7

number of column: 4

Matrix A

0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0

Matrix B

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

Matrix C

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

Figure 14.4: Matrix Multiplication in Windows Forms Application

```
using System.Linq;
using System.Text;
using System.Windows.Forms;

using MathWorks.MATLAB.NET.Utility;
using MathWorks.MATLAB.NET.Arrays ;
using MatrixMultiplicationNameSpace ;

namespace Example
{
  public partial class Form1 : Form
  {
    public TextBox[,] arrTextBoxMatrixA ;
    public double[,] db_MatrixA  ;

    public TextBox[,] arrTextBoxMatrixB ;
    public double[,] db_MatrixB  ;

    public TextBox[,] arrTextBoxMatrixC ;
    public double[,] db_MatrixC  ;

    public int m_numRowMatricATextBox ;
    public int m_numColMatricATextBox ;

    public int m_tbLength ;
    public int m_tbHeight ;

    public Point m_startPoint = new Point() ;
    public Point m_startPointMatrixB = new Point() ;
    public Point m_startPointMatrixC = new Point() ;

    int m_numRowMatrixBTextBox ;
    int m_numColMatrixBTextBox ;

    int m_FormLength ;
    int m_FormHeight ;
    int m_numRowMatriCTextBox ;
    int m_numColMatriCTextBox ;

    /* *********************************** */
    public Form1()
    {
      InitializeComponent();

      m_startPoint.X = 16 ;
      m_startPoint.Y = 202 ;

      m_numRowMatricATextBox = 5 ;
      m_numColMatricATextBox = 7 ;

      m_numRowMatrixBTextBox = m_numColMatricATextBox ;
      m_numColMatrixBTextBox = 4 ;

      m_numRowMatriCTextBox = m_numRowMatricATextBox ;
      m_numColMatriCTextBox = m_numColMatrixBTextBox ;

      m_tbLength = 38 ;
      m_tbHeight = 20 ;

      int adistance = 50 ;
      // = length of A + a distance
      m_startPointMatrixB.X = m_startPoint.X + (m_numColMatricATextBox)*(m_tbLength+2)
                                                                     - 2 + adistance ;
      m_startPointMatrixB.Y = m_startPoint.Y ;

      int lbMatrixBlength = (m_numColMatrixBTextBox)*(m_tbLength+2) - 2;
      m_FormLength = m_startPointMatrixB.X + lbMatrixBlength + adistance ;
      m_FormHeight = 500 ;

      arrTextBoxMatrixA   = new TextBox[m_numRowMatricATextBox, m_numColMatricATextBox] ;
      db_MatrixA      = new double [m_numRowMatricATextBox, m_numColMatricATextBox] ;
      arrTextBoxMatrixB   = new TextBox[m_numRowMatrixBTextBox, m_numColMatrixBTextBox] ;
      db_MatrixB      = new double [m_numRowMatrixBTextBox, m_numColMatrixBTextBox] ;

      arrTextBoxMatrixC   = new TextBox[m_numRowMatriCTextBox, m_numColMatriCTextBox] ;
      db_MatrixC      = new double [m_numRowMatriCTextBox, m_numColMatriCTextBox] ;

      SetProperty() ;
    }

    /* ************************************ */
    public void SetTextBoxMatrixA()
    {
      for(int i=0; i<m_numRowMatricATextBox; i++)
      {
        for(int j=0; j<m_numColMatricATextBox; j++)
        {
          arrTextBoxMatrixA[i,j] = new System.Windows.Forms.TextBox();
          arrTextBoxMatrixA[i,j].Location = new System.Drawing.Point(
                 m_startPoint.X + (m_tbLength+2)*j, m_startPoint.Y + (m_tbHeight+2)*i );
          arrTextBoxMatrixA[i,j].Name = "textBox1";
          arrTextBoxMatrixA[i,j].Size = new System.Drawing.Size(m_tbLength, m_tbHeight);
          arrTextBoxMatrixA[i,j].TabIndex = i;
          arrTextBoxMatrixA[i,j].Text = "0.0";
          this.Controls.Add(arrTextBoxMatrixA[i,j]) ;
        }
      }
    }

    /* ************************************ */
    public void SetLabelMatrixA()
    {
      Label lbMatrixA = new System.Windows.Forms.Label();

      int lbMatrixAlength = (m_numColMatricATextBox)*(m_tbLength+2) - 2;
      lbMatrixA.Font = new System.Drawing.Font("Microsoft Sans Serif", 9.75F,
            System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
                           ((System.Byte)(0)));
      lbMatrixA.Location = new System.Drawing.Point(m_startPoint.X, m_startPoint.Y - 40);
      lbMatrixA.Size = new System.Drawing.Size(lbMatrixAlength, 28);
      lbMatrixA.Name = "lbMatrixA";
      lbMatrixA.Text = "Matrix A";
      lbMatrixA.TextAlign = System.Drawing.ContentAlignment.MiddleCenter;

      this.Controls.Add(lbMatrixA) ;
    }

    /* *********************************** */
    public void SetLabelMatrixB()
    {
      Label lbMatrixB = new System.Windows.Forms.Label();
      lbMatrixB.Font = new System.Drawing.Font("Microsoft Sans Serif", 9.75F,
        System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
                                                        ((System.Byte)(0)));

      // row B = col A
      int lbMatrixBlength = (m_numColMatrixBTextBox)*(m_tbLength+2) - 2; ;
      lbMatrixB.Location = new System.Drawing.Point(m_startPointMatrixB.X,
                                                    m_startPoint.Y - 40);
      lbMatrixB.Size = new System.Drawing.Size(lbMatrixBlength, 28);
      lbMatrixB.Name = "lbMatrixB";
      lbMatrixB.Text = "Matrix B";

      lbMatrixB.TextAlign = System.Drawing.ContentAlignment.MiddleCenter;
      this.Controls.Add(lbMatrixB) ;
    }

    /* *********************************** */
    public void SetLabelMatrixC()
    {
      Label lbMatrixC = new System.Windows.Forms.Label();
      lbMatrixC.Font  = new System.Drawing.Font("Microsoft Sans Serif", 9.75F,
        System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((System.Byte)(0)));

      // row B = col A
      int lbMatrixClength = (m_numColMatriCTextBox)*(m_tbLength+2) - 2; ;
      lbMatrixC.Location = new System.Drawing.Point(
                      m_startPointMatrixC.X, m_startPointMatrixC.Y - 40);
      lbMatrixC.Size = new System.Drawing.Size(lbMatrixClength, 28);
      lbMatrixC.Name = "lbMatrixC";
      lbMatrixC.Text = "Matrix C";

      lbMatrixC.TextAlign = System.Drawing.ContentAlignment.MiddleCenter;
      this.Controls.Add(lbMatrixC) ;
    }

    /* *********************************** */
    public void SetTextBoxMatrixB()
    {
      for(int i=0; i<m_numRowMatrixBTextBox; i++)
      {
        for(int j=0; j<m_numColMatrixBTextBox; j++)
        {
          arrTextBoxMatrixB[i,j] = new System.Windows.Forms.TextBox();
          arrTextBoxMatrixB[i,j].Location = new System.Drawing.Point(
           m_startPointMatrixB.X + (m_tbLength+2)*j, m_startPointMatrixB.Y + (m_tbHeight+2)*i );
          arrTextBoxMatrixB[i,j].Name = "textBox2";
          arrTextBoxMatrixB[i,j].Size = new System.Drawing.Size(m_tbLength, m_tbHeight);
          arrTextBoxMatrixB[i,j].TabIndex = i;
          arrTextBoxMatrixB[i,j].Text = "0.0";
          this.Controls.Add(arrTextBoxMatrixB[i,j]) ;
        }
      }
    }

    /* *********************************** */
    public void SetTextBoxMatrixC()
    {
      for(int i=0; i<m_numRowMatriCTextBox; i++)
      {
        for(int j=0; j<m_numColMatriCTextBox; j++)
        {
          arrTextBoxMatrixC[i,j] = new System.Windows.Forms.TextBox();
          arrTextBoxMatrixC[i,j].Location = new System.Drawing.Point(m_startPointMatrixC.X
                           + (m_tbLength+2)*j, m_startPointMatrixC.Y + (m_tbHeight+2)*i );
          arrTextBoxMatrixC[i,j].Name = "textBox2";
          arrTextBoxMatrixC[i,j].Size = new System.Drawing.Size(m_tbLength, m_tbHeight);
          arrTextBoxMatrixC[i,j].TabIndex = i;
          arrTextBoxMatrixC[i,j].Text = "0.0";
          arrTextBoxMatrixC[i,j].ReadOnly = true;
          this.Controls.Add(arrTextBoxMatrixC[i,j]) ;
        }
      }
    }

    /* ************************************ */
    private void btSetNumber_Click(object sender, System.EventArgs e)
    {
      int rowA, colA ;
      int rowB, colB ;

      try
      {
        rowA = Convert.ToInt32(tbMatrixA_row.Text);
        colA = Convert.ToInt32(tbMatrixA_col.Text);

        rowB = Convert.ToInt32(tbMatrixB_row.Text);
        colB = Convert.ToInt32(tbMatrixB_col.Text);
      }
      catch
      {
        MessageBox.Show("Please enter the number") ;
        return ;
      }
      m_numRowMatricATextBox = rowA ;
      m_numColMatricATextBox = colA ;

      m_numRowMatrixBTextBox = m_numColMatricATextBox ;
      m_numColMatrixBTextBox = colB ;

      m_numRowMatriCTextBox = m_numRowMatricATextBox ;
      m_numColMatriCTextBox = m_numColMatrixBTextBox ;

      try
      {
        SetProperty();
      }
      catch {
        MessageBox.Show(@"This is an example for matrix multiplication C = A*B, with
        max of A = 7x5 and B = 5x4");
      } ;

    }

    /* *********************************** */
    public void SetProperty()
    {
      this.Controls.Clear() ;
      m_tbLength = 38 ;
      m_tbHeight = 20 ;

      int adistance = 50 ;
      // = length of A + a distance
      m_startPointMatrixB.X = m_startPoint.X + (m_numColMatricATextBox)*(m_tbLength+2)
                                                                         - 2 + adistance ;
      m_startPointMatrixB.Y = m_startPoint.Y ;

      int lbMatrixBlength = (m_numColMatrixBTextBox)*(m_tbLength+2) - 2;
```

```
m_startPointMatrixC.X = m_startPoint.X ;
int aheight_A = m_startPoint.Y + (m_numRowMatricATextBox)*(m_tbHeight+2)+ 3*adistance ;
int aheight_B = m_startPoint.Y + (m_numRowMatrixBTextBox)*(m_tbHeight+2)+ 3*adistance ;

int aheight = Math.Max(aheight_A, aheight_B) ;
m_startPointMatrixC.Y = aheight ;

// reset form size
InitializeComponent();
tbMatrixA_row.Text = m_numRowMatricATextBox.ToString() ;
tbMatrixA_col.Text = m_numColMatricATextBox.ToString() ;

tbMatrixB_row.Text = m_numRowMatrixBTextBox.ToString() ;
tbMatrixB_col.Text = m_numColMatrixBTextBox.ToString() ;

m_numRowMatriCTextBox = m_numRowMatricATextBox ;
m_numColMatriCTextBox = m_numColMatrixBTextBox ;

SetLabelMatrixA() ;
SetLabelMatrixB() ;
SetLabelMatrixC() ;

// set calculation button
//this.btCalculation.Location = new System.Drawing.Point(m_startPoint.X, aheight - 100);

SetTextBoxMatrixA() ;
SetTextBoxMatrixB() ;
SetTextBoxMatrixC() ;

m_FormLength = m_startPointMatrixB.X + lbMatrixBlength + adistance ;
m_FormHeight = m_startPointMatrixC.Y + m_numRowMatriCTextBox*(m_tbHeight+2) + adistance;

m_FormLength = Math.Max(544, m_FormLength) ;
this.ClientSize = new System.Drawing.Size(m_FormLength, m_FormHeight);

this.Refresh() ;
```

```
}

/* ************************************ */
private void btCalculation_Click(object sender, System.EventArgs e)
{
  ////////// get matrix A /////////////
  try
  {
    GetMatrixA() ;
  }
  catch
  {
    MessageBox.Show("Data in Matrix A is not valid") ;
    return ;
  }

  ////////// get matrix B /////////////
  try
  {
    GetMatrixB() ;
  }
  catch
  {
    MessageBox.Show("Data in Matrix B is not valid") ;
    return ;
  }



  this.labelCalcInfo.Text = @"Calculation is in process. Please wait.";
  this.Invalidate() ;
  this.Refresh() ;

  /////////////// calculate matrix C ////////////
  GetMatrixC() ;

  labelCalcInfo.Text = @"Calculation has done";
```

```
}

/* *********************************** */
public void GetMatrixA()
{
  int rowA, colA ;

  rowA = Convert.ToInt32(tbMatrixA_row.Text);
  colA = Convert.ToInt32(tbMatrixA_col.Text);
  int i, j ;

  for( i=0; i<m_numRowMatricATextBox; i++)
  {
    for( j=0; j<m_numColMatricATextBox; j++)
    {
      db_MatrixA[i,j] = Convert.ToDouble(arrTextBoxMatrixA[i,j].Text) ;
    }
  }

}

/* *********************************** */
public void GetMatrixB()
{
  int rowB, colB ;
  rowB = Convert.ToInt32(tbMatrixB_row.Text);
  colB = Convert.ToInt32(tbMatrixB_col.Text);

  int i, j ;

  for( i=0; i<m_numRowMatrixBTextBox; i++)
  {
    for( j=0; j<m_numColMatrixBTextBox; j++)
    {
      db_MatrixB[i,j] = Convert.ToDouble(arrTextBoxMatrixB[i,j].Text) ;
    }
  }
}

/* ************************************ */
public void GetMatrixC()
{
  // calculate matrix multiplication
  MWNumericArray mw_A = new MWNumericArray(db_MatrixA) ;
  MWNumericArray mw_B = new MWNumericArray(db_MatrixB) ;
  MWNumericArray mw_C = null ;

  /* call an implemental function */
  MatrixMultiplication objMatlab = new MatrixMultiplication() ;
  mw_C = (MWNumericArray) objMatlab.mymtimes(mw_A, mw_B) ;

  /* convert back to C# double  */
  db_MatrixC = (double[,]) mw_C.ToArray(MWArrayComponent.Real) ;

  // display to GUI
  int i, j ;
  for( i=0; i<m_numRowMatriCTextBox; i++)
  {
    for( j=0; j<m_numColMatriCTextBox; j++)
    {
      arrTextBoxMatrixC[i,j].Text = db_MatrixC[i,j].ToString() ;
    }
  }

  mw_A.Dispose() ;
  mw_B.Dispose() ;
  mw_C.Dispose() ;
}

/* ************************************ */
private void tbMatrixA_col_TextChanged(object sender, System.EventArgs e)
{
      tbMatrixB_row.Text = tbMatrixA_col.Text ;
    }

/* ************************************ */
/* ************************************ */
/* ************************************ */
  }
}
```

end code

Part III:

Plotting MATLAB Graphics Figures In C#

Chapter 15

Using MATLAB Graphics In C$^{\#}$ Functions

In this chapter we'll generate a class ***GeneratingGraphics*** from three common M-files to plot MATLAB 2D Graphics figures. The generated functions of this class will be used in MSVC# .Net version 2008 project to plot 2D figures.

The procedure to create the class ***GeneratingGraphics*** is the same as the procedure in Chapter 2. We will write the M-files as shown below to generate that class.

simplePlot.m

```
function simplePlot(x, y, strColor)

plot(x, y, strColor) ;

grid on ;
```

multiPlotForArrays.m

```
function multiPlotForArrays(t, y1, y2, y3, y4, ...
                                  strTitle, strxlabel, strylabel )

hold on ;
```

```
Hplot1 =  plot (t, y1) ;
Hplot2 =  plot (t, y2) ;
Hplot3 =  plot (t, y3) ;
Hplot4 =  plot (t, y4) ;

set(Hplot1, 'Color', 'r', 'Marker', '*', 'LineStyle', '-' )  ;
set(Hplot2, 'Color', 'b', 'Marker', '^', 'LineStyle', ':' )  ;
set(Hplot3, 'Color', 'k', 'Marker', 'o', 'LineStyle', '--')  ;
set(Hplot4, 'Color', 'g', 'Marker', 'o', 'LineStyle', '--')  ;

string1 = 'plot 1' ;
string2 = 'plot 2' ;
string3 = 'plot 3' ;
string4 = 'plot 4' ;

pos = 'SouthEast' ;

Hlegend = legend(string1, string2, string3, string4, 'Location', pos) ;
set(Hlegend, 'FontWeight', 'bold') ;
legend('boxon') ;

hold off ;

%title('Figure Legends') ;
 title(strTitle) ;

%xlabel('x') ;
 xlabel(strxlabel) ;

%ylabel('y') ;
 ylabel(strylabel) ;
```

```
grid on ;
```

multiPlotForFuncs.m

```
function multiPlotForFuncs(strPlot1, colorPlot1, markerPlot1, ...
                          strPlot2, colorPlot2, markerPlot2, ...
                          strPlot3, colorPlot3, markerPlot3, ...
                          strTitle, strxlabel , strylabel )

t = linspace(0, 2*pi, 20) ;

% y1 = cos(t)           ;
% y2 = cos(t + pi/3)    ;
% y3 = cos(t + 2*pi/3)  ;

f1 = inline(strPlot1)   ;
y1 = feval( f1, t )     ;

f2 = inline(strPlot2)   ;
y2 = feval( f2, t )     ;

f3 = inline(strPlot3)   ;
y3 =  feval( f3, t )    ;

hold on ;

Hplot1 =  plot (t, y1) ;
Hplot2 =  plot (t, y2) ;
Hplot3 =  plot (t, y3) ;

% set(Hplot1, ’Color’, ’r’, ’Marker’, ’*’, ’LineStyle’, ’-’)    ;
% set(Hplot2, ’Color’, ’b’, ’Marker’, ’^’, ’LineStyle’, ’none’) ;
% set(Hplot3, ’Color’, ’k’, ’Marker’, ’o’, ’LineStyle’, ’--’)   ;
%
```

```
 set(Hplot1, 'Color', colorPlot1, 'Marker', markerPlot1, 'LineStyle', '-')      ;
 set(Hplot2, 'Color', colorPlot2, 'Marker', markerPlot2, 'LineStyle', 'none')   ;
 set(Hplot3, 'Color', colorPlot3, 'Marker', markerPlot3, 'LineStyle', '--')     ;

string1 = 'plot 1' ;
string2 = 'plot 2' ;
string3 = 'plot 3' ;

pos = 'SouthEast' ;

Hlegend = legend(string1, string2, string3, 'Location', pos) ;
set(Hlegend, 'FontWeight', 'bold') ;
legend('boxon') ;

hold off ;

%title('Figure Legends') ;
 title(strTitle) ;

%xlabel('x') ;
 xlabel(strxlabel) ;

%ylabel('y') ;
 ylabel(strylabel) ;

grid on ;
```

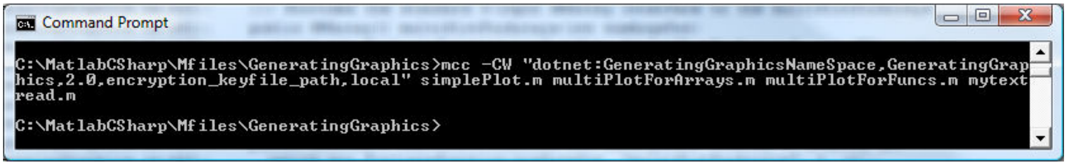
mytextread.m

```
function varargout = mytextread(fileName, colNum, mydelimiter)

  switch colNum
  case 2
  [A1, A2] = textread(fileName, '%f%f','delimiter', mydelimiter) ;
  varargout{1} = A1 ; varargout{2} = A2 ;

  case 3
  [A1, A2, A3] = textread(fileName, '%f%f%f', 'delimiter', mydelimiter) ;
  varargout{1} = A1 ; varargout{2} = A2 ; varargout{3} = A3 ;

  case 4
  [A1, A2, A3, A4] = textread(fileName, '%f%f%f%f', 'delimiter', mydelimiter) ;
  varargout{1} = A1 ; varargout{2} = A2 ; varargout{3} = A3 ;
  varargout{4} = A4 ;

  case 5
  [A1, A2, A3, A4, A5] = textread(fileName, '%f%f%f%f%f', 'delimiter', mydelimiter) ;
  varargout{1} = A1 ; varargout{2} = A2 ; varargout{3} = A3 ;
  varargout{4} = A4 ; varargout{5} = A5 ;

  otherwise
      disp('This function reads files with max colums = 5.');

end
```

The following procedure to create the class ***GeneratingGraphics*** is the same as the procedure in Chapter 2 as follows:

1. Write the following command in **Command Prompt** to generate the name space ***GeneratingGraphicsNameSpace*** that contains the class ***GeneratingGraphics*** is (see Fig.15.1):

   ```
   mcc -CW "dotnet:GeneratingGraphicsNameSpace,GeneratingGraphics,2.0,
   encryption_keyfile_path,local" simplePlot.m multiPlotForArrays.m
   multiPlotForFuncs.m mytextread.m
   ```



```
C:\MatlabCSharp\Mfiles\GeneratingGraphics>mcc -CW "dotnet:GeneratingGraphicsNameSpace,GeneratingGrap
hics,2.0,encryption_keyfile_path,local" simplePlot.m multiPlotForArrays.m multiPlotForFuncs.m mytext
read.m

C:\MatlabCSharp\Mfiles\GeneratingGraphics>
```

Figure 15.1: Command of creating the class ***GeneratingGraphics***

2. Create a regular C# project in Windows Application of Microsoft Visual Studio 2008.

3. Click **Build**, then click **Rebuild** to make sure the program works.

4. In the project menu, click **Project**, **Add Reference**. Then the project will pop up a dialog to choose a reference.

5. Click on tab **.Net**, **MathWorks, .NET MWArray API**. Then the project will add MATLAB array wrapper classes for .NET, **MWArray** into the C# project.

6. Copy the two files `dotnet_mcc_component_data.cs` and `GeneratingGraphics.cs` from previous steps and put them into the same directory of the `Program.cs` file.

7. Copy the file `GeneratingGraphicsNameSpace.ctf` and put it into the same directory of the project executable file, `Example.exe`.

8. Add the two files to the C# project by click **Project** on the project menu, then click **Add Existing Item**. The project will pop up a dialog then choose the two files `dotnet_mcc_component_data.cs` and `GeneratingGraphics.cs`. This step will add these two files to the project.

9. At the top of the file `Program.cs`, add the following:

```
using MathWorks.MATLAB.NET.Arrays;
using MathWorks.MATLAB.NET.Utility;
using GeneratingGraphicsNameSpace;
```

15.1 Using MATLAB Graphics to Plot a Simple 2D Figure

Problem 1

Plot a figure from two arrays with black color :

```
X =   11.1,     22.2,       33.3  ;
Y =   110.1,    220.2,      330.3 ;
```

The code shown to solve Problem 1 is using the function `simplePlot.m`.

Listing code

```
public void SimplePlot()
{
    double[] db_vectorX =  { 11.1,  22.2 , 33.3  } ;
    double[] db_vectorY =  { 110.1, 220.2, 330.3 } ;

    String strColor = "k" ; // k: black, b: blue, r: red

    // Declare mwArray variables
    MWNumericArray mw_X = new MWNumericArray(db_vectorX) ;
    MWNumericArray mw_Y = new MWNumericArray(db_vectorY) ;

    // Set color, see color symbols in MATLAB Graphics
    MWCharArray mw_strColor  = new MWCharArray(strColor)      ;

    // Call the implemental function
    GeneratingGraphics objMatlab = new GeneratingGraphics() ;
    objMatlab.simplePlot(mw_X, mw_Y, mw_strColor);
    objMatlab.WaitForFiguresToDie() ; // program will continue after close this figure

    mw_strColor.Dispose() ;
    mw_X.Dispose() ;
    mw_Y.Dispose() ;
}
```

end code

The program creates a figure as shown in Fig. 15.2.

The following code shows another simple plot that uses the function `simplePlot.m`.

Listing code

```
public void AnotherSimplePlot(double[] arrayX, double[] arrayY)
```

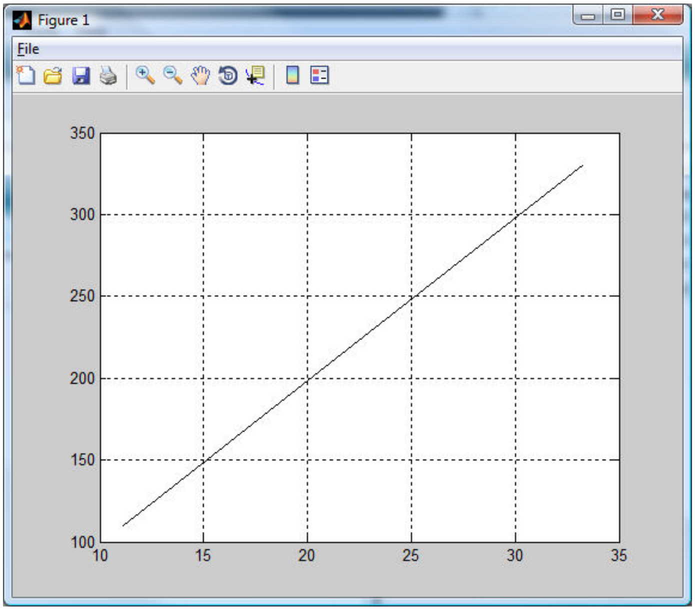


Figure 15.2: A simple plot

```
{
    String strColor = "b" ; // k: black, b: blue, r: red

    // Declare mwArray variables
    MWNumericArray mw_X = new MWNumericArray(arrayX) ;
    MWNumericArray mw_Y = new MWNumericArray(arrayY) ;

    // Set color, see color symbols in MATLAB Graphics
    MWCharArray mw_strColor  = new MWCharArray(strColor)      ;

    // Call the implemental function
    GeneratingGraphics objMatlab = new GeneratingGraphics() ;
    objMatlab.simplePlot(mw_X, mw_Y, mw_strColor);
    objMatlab.WaitForFiguresToDie() ; // program will continue after close this figure

    mw_strColor.Dispose() ;
    mw_X.Dispose() ;
    mw_Y.Dispose() ;
}
```

end code

15.2 Using MATLAB Graphics to Plot a 2D Figure with Data From a File

Problem 2

Plot a figure with red color from two columns of a data file **PlottingFile.txt** as shown below (this file **PlottingFile.txt** is put in the directory `.\Example\Example\bin\Debug`).

```
1.1   100.1
2.2   200.2
3.3   300.3
4.4   400.4
5.5   500.5
6.6   600.6
7.7   700.7
8.8   800.8
9.9   900.9
```

The following shows the code to solve Problem 2 by using the MATLAB M-files functions `mytextread.m` and `simplePlot.m` (shown at the beginning of this chapter).

The program creates a figure as shown in Fig. 15.3.

Listing code

```
public void PlotDataFromFile()
{
    // Reading data from a file
    const string fileName = "PlottingFile.txt";

    MWArray [] mw_ArrayOut = null ;
    MWNumericArray mw_X = null ;
    MWNumericArray mw_Y = null ;
```

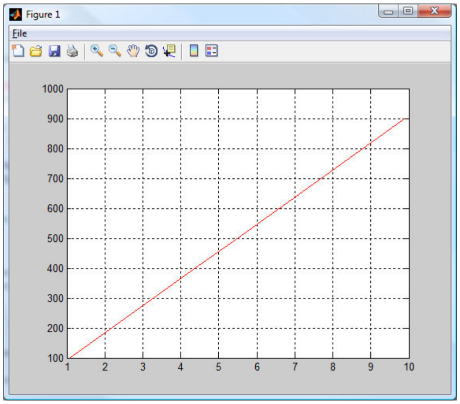


Figure 15.3: Figure from a data file

```
GeneratingGraphics objMatlab = new GeneratingGraphics() ;
String delimiter = @" " ; // blank
int numCol = 2 ;

// Value of the first 2 is two of ouput of varargout in mytextread.m
mw_ArrayOut = objMatlab.mytextread(2, fileName, numCol, delimiter) ;

mw_X = (MWNumericArray) mw_ArrayOut[0] ;
mw_Y = (MWNumericArray) mw_ArrayOut[1] ;

// End of Reading data from a file

// Set color, see color symbols in MATLAB Graphics
// Plot graphic
String strColor = "r" ; // k: black, b: blue, r: red
MWCharArray mw_strColor  = new MWCharArray(strColor) ;
objMatlab.simplePlot(mw_X, mw_Y, mw_strColor);

objMatlab.WaitForFiguresToDie() ; // program will continue after close this figure

mw_strColor.Dispose() ;
MWNumericArray.DisposeArray(mw_ArrayOut) ;
mw_X.Dispose() ;
```

```
    mw_Y.Dispose() ;
}
```

end code

15.3 Using MATLAB Graphics 2D to Plot Multiple Figures from Mathematical Functions

Similar to the previous section, in this section we'll generate multiple figures with data from mathematical functions by using the MATLAB M-file `multiPlotForFuncs.m` (shown at the beginning of this chapter). In this function, we particularly plot three curves. If you would like to plot with another number of curves you should modify this function.

Problem 3

Plot figures from the following functions and their properties:

```
Function 1 = cos(t)
Function 2 = cos(t + pi/3)
Function 3 = cos(t + 2*pi/3)

marker of Plot 1 = *
marker of Plot 2 = ^
marker of Plot 3 = o

color of Plot 1 =  red
color of Plot 2 =  blue
color of Plot 3 =  black

Title   = Figure Legends
x label = My x label
y label = My y label
```

The following shows the code to solve Problem 3 by using the function myplotmultiple2D(..). This program creates a figure as shown in Fig. 15.4

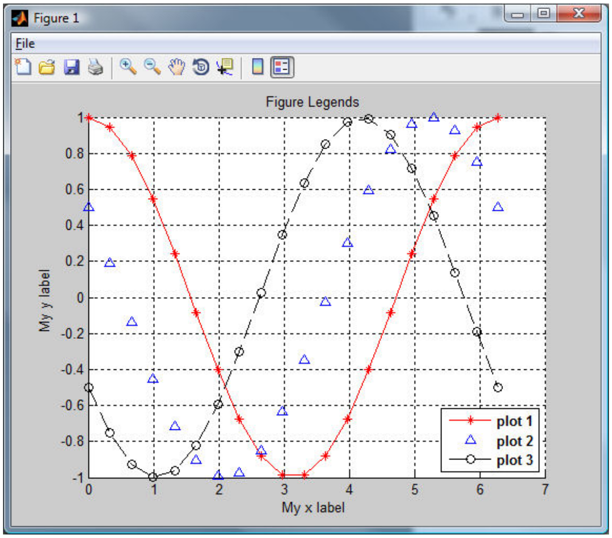


Figure 15.4: Multiple figures from math functions

Listing code

```
public void PlotFromFunctions() {

    String strfunc1 = "cos(t)"          ;
    String strfunc2 = "cos(t + pi/3)"   ;
    String strfunc3 = "cos(t + 2*pi/3)" ;

    MWCharArray  strPlot1 = new MWCharArray(strfunc1)   ;
    MWCharArray  strPlot2 = new MWCharArray(strfunc2)   ;
    MWCharArray  strPlot3 = new MWCharArray(strfunc3)   ;

    MWCharArray markerPlot1 = new MWCharArray("*") ;
    MWCharArray markerPlot2 = new MWCharArray("^") ;
    MWCharArray markerPlot3 = new MWCharArray("o") ;

    MWCharArray colorPlot1 = new MWCharArray("r") ; // red
    MWCharArray colorPlot2 = new MWCharArray("b") ; // blue
    MWCharArray colorPlot3 = new MWCharArray("k") ; // black
    MWCharArray  strTitle  = new MWCharArray("Figure Legends "  ) ;
    MWCharArray  strxlabel = new MWCharArray("My x label "      ) ;
    MWCharArray  strylabel = new MWCharArray("My y label "      ) ;

    // Call the implemental function
    GeneratingGraphics objMatlab = new GeneratingGraphics();
    objMatlab.multiPlotForFuncs(strPlot1, colorPlot1, markerPlot1,
                                strPlot2, colorPlot2, markerPlot2,
                                strPlot3, colorPlot3, markerPlot3,
                                strTitle, strxlabel , strylabel) ;

    objMatlab.WaitForFiguresToDie() ; // program will continue after close this figure

    strPlot1.Dispose()      ;
    strPlot2.Dispose()      ;
    strPlot3.Dispose()      ;

    markerPlot1.Dispose()   ;
    markerPlot2.Dispose()   ;
    markerPlot3.Dispose()   ;

    colorPlot1.Dispose()    ;
    colorPlot2.Dispose()    ;
    colorPlot3.Dispose()    ;

    strTitle.Dispose()      ;
    strxlabel.Dispose()     ;
    strylabel.Dispose()     ;
}
```

end code

15.4 Using MATLAB Graphics to Plot Multiple Figures with Data from Arrays

Similar to the previous section, in this section we will generate multiple figures with data from arrays.

Problem 4

Plot multiple figures from the following arrays and their properties:

```
double db_vectort[3] =  { 0.1,      0.2,   0.3   } ;


double db_vectorY1[3] =  { 11.1,    22.2,  33.3  } ;
double db_vectorY2[3] =  { 40.1,    50.2,  60.3  } ;
double db_vectorY3[3] =  { 70.1,    80.2,  90.3  } ;
double db_vectorY4[3] =  { 100.1,   110.2, 120.3 } ;


Title   = The Title
x label = time
y label = voltage
```

The following shows the code to solve Problem 3 by using the function myplotmultiple2DFromArray(..). The program creates a figure as shown in Fig. 15.5.

Listing code

```
public void MultiPlotFromArrays()
{
    double[] db_vectort =   {   0.1,   0.2,  0.3  } ;

    double[] db_vectorY1 =  {  11.1,  22.2, 33.3  } ;
    double[] db_vectorY2 =  {  40.1,  50.2, 60.3  } ;
    double[] db_vectorY3 =  {  70.1,  80.2, 90.3  } ;
    double[] db_vectorY4 =  { 100.1, 110.2, 120.3 } ;

    // Declare mwArray variables
```

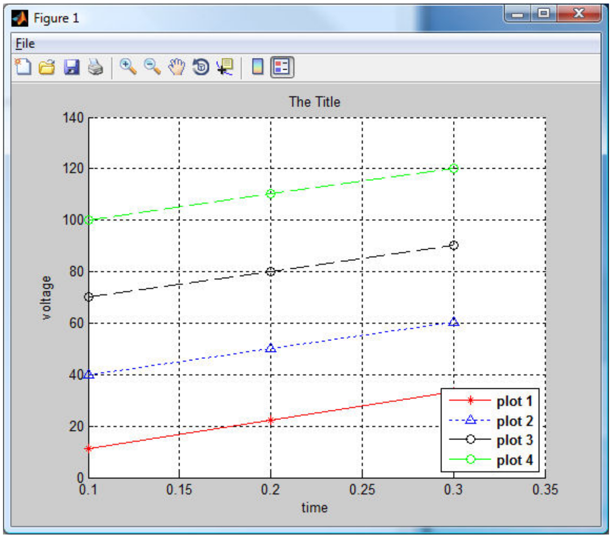


Figure 15.5: Multiple figures from arrays

```
MWNumericArray mw_t  = new MWNumericArray(db_vectort ) ;

MWNumericArray mw_y1 = new MWNumericArray(db_vectorY1) ;
MWNumericArray mw_y2 = new MWNumericArray(db_vectorY2) ;
MWNumericArray mw_y3 = new MWNumericArray(db_vectorY3) ;
MWNumericArray mw_y4 = new MWNumericArray(db_vectorY4) ;

// Set properties for MATLAB Graphics
MWCharArray mw_Title  = new MWCharArray("The Title" ) ;
MWCharArray mw_xlabel = new MWCharArray("time"      ) ;
MWCharArray mw_ylabel = new MWCharArray("voltage"   ) ;

// Call the implemental function
GeneratingGraphics objMatlab = new GeneratingGraphics() ;
objMatlab.multiPlotForArrays(mw_t, mw_y1, mw_y2, mw_y3, mw_y4,
                          mw_Title, mw_xlabel, mw_ylabel ) ;

objMatlab.WaitForFiguresToDie() ; // program will continue after close this figure

mw_t.Dispose() ;
```

```
    mw_y1.Dispose() ;
    mw_y2.Dispose() ;
    mw_y3.Dispose() ;
    mw_y4.Dispose() ;

    mw_Title.Dispose()  ;
    mw_xlabel.Dispose() ;
    mw_ylabel.Dispose() ;
}
```

end code

15.5 Using MATLAB Graphics to Plot Multiple Figures with Data from a File

Problem 5

Plot multiple figures from a data file MultiArrays.csv and their properties as follows:

```
0.1,    1,  2,  4,      8
0.2,    2,  4,  8,      16
0.3,    4,  8,  16,     32
0.4,    8,  16, 32,     64
0.5,    16, 32, 64,     28
0.6,    32, 64, 128,    256
```

The following shows the code to solve Problem 5 by using the function MultiPlotFromFile(..). The program creates a figure as shown in Fig. 15.6.

Listing code

```
public void MultiPlotFromFile()
{
    // Read data from a file
    const string fileName = "MultiArrays.csv";

    MWArray [] mw_ArrayOut = null ;
```

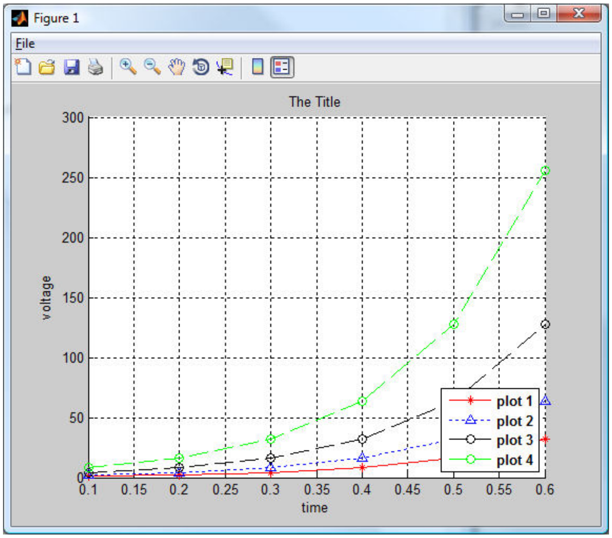


Figure 15.6: Multiple figures from a data file

```
MWNumericArray mw_Col0 = null ;
MWNumericArray mw_Col1 = null ;
MWNumericArray mw_Col2 = null ;
MWNumericArray mw_Col3 = null ;
MWNumericArray mw_Col4 = null ;

GeneratingGraphics objMatlab = new GeneratingGraphics() ;
String delimiter = @"," ; // comma
int numCol = 5 ;

// Value of the first 5 is five of ouput of varargout in mytextread.m
mw_ArrayOut = objMatlab.mytextread(5, fileName, numCol, delimiter) ;

mw_Col0 = (MWNumericArray) mw_ArrayOut[0] ;
mw_Col1 = (MWNumericArray) mw_ArrayOut[1] ;
mw_Col2 = (MWNumericArray) mw_ArrayOut[2] ;
mw_Col3 = (MWNumericArray) mw_ArrayOut[3] ;
mw_Col4 = (MWNumericArray) mw_ArrayOut[4] ;

// End of Reading data from a file
```

```
    //Plot figures
    // Set properties for MATLAB Graphics
    MWCharArray mw_Title  = new MWCharArray("The Title");
    MWCharArray mw_xlabel = new MWCharArray("time");
    MWCharArray mw_ylabel = new MWCharArray("voltage");

    // Call the implemental function
    objMatlab.multiPlotForArrays(mw_Col0, mw_Col1, mw_Col2, mw_Col3, mw_Col4,
                                 mw_Title, mw_xlabel, mw_ylabel);

    objMatlab.WaitForFiguresToDie() ; // program will continue after close this figure

    MWNumericArray.DisposeArray(mw_ArrayOut) ;
    mw_Col0.Dispose();
    mw_Col1.Dispose();
    mw_Col2.Dispose();
    mw_Col3.Dispose();
    mw_Col4.Dispose();

    mw_Title.Dispose();
    mw_xlabel.Dispose();
    mw_ylabel.Dispose();
}
```

end code

The following is the full code for this chapter.

Listing code

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO ;

using MathWorks.MATLAB.NET.Arrays;
using MathWorks.MATLAB.NET.Utility;
using GeneratingGraphicsNameSpace;

namespace Example
{
  class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine(" Using the plot function. Please wait. ") ;
      Program objProgram = new Program() ;

      // Prolem 1: Plot a simple 2D figure
      //objProgram.SimplePlot() ;



      // Prolem : Plot a 2D figure by passing arrays
      /*
      double[]X = {0, 50, 100, 150, 200, 250, 300, 350, 400, 450, 500, 550, 600} ;

      double[]Y = {0.4000, 0.2426, 0.1472, 0.0893, 0.0541, 0.0328, 0.0199,
                   0.0121, 0.0073, 0.0044, 0.0027, 0.0016,  0.0010 } ;

      objProgram.AnotherSimplePlot(X, Y) ;
      */

      /*
      // Prolem 2: Plot a 2D figure with data from a file
      objProgram.PlotDataFromFile() ;
      */

      /*
      // Problem 3: Plot multiple figures with data from math functions
       objProgram.PlotFromFunctions() ;
      */
      /*
      // Problem 4: Plot multiple figures with data from arrays
      objProgram.MultiPlotFromArrays() ;
      */

      // Problem 5: Plot multiple figures with data from a file
      objProgram.MultiPlotFromFile() ;
    }

    /* **************************** */
    public void SimplePlot()
    {
      double[] db_vectorX =  { 11.1, 22.2 , 33.3  } ;
      double[] db_vectorY =  { 110.1, 220.2, 330.3 } ;

      String strColor = "k" ; // k: black, b: blue, r: red

      // Declare mwArray variables
      MWNumericArray mw_X = new MWNumericArray(db_vectorX) ;
      MWNumericArray mw_Y = new MWNumericArray(db_vectorY) ;

      // Set color, see color symbols in MATLAB Graphics
      MWCharArray mw_strColor  = new MWCharArray(strColor)      ;

    // Call the implemental function
      GeneratingGraphics objMatlab = new GeneratingGraphics() ;
      objMatlab.simplePlot(mw_X, mw_Y, mw_strColor);
      objMatlab.WaitForFiguresToDie() ; // program will continue after close this figure

      /*
      Typically you use WaitForFiguresToDie when:
      . There are one or more figures open that were created
        by a .NET component created by the builder.
      . The method that displays the graphics requires user input before continuing.
      . The method that calls the figures was called from main() in a console program.
      When WaitForFiguresToDie is called, execution of the calling program is blocked
      if any figures created by the calling object remain open.
      */

      mw_strColor.Dispose() ;
      mw_X.Dispose() ;
      mw_Y.Dispose() ;
    }

    /* **************************** */
    public void AnotherSimplePlot(double[] arrayX, double[] arrayY)
    {
      String strColor = "b" ; // k: black, b: blue, r: red

      // Declare mwArray variables
      MWNumericArray mw_X = new MWNumericArray(arrayX) ;
      MWNumericArray mw_Y = new MWNumericArray(arrayY) ;

      // Set color, see color symbols in MATLAB Graphics
      MWCharArray mw_strColor  = new MWCharArray(strColor)      ;

    // Call the implemental function
      GeneratingGraphics objMatlab = new GeneratingGraphics() ;
      objMatlab.simplePlot(mw_X, mw_Y, mw_strColor);
      objMatlab.WaitForFiguresToDie() ; // program will continue after close this figure

      mw_strColor.Dispose() ;
      mw_X.Dispose() ;
      mw_Y.Dispose() ;
    }

    /* **************************** */
    public void PlotDataFromFile()
    {
      // Reading data from a file
      const string fileName = "PlottingFile.txt";
      MWArray [] mw_ArrayOut = null ;
      MWNumericArray mw_X = null ;
      MWNumericArray mw_Y = null ;

      GeneratingGraphics objMatlab = new GeneratingGraphics() ;
      String delimiter = @" " ; // blank
      int numCol = 2 ;

      // Value of the first 2 is two of ouput of varargout in mytextread.m
      mw_ArrayOut = objMatlab.mytextread(2, fileName, numCol, delimiter) ;

      mw_X = (MWNumericArray) mw_ArrayOut[0] ;
      mw_Y = (MWNumericArray) mw_ArrayOut[1] ;

      // End of Reading data from a file

      // Set color, see color symbols in MATLAB Graphics
      // Plot graphic
      String strColor = "r" ; // k: black, b: blue, r: red
      MWCharArray mw_strColor  = new MWCharArray(strColor) ;
      objMatlab.simplePlot(mw_X, mw_Y, mw_strColor);

      objMatlab.WaitForFiguresToDie() ; // program will continue after close this figure

      mw_strColor.Dispose() ;
      MWNumericArray.DisposeArray(mw_ArrayOut) ;
      mw_X.Dispose() ;
      mw_Y.Dispose() ;
    }

    /* *************************** */
    public void PlotFromFunctions() {

      String strfunc1 = "cos(t)"          ;
      String strfunc2 = "cos(t + pi/3)"   ;
      String strfunc3 = "cos(t + 2*pi/3)" ;

      MWCharArray  strPlot1 = new MWCharArray(strfunc1) ;
      MWCharArray  strPlot2 = new MWCharArray(strfunc2) ;
      MWCharArray  strPlot3 = new MWCharArray(strfunc3) ;

      MWCharArray markerPlot1 = new MWCharArray("*") ;
      MWCharArray markerPlot2 = new MWCharArray("^") ;
      MWCharArray markerPlot3 = new MWCharArray("o") ;

      MWCharArray colorPlot1 = new MWCharArray("r") ; // red
      MWCharArray colorPlot2 = new MWCharArray("b") ; // blue
      MWCharArray colorPlot3 = new MWCharArray("k") ; // black

      MWCharArray  strTitle  = new MWCharArray("Figure Legends "  ) ;
      MWCharArray  strxlabel = new MWCharArray("My x label "     ) ;
      MWCharArray  strylabel = new MWCharArray("My y label "     ) ;

      // Call the implemental function
      GeneratingGraphics objMatlab = new GeneratingGraphics();
      objMatlab.multiPlotForFuncs(strPlot1, colorPlot1, markerPlot1,
                                  strPlot2, colorPlot2, markerPlot2,
                                  strPlot3, colorPlot3, markerPlot3,
                                  strTitle, strxlabel , strylabel) ;

      objMatlab.WaitForFiguresToDie() ; // program will continue after close this figure

      strPlot1.Dispose()      ;
      strPlot2.Dispose()      ;
      strPlot3.Dispose()      ;

      markerPlot1.Dispose()   ;
      markerPlot2.Dispose()   ;
      markerPlot3.Dispose()   ;

      colorPlot1.Dispose()    ;
```

```
      colorPlot2.Dispose()    ;
      colorPlot3.Dispose()    ;

      strTitle.Dispose()      ;
      strxlabel.Dispose()     ;
      strylabel.Dispose()     ;
    }

    /* **************************** */
    public void MultiPlotFromArrays()
    {
      double[] db_vectort =   {   0.1,   0.2,  0.3  } ;

      double[] db_vectorY1 =  {  11.1,  22.2, 33.3  } ;
      double[] db_vectorY2 =  {  40.1,  50.2, 60.3  } ;
      double[] db_vectorY3 =  {  70.1,  80.2, 90.3  } ;
      double[] db_vectorY4 =  { 100.1, 110.2, 120.3 } ;

      // Declare mwArray variables
      MWNumericArray mw_t  = new MWNumericArray(db_vectort ) ;

      MWNumericArray mw_y1 = new MWNumericArray(db_vectorY1) ;
      MWNumericArray mw_y2 = new MWNumericArray(db_vectorY2) ;
      MWNumericArray mw_y3 = new MWNumericArray(db_vectorY3) ;
      MWNumericArray mw_y4 = new MWNumericArray(db_vectorY4) ;

      // Set properties for MATLAB Graphics
      MWCharArray mw_Title  = new MWCharArray("The Title" ) ;
      MWCharArray mw_xlabel = new MWCharArray("time"     ) ;
      MWCharArray mw_ylabel = new MWCharArray("voltage" ) ;

      // Call the implemental function
      GeneratingGraphics objMatlab = new GeneratingGraphics() ;
      objMatlab.multiPlotForArrays(mw_t, mw_y1, mw_y2, mw_y3, mw_y4,
                       mw_Title, mw_xlabel, mw_ylabel ) ;
```

```
  objMatlab.WaitForFiguresToDie() ; // program will continue after close this figure

  mw_t.Dispose() ;

  mw_y1.Dispose() ;
  mw_y2.Dispose() ;
  mw_y3.Dispose() ;
  mw_y4.Dispose() ;

  mw_Title.Dispose()  ;
  mw_xlabel.Dispose() ;
  mw_ylabel.Dispose() ;
}

/* *************************** */
public void MultiPlotFromFile()
{
  // Read data from a file
  const string fileName = "MultiArrays.csv";

  MWArray [] mw_ArrayOut = null ;
  MWNumericArray mw_Col0 = null ;
  MWNumericArray mw_Col1 = null ;
  MWNumericArray mw_Col2 = null ;
  MWNumericArray mw_Col3 = null ;
  MWNumericArray mw_Col4 = null ;

  GeneratingGraphics objMatlab = new GeneratingGraphics() ;
  String delimiter = @"," ; // comma
  int numCol = 5 ;

  // Value of the first 5 is five of ouput of varargout in mytextread.m
  mw_ArrayOut = objMatlab.mytextread(5, fileName, numCol, delimiter) ;

  mw_Col0 = (MWNumericArray) mw_ArrayOut[0] ;
  mw_Col1 = (MWNumericArray) mw_ArrayOut[1] ;
```

```
      mw_Col2 = (MWNumericArray) mw_ArrayOut[2] ;
      mw_Col3 = (MWNumericArray) mw_ArrayOut[3] ;
      mw_Col4 = (MWNumericArray) mw_ArrayOut[4] ;

      // End of Reading data from a file

      //Plot figures
      // Set properties for MATLAB Graphics
      MWCharArray mw_Title  = new MWCharArray("The Title");
      MWCharArray mw_xlabel = new MWCharArray("time");
      MWCharArray mw_ylabel = new MWCharArray("voltage");

      // Call the implemental function
      objMatlab.multiPlotForArrays(mw_Col0, mw_Col1, mw_Col2, mw_Col3, mw_Col4,
                                   mw_Title, mw_xlabel, mw_ylabel);

      objMatlab.WaitForFiguresToDie() ; // program will continue after close this figure

      MWNumericArray.DisposeArray(mw_ArrayOut) ;
      mw_Col0.Dispose();
      mw_Col1.Dispose();
      mw_Col2.Dispose();
      mw_Col3.Dispose();
      mw_Col4.Dispose();

      mw_Title.Dispose();
      mw_xlabel.Dispose();
      mw_ylabel.Dispose();
    }
/* *************************** */
/* *************************** */
/* *************************** */
  }
}
```

end code

Part IV:

Creating and Using COM From MATLAB Builder for .NET In C#

Chapter 16

Using COM Created From MATLAB Builder for .NET In C#

This chapter describes how to use COM created from MATLAB Builder in a simple C# Console Application.

In Windows Vista and Windows 7, there was an error when I was creating a COM. The versions of MATLAB used in this book are MATLAB 2009(b), MATLAB Compiler 4.11, MATLAB Builder NE 3.0.2, with Microsoft Visual 2008. With these versions, MATLAB Compiler supports only Microsoft .Net Framework SDK 2.0, see

http://www.mathworks.com/support/compilers/R2009b/

The .NET Framework version 2.0 is .NET Framework which is included in .NET Framework 3.5 SP1. Also, Windows Vista and Windows 7 won't let user install .Net Framework 2.0. This causes errors when creating a COM from these MATLAB versions, see

http://social.msdn.microsoft.com/forums/en-US/vssetup/
`thread/60424309-bd78-4ca2-b618-03c4a16123b6`

All COMs in this chapter and the next chapter are created in Windows XP including .NET framework 2.0.

The following are steps to create a COM from an M-file and use it in C# functions:

1. Write an M-file, say myplus.m
2. Use MATLAB Builder for .Net to create COM from the M-file myplus.m (the steps shown how to do this are in Section 16.1)
3. Use COM in C# functions

16.1 Creating COM From MATLAB Builder for .NET

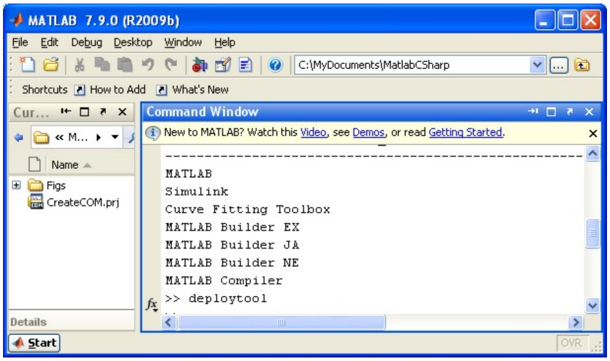
To create COM from MATLAB:

1. Write a MATLAB M-file, for example myplus.m, as follows:

```
function y = myplus(a, b)


y = a + b ;
```

2. Run the following command at the MATLAB prompt: `deploytool`.

   The command initializes to choose a project name as shown in Fig.16.1.



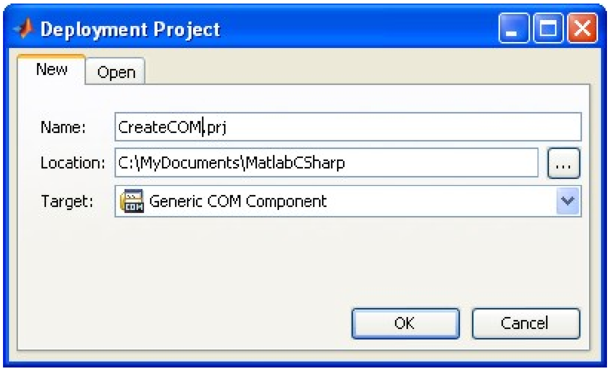
MATLAB 7.9.0 (R2009b)
File Edit Debug Desktop Window Help
C:\MyDocuments\MatlabCSharp
Shortcuts How to Add What's New
Cur...
« M...
Name
Figs
CreateCOM.prj
Details
Start
Command Window
OVR

New to MATLAB? Watch this Video, see Demos, or read Getting Started.

```
-------------------------------------------------------------
MATLAB
Simulink
Curve Fitting Toolbox
MATLAB Builder EX
MATLAB Builder JA
MATLAB Builder NE
MATLAB Compiler
>> deploytool
```

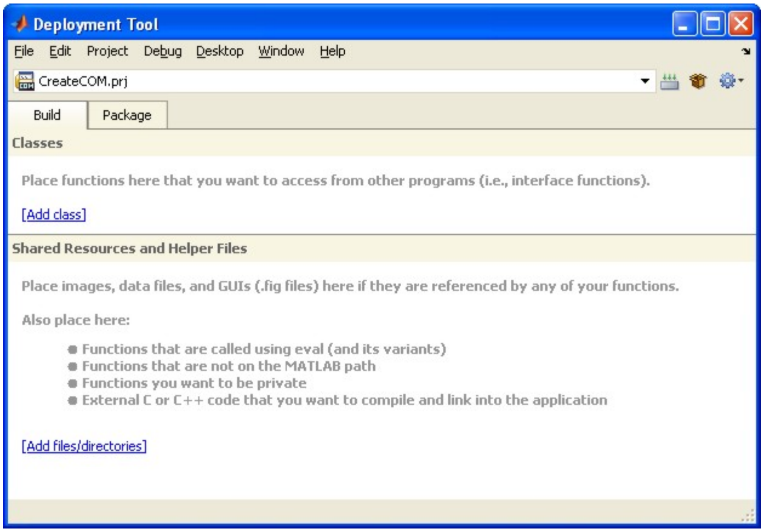
Figure 16.1: Initialization for creating COM

3. Set up the project name, say **CreateCOM.prj**, and target **Generic COM Component** as shown in Fig.16.2. Note that the project name will be the name of **namespace** in the code.
4. Then click OK and it should pop up another dialog to set up files and classes as shown in Fig.16.3.



Name: CreateCOM.prj
Location: C:\MyDocuments\MatlabCSharp
Target: Generic COM Component

Figure 16.2: Choosing a creating COM project



Classes

Place functions here that you want to access from other programs (i.e., interface functions).

Shared Resources and Helper Files

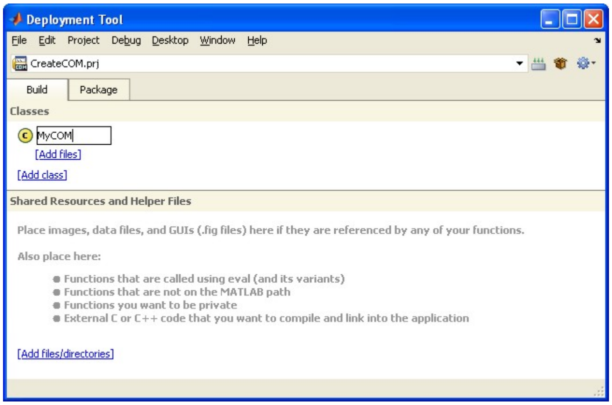
Place images, data files, and GUIs (.fig files) here if they are referenced by any of your functions.

Also place here:

- Functions that are called using eval (and its variants)
- Functions that are not on the MATLAB path
- Functions you want to be private
- External C or C++ code that you want to compile and link into the application

Figure 16.3: Creating COM (continue)

5. In this dialog (Fig.16.3), click on **Add Class** (see Fig.16.4) to add a class name, say **MyCOM**. Note that, the class name that MATLAB will create is **MyCOMClass** (MATLAB adds the suffix **Class** at the end).



CreateCOM.prj

MyCOM

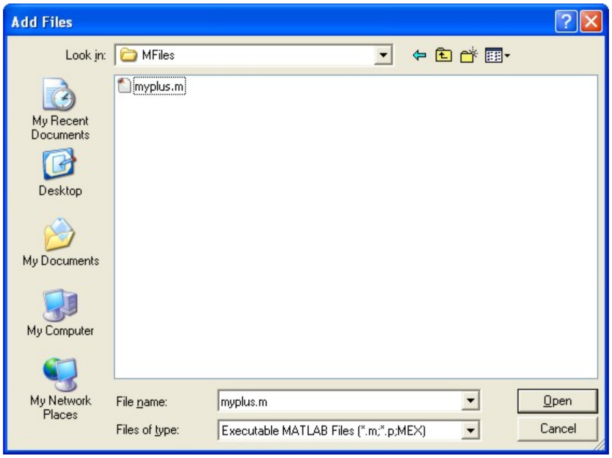
Place images, data files, and GUIs (.fig files) here if they are referenced by any of your functions.

Also place here:

- Functions that are called using eval (and its variants)
- Functions that are not on the MATLAB path
- Functions you want to be private
- External C or C++ code that you want to compile and link into the application

Figure 16.4: Creating COM – Setting a class

6. Click on **Add files** (below **MyCOM**) to choose the M-file *myplus.m* (see Fig.16.5). You can see the file *myplus.m* added to the project as shown in Fig.16.6.



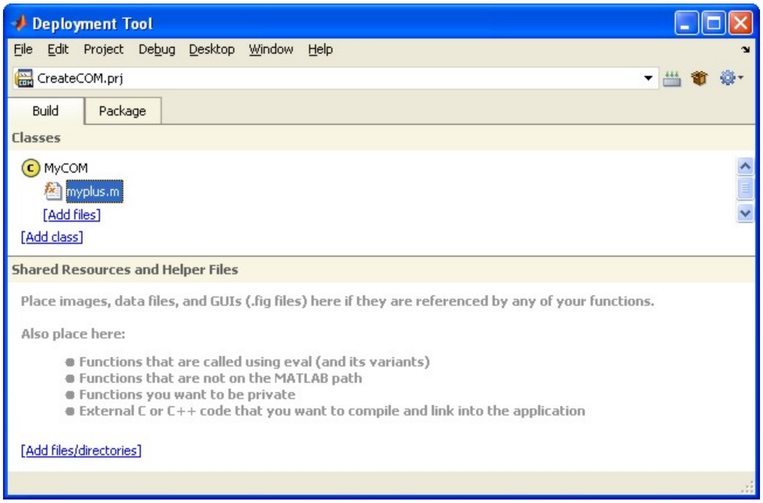
Look in: MFiles

File name: myplus.m

Files of type: Executable MATLAB Files (*.m;*.p;MEX)

Figure 16.5: Creating COM – Setting a M-file

7. On Fig16.6, click on **Package** to see the *dll* file name, **CreateCOM_1_0.dll** (see Fig.16.7). We'll choose this file to add as reference later in coding.



Place images, data files, and GUIs (.fig files) here if they are referenced by any of your functions.

Also place here:

- Functions that are called using eval (and its variants)
- Functions that are not on the MATLAB path
- Functions you want to be private
- External C or C++ code that you want to compile and link into the application

Figure 16.6: Creating COM – Setting a M-file (continue)

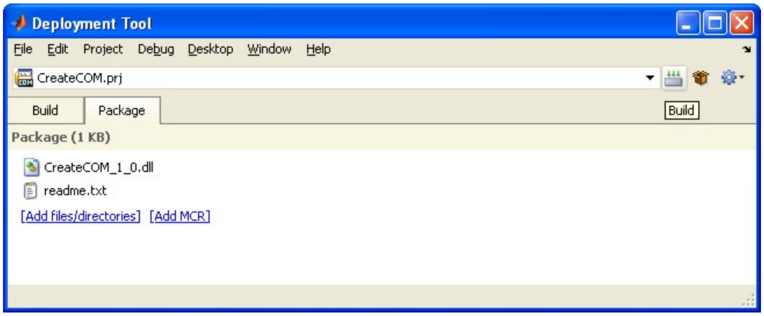
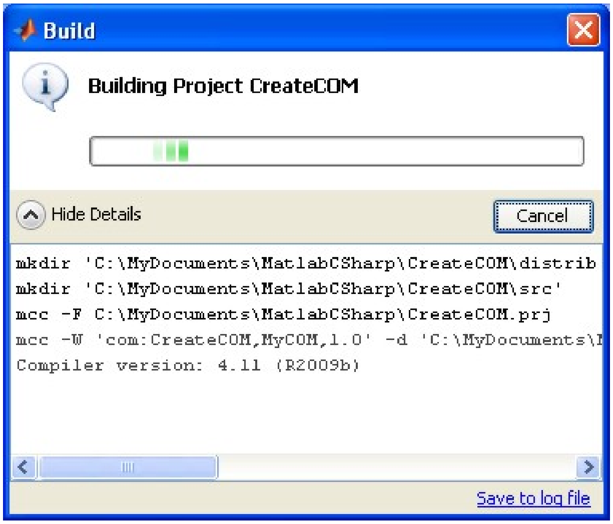


Figure 16.7: Creating COM – The *dll* file name

8. In Fig16.7, click on the **Build** icon (or click on **Project, Build**) to generate the COM. The project will generate a COM based on M-files we add to the project, see Fig.16.8.



```
mkdir 'C:\MyDocuments\MatlabCSharp\CreateCOM\distrib
mkdir 'C:\MyDocuments\MatlabCSharp\CreateCOM\src'
mcc -F C:\MyDocuments\MatlabCSharp\CreateCOM.prj
mcc -W 'com:CreateCOM,MyCOM,1.0' -d 'C:\MyDocuments\
Compiler version: 4.11 (R2009b)
```

Figure 16.8: Creating COM – Process of building COM

9. The project will create the COM **CreateCOM_1_0.dll** in the directory **...\CreateCOM\distrib** (see Fig.16.9).

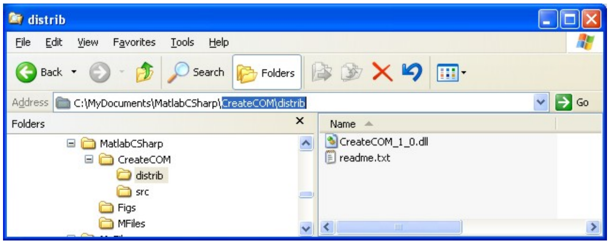


Figure 16.9: Creating COM – Directory of the *dll* file

16.2 Using COM in C# functions

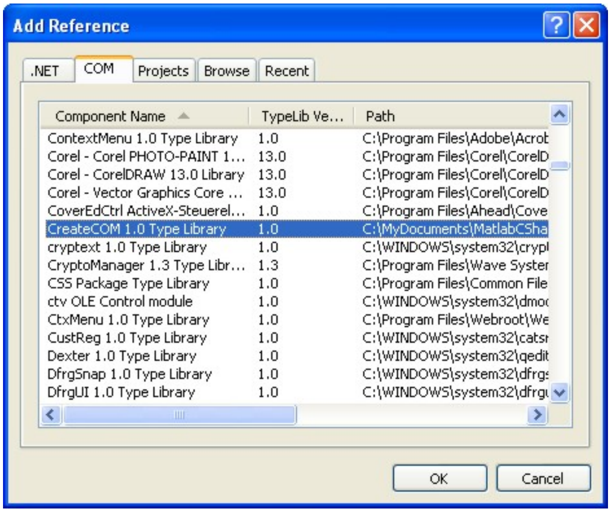
In this section we'll describe the steps to use the created COM, `CreateCOM_1_0.dll`, in a target machine (without MATLAB software) through an example. The steps are:

1. Copy and install the file `MCRInstaller.exe` to your target machine. The contained folder

of `MCRInstaller.exe` is shown in the file `readme.txt` in the created directory **...\Create-COM\distrib**. In my computer, this file is located at **C:\Program Files\MATLAB\R2009b\toolbox\compiler\deploy win32\MCRInstaller.exe**.

2. Create a normal C# Console Application, say **Example**.

3. In the project **Example**, right click on **References** then click **Add Reference**. The Add Reference dialog appears (Fig. 16.10) and then choose **CreateCOM_1_0 Type Library**. Click OK to choose this COM.



Component Name	TypeLib Ve...	Path
ContextMenu 1.0 Type Library	1.0	C:\Program Files\Adobe\Acrot
Corel - Corel PHOTO-PAINT 1...	13.0	C:\Program Files\Corel\CorelD
Corel - CorelDRAW 13.0 Library	13.0	C:\Program Files\Corel\CorelD
Corel - Vector Graphics Core ...	13.0	C:\Program Files\Corel\CorelD
CoverEdCtrl ActiveX-Steuerel...	1.0	C:\Program Files\Ahead\Cove
CreateCOM 1.0 Type Library	1.0	C:\MyDocuments\MatlabCSha
cryptext 1.0 Type Library	1.0	C:\WINDOWS\system32\crypl
CryptoManager 1.3 Type Libr...	1.3	C:\Program Files\Wave Syster
CSS Package Type Library	1.0	C:\Program Files\Common File
ctv OLE Control module	1.0	C:\WINDOWS\system32\dmo
CtxMenu 1.0 Type Library	1.0	C:\Program Files\Webroot\We
CustReg 1.0 Type Library	1.0	C:\WINDOWS\system32\catsr
Dexter 1.0 Type Library	1.0	C:\WINDOWS\system32\qedit
DfrgSnap 1.0 Type Library	1.0	C:\WINDOWS\system32\dfrg
DfrgUI 1.0 Type Library	1.0	C:\WINDOWS\system32\dfrg

Figure 16.10: Adding COM to References

Note: Before going to the code, we want to mention something here:

1. When we add the COM `CreateCOM_1_0.dll` to **References**, a class is created and named **MyCOMClass**. This name is added a suffix **'Class'** to our predefined class name **MyCOM**.

2. The created function from the M-function `myplus(a, b)` is `void myplus(int, ref object, object, object)`.

The following code demonstrates how to call the function *myplus(..)* in a C# function.

Listing code

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using CreateCOM ;

namespace Example
{
  class Program
  {
    static void Main(string[] args)
    {
      Program objPro = new Program() ;

      double result = objPro.CalculatePlus(1.2, 3.4) ;
      Console.WriteLine("Result :{0}", result.ToString() ) ;
    }

    /* ****************** */
    public double CalculatePlus(double db_a, double db_b)
    {
      MyCOMClass objMatlab = new MyCOMClass() ;

      object obj_A = (object) db_a ;
      object obj_B = (object) db_b ;
      object obj_C = (object) 0 ;

      objMatlab.myplus(1, ref obj_C, obj_A, obj_B) ;
      double outputC = (double) obj_C ;

      return outputC ;
    }

/* ****************** */
/* ******************* */
/* ******************* */
  }
}
```

end code

Chapter 17

Creating and Using COM From Multiple M-files In C#

This chapter describes how to use COM created from multiple MATLAB M-files and use them in C# Console Application.

In Windows Vista and Windows 7, there was an error when I was creating a COM. The versions of MATLAB used in this book are MATLAB 2009(b), MATLAB Compiler 4.11, MATLAB Builder NE 3.0.2, with Microsoft Visual 2008. With these versions, MATLAB Compiler supports only Microsoft .Net Framework SDK 2.0, see

http://www.mathworks.com/support/compilers/R2009b/

The .NET Framework version 2.0 is .NET Framework which is included in .NET Framework 3.5 SP1. Also, Windows Vista and Windows 7 won't let user install .Net Framework 2.0, so this causes errors when creating a COM from these MATLAB versions, see

http://social.msdn.microsoft.com/forums/en-US/vssetup/
`thread/60424309-bd78-4ca2-b618-03c4a16123b6`

The COM in this chapter was created by Windows XP including .NET framework 2.0.

17.1 Creating COM From Multiple MATLAB M-files

We'll write the following M-files to create a COM ***MATLABCOMLinear***.

`mydet.m, myinv.m, myminus.m, mymtimes.m, myplus.m, and mytranspose.m`

mymtimes.m

```
function y = mymtimes(a, b)


y = a*b ;
```

mytranspose.m

```
function y = mytranspose( x )


y = x' ;
```

mylu.m

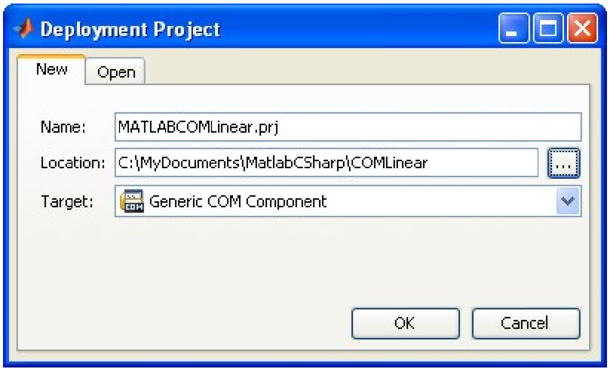
```
function [L,U,P] = mylu(A)


[L,U,P] = lu(A) ;
```

mymldivide.m

```
function x = mymldivide(A, b)
%solve equation Ax = b
x = A\b ;
```

From these M-files we'll use MATLAB Builder for .Net to create a COM, named **MATLABCOMLinear**. The procedure to create this COM and add it to **References** of the C# project is the same as described in the previous chapter, Chapter 16.

1. Create a COM named **MATLABCOMLinear**
2. Adding all M-files in above to the COM
3. MATLAB Builder for .NET will generate a dll file MATLABCOMLinear_1_0.dll
4. Add this COM `MATLABCOMLinear` to **References** of the C# project

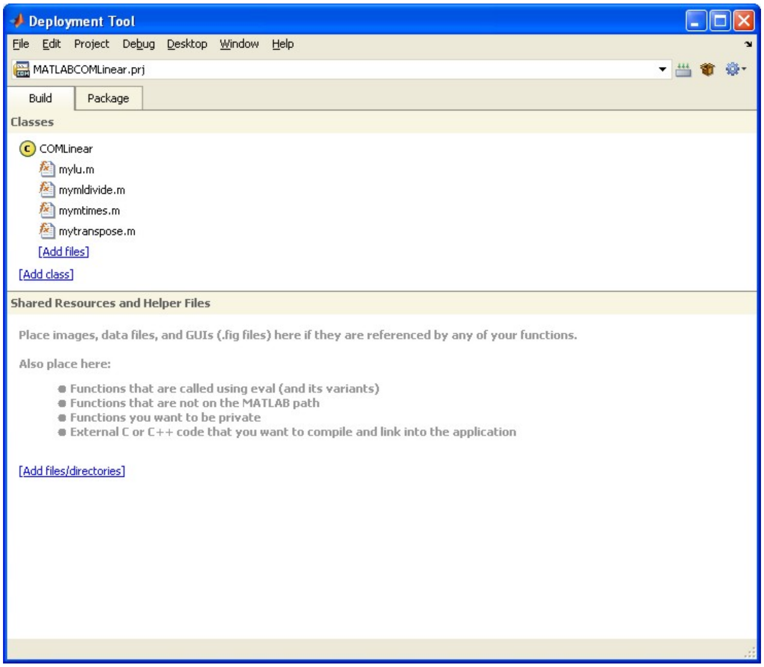


Name: MATLABCOMLinear.prj

Location: C:\MyDocuments\MatlabCSharp\COMLinear

Target: Generic COM Component

Figure 17.1: Creating COM from Multiple M-files



MATLABCOMLinear.prj

Classes

- COMLinear
  - mylu.m
  - mymldivide.m
  - mymtimes.m
  - mytranspose.m

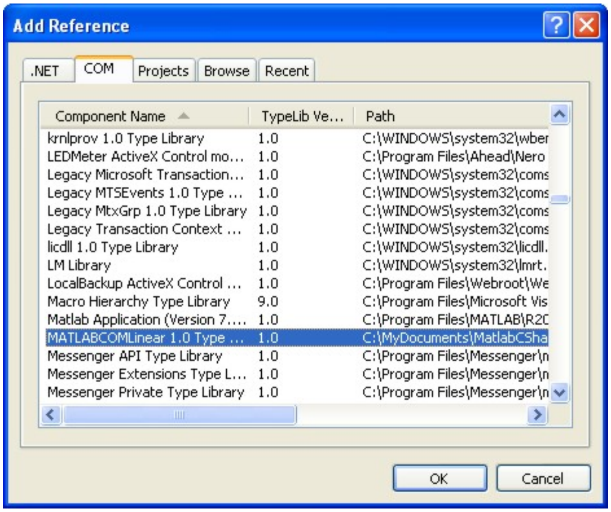
Shared Resources and Helper Files

Place images, data files, and GUIs (.fig files) here if they are referenced by any of your functions.

Also place here:

- Functions that are called using eval (and its variants)
- Functions that are not on the MATLAB path
- Functions you want to be private
- External C or C++ code that you want to compile and link into the application

Figure 17.2: Adding M-files to COM



Component Name	TypeLib Ve...	Path
krnlprov 1.0 Type Library	1.0	C:\WINDOWS\system32\wber
LEDMeter ActiveX Control mo...	1.0	C:\Program Files\Ahead\Nero
Legacy Microsoft Transaction...	1.0	C:\WINDOWS\system32\coms
Legacy MTSEvents 1.0 Type ...	1.0	C:\WINDOWS\system32\coms
Legacy MtxGrp 1.0 Type Library	1.0	C:\WINDOWS\system32\coms
Legacy Transaction Context ...	1.0	C:\WINDOWS\system32\coms
licdll 1.0 Type Library	1.0	C:\WINDOWS\system32\licdll.
LM Library	1.0	C:\WINDOWS\system32\lmrt.
LocalBackup ActiveX Control ...	1.0	C:\Program Files\Webroot\We
Macro Hierarchy Type Library	9.0	C:\Program Files\Microsoft Vis
Matlab Application (Version 7....	1.0	C:\Program Files\MATLAB\R2C
MATLABCOMLinear 1.0 Type ...	1.0	C:\MyDocuments\MatlabCSha
Messenger API Type Library	1.0	C:\Program Files\Messenger\n
Messenger Extensions Type L...	1.0	C:\Program Files\Messenger\n
Messenger Private Type Library	1.0	C:\Program Files\Messenger\n

Figure 17.3: Adding M-files to COM (cont.)

17.2 Using COM From MATLAB Builder for .NET To Calculate Matrix Multiplication

Problem 1

input Matrix **A** and **B**

$$\mathbf{A} = \begin{bmatrix} 1.1 & 2.2 & 3.3 & 4.4 \\ 5.5 & 6.6 & 7.7 & 8.8 \\ 9.9 & 10.10 & 11.11 & 12.12 \end{bmatrix} \quad , \quad \mathbf{B} = \begin{bmatrix} 10 & 11 \\ 12 & 13 \\ 14 & 15 \\ 16 & 17 \end{bmatrix}$$

output Finding the product matrix $\mathbf{C} = \mathbf{A} * \mathbf{B}$

This following code describes how to use the created COM **MATLABCOMLinear** to calculate the matrix multiplication in Problem 1.

Listing code

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using MATLABCOMLinear ;

namespace Example
{
  class Program
  {
    static void Main(string[] args)
    {
      Program objPro = new Program();

      Console.WriteLine("\n Matrix Multiplication");
      objPro.MatrixMultiplication();

      Console.WriteLine("\n Linear System Equation");
      objPro.LinearSystemEquations();

      Console.WriteLine("\n LU decompression");
      objPro.LU_decompression();

    }

/* ******************************** */
public void MatrixMultiplication()
{
  COMLinearClass matlabObj = new COMLinearClass() ;

  double[,] db_A =  {   { 1.1, 2.2  , 3.3  , 4.4   }  ,
                        { 5.5, 6.6  , 7.7  , 8.8   }  ,
                        { 9.9, 10.10, 11.11, 12.12 }  };
  double[,] db_B = { { 10, 11 }, { 12, 13 }, { 14, 15 }, { 16, 17 } };

  int rowA = db_A.GetUpperBound(0) - db_A.GetLowerBound(0) + 1;
  int colA = db_A.GetUpperBound(1) - db_A.GetLowerBound(1) + 1;

  int rowB = db_B.GetUpperBound(0) - db_B.GetLowerBound(0) + 1;
  int colB = db_B.GetUpperBound(1) - db_B.GetLowerBound(1) + 1;

  object obj_A = (object)db_A;
  object obj_B = (object)db_B;

  int rowC = rowA;
  int colC = colB;

  object obj_C = (object)0;

  matlabObj.mymtimes(1, ref obj_C, obj_A, obj_B);



  double[,] db_C = new double[rowC, colC];
  int aLength = rowC * colC;

  Array.Copy((double[,])obj_C, db_C, aLength);
  PrintMatrix(db_C);

}



/* ******************************** */
public static void PrintMatrix(double[,] matrix)
{
  int i, j, row, col ;
  row = matrix.GetUpperBound(0) - matrix.GetLowerBound(0) + 1;
  col = matrix.GetUpperBound(1) - matrix.GetLowerBound(1) + 1;

  for (i=0; i<row; i++)
  {
    for (j=0; j<col; j++)
    {
      Console.Write("{0} \t", matrix.GetValue(i,j).ToString() ) ;
    }
    Console.WriteLine() ;
  }

}
```

end code

17.3 Using COM from MATLAB COM Builder to Solve Linear System Equations

Problem 2

input Matrix **A** and vector **b**

$$\mathbf{A} = \begin{bmatrix} 1.1 & 5.6 & 3.3 \\ 4.4 & 12.3 & 6.6 \\ 7.7 & 8.8 & 9.9 \end{bmatrix} \quad , \quad \mathbf{b} = \begin{bmatrix} 12.5 \\ 32.2 \\ 45.6 \end{bmatrix}$$

output . Finding the solution **x** of linear system equations, $\mathbf{Ax} = \mathbf{b}$

. Finding the lower **L** and upper **U** of the matrix **A**

This following code describes how to use the created COM to solve Problem 2.

Listing code

```
public void LinearSystemEquations()
{
  COMLinearClass matlabObj = new COMLinearClass() ;
  /* Solve general linear system equations Ax = b */
  double[,] db_A =  {   {1.1,  5.6, 3.3}  ,
                        {4.4, 12.3, 6.6}  ,
                        {7.7,  8.8, 9.9}  };

  double[] db_vectorb =  { 12.5, 32.2, 45.6 };

  object obj_A    = (object)db_A;
  object obj_b    = (object)db_vectorb;
  object obj_Transb   = (object)0;

  // to get a comlumn vector b
  // note: type of obj_Transb is Double[,]
  matlabObj.mytranspose(1, ref obj_Transb, obj_b);

  object obj_x = (object)0;
  matlabObj.mymldivide(1, ref obj_x, obj_A, obj_Transb);

  // note: type of obj_x is a matrix with column = 1
  int aLength = db_vectorb.Length;
  double[,] db_x = new double[aLength, 1];

  Array.Copy((double[,])obj_x, db_x, aLength);
  PrintMatrix(db_x);

}

/* **************************** */
public void LU_decompression()
{
  /* find lower and upper matrixes */
  double[,] db_A =  {   {1.1,  5.6, 3.3}  ,
                                      {4.4, 12.3, 6.6}  ,
                                      {7.7,  8.8, 9.9}  };
  object obj_A = (object)db_A;
  object obj_L = (object)0;
  object obj_U = (object)0;
  object obj_P = (object)0;

  /* call an implemental function */
  COMLinearClass matlabObj = new COMLinearClass() ;

  matlabObj.mylu(3, ref obj_L, ref obj_U, ref obj_P, obj_A);

  int row = db_A.GetUpperBound(0) - db_A.GetLowerBound(0) + 1;
  int col = db_A.GetUpperBound(1) - db_A.GetLowerBound(1) + 1;

  double[,] db_L = new double[row, col];
  double[,] db_U = new double[row, col];

  int aLength = row * col;
  Array.Copy((double[,])obj_L, db_L, aLength);
  Array.Copy((double[,])obj_U, db_U, aLength);

  /* print out */
  Console.WriteLine("\n The lower matrix");
  PrintMatrix(db_L);

  Console.WriteLine("\n The upper matrix");
  PrintMatrix(db_U);

}
```

end code

Part V:

Using MATLAB API Functions in C# – Using MATLAB Workspace in C#

Chapter 18

Calling MATLAB Workspace in C# Functions

This chapter describes how to call the MATLAB workspace to perform particular tasks from a C# function. When performing tasks in the MATLAB workspace, we need to put inputs from a C# function to the MATLAB workspace and then get outputs from the MATLAB workspace back to the C# function. In this chapter, scalars, vectors, and matrixes are used as the function inputs/outputs in the example codes. It also shows how to put a string to MATLAB workspace and plot graphic from MATLAB workspace.

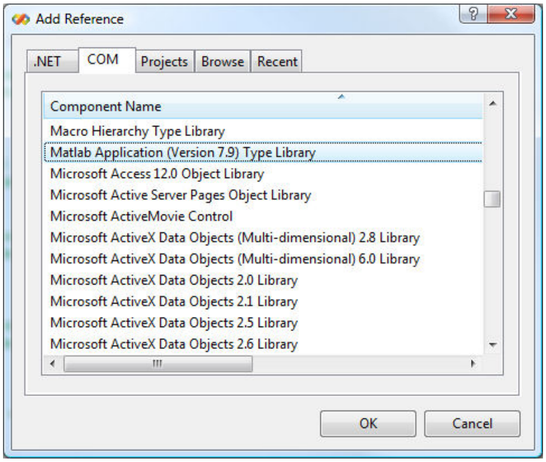
18.1 Setting Up a Project To Call MATLAB Workspace

In this section we built a C# project to call MATLAB workspace into the project. Firstly, we need to set up a C# project following below steps.

1. Create a regular C# project in Console Application
2. Add reference from MATLAB as follows:
   - On project menu, click *Project*, *Add Reference* .
   - The project will pop up a following window, see Fig. 18.1, then click on *COM* tab, *Matlab Application Type Library* (see Fig.18.2).
3. Add the class *MatlabCSharpAPI* to the project. This class *MatlabCSharpAPI* is built

based-on the MATLAB class *MLAppClass* and the code of this class *MLAppClass* is at the end of this chapter. Do the following steps:

- Put the file *MatlabCSharpAPIVer4p11.cs* in the same directory of the file *Program.cs*, see Fig.18.3
- Add this file to the project by click *Project* on the project menu, then click *Add Existing Item*. The project will pop up a window then choose the file *MatlabCSharpAPIVer4p11.cs*. The file should be in the project as shown in Fig.18.4.



Component Name
Macro Hierarchy Type Library
Matlab Application (Version 7.9) Type Library
Microsoft Access 12.0 Object Library
Microsoft Active Server Pages Object Library
Microsoft ActiveMovie Control
Microsoft ActiveX Data Objects (Multi-dimensional) 2.8 Library
Microsoft ActiveX Data Objects (Multi-dimensional) 6.0 Library
Microsoft ActiveX Data Objects 2.0 Library
Microsoft ActiveX Data Objects 2.1 Library
Microsoft ActiveX Data Objects 2.5 Library
Microsoft ActiveX Data Objects 2.6 Library

Figure 18.1: Adding reference in C#

The following subroutines will show how to call MATLAB workspace in a C# function to solve problems. The rest of code is at the end of this chapter.

18.2 Calling MATLAB Workspace with Input/Output as a Scalar

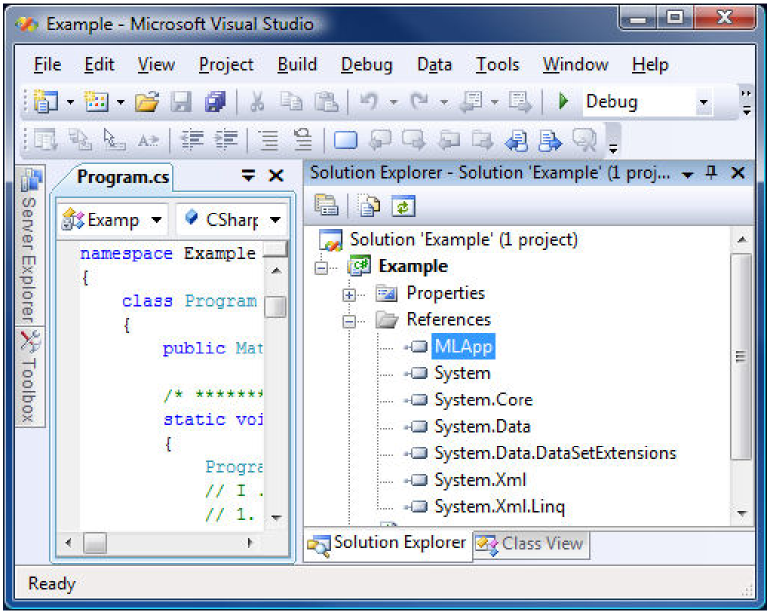
In this section, we will use MATLAB functions to solve the following problems.

Problem 1 Put and get numbers between C# and MATLAB workspace.

input Two given numbers $a = 1.2$ and $b = 2.5$

output

Call the MATLAB workspace to perform the task, $c = a + b$, in a C# function


Example - Microsoft Visual Studio
File Edit View Project Build Debug Data Tools Window Help
Debug
Program.cs
Examp
CSharp
Server Explorer
Toolbox
Solution Explorer - Solution 'Example' (1 proj...
Solution 'Example' (1 project)
Example
Properties
References
MLApp
System
System.Core
System.Data
System.Data.DataSetExtensions
System.Xml
System.Xml.Linq
Solution Explorer
Class View
Ready

```
namespace Example
{
    class Program
    {
        public Mat

        /* *******
        static voi
        {
            Progra
            // I .
            // 1.
```

Figure 18.2: Reference in a C# project

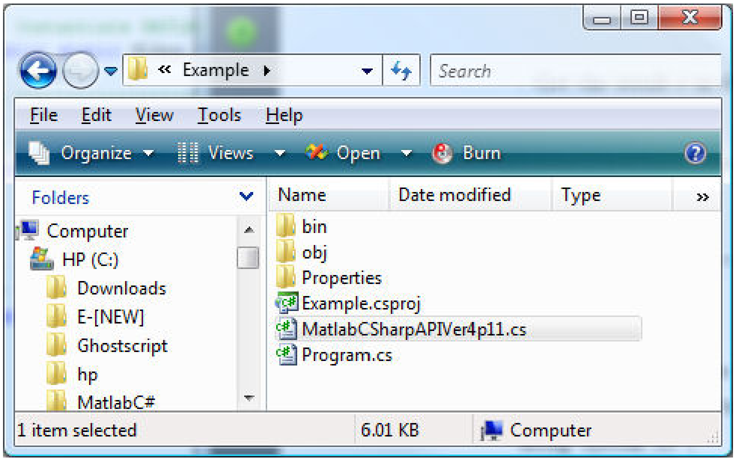
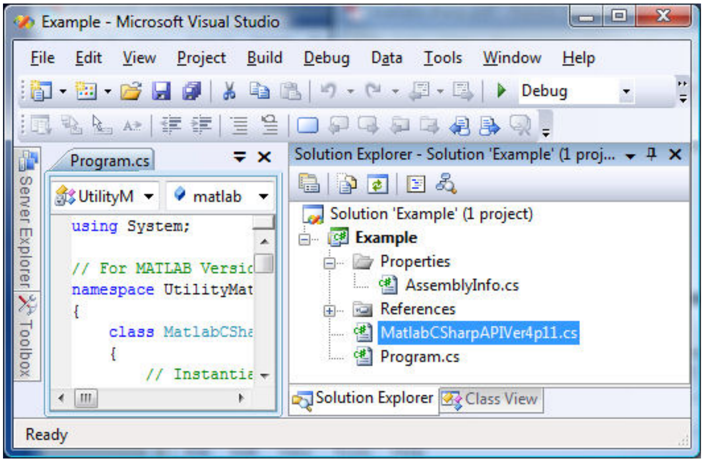


Figure 18.3: Add the file $MatlabCsharpAPIVer4p11.cs$ in a C# project



```
using System;

// For MATLAB Versi
namespace UtilityMat
{
    class MatlabCSha
    {
        // Instanti
```

Figure 18.4: File *MatlabCsharpAPIVer4p11.cs* in a C# project

Get the result *c* in the MATLAB workspace and put back to the C# function

The following subroutine is used to solve Problem 1.

The full of code is at the end of this chapter.

Listing code

```
public void SimplePlus()
{
    double db_a = 1.1 ;
    double db_b = 2.2 ;

    MatlabCSharpAPI.PutScalarReal("ml_a", db_a ) ;
    MatlabCSharpAPI.PutScalarReal("ml_b", db_b ) ;

    MatlabCSharpAPI.Execute(" ml_c = ml_a + ml_b ; ") ;

    double db_c = 0 ;
    MatlabCSharpAPI.GetScalarReal("ml_c", ref db_c) ;

    //// print out
    Console.WriteLine("The result from simple addition: ") ;
    Console.WriteLine( db_c.ToString() ) ;
}
```

end code

Problem 2 Put and get complex numbers between C# and MATLAB workspace.

input Two given complex numbers a and b:

real of a = 1.1, imaginary of a = 101.1

real of b = 2.2, imaginary of b = 202.2

output

Call the MATLAB workspace to perform the task, $c = a + b$, in a C# function

Get the result of the complex number c in the MATLAB workspace and put back to the C# function

The following subroutine is used to solve Problem 2.

Listing code

```
public static void PlusComplexNumbers() {

  // 1. Set number
  double aReal = 1.1    ;
  double aImag = 101.1  ;

  double bReal = 2.2    ;
  double bImag = 202.2  ;

  // 2. Perform tasks in Matlab Workspace
  matlab.PutWorkspaceData("ml_aReal", "base", aReal) ;
  matlab.PutWorkspaceData("ml_aImag", "base", aImag) ;
  matlab.Execute(" ml_aComp = complex(ml_aReal, ml_aImag) ; ") ;

  matlab.PutWorkspaceData("ml_bReal", "base", bReal) ;
  matlab.PutWorkspaceData("ml_bImag", "base", bImag) ;
  matlab.Execute(" ml_bComp = complex(ml_bReal, ml_bImag) ; ") ;

  matlab.Execute(" ml_cComp = ml_aComp + ml_bComp ; " ) ;

  matlab.Execute(" ml_cReal = real(ml_cComp) ; ") ;
  matlab.Execute(" ml_cImag = imag(ml_cComp) ; ") ;

  // 3. Get data from Matlab Workspace
  object obj_cReal, obj_cImag ;
  matlab.GetWorkspaceData("ml_cReal", "base", out obj_cReal) ;
  matlab.GetWorkspaceData("ml_cImag", "base", out obj_cImag) ;

  // 4. Print out
  double cReal = 0 ;
  double cImag = 0 ;

  cReal = Convert.ToDouble(obj_cReal) ;
  cImag = Convert.ToDouble(obj_cImag) ;

  Console.WriteLine("Result of a complex number :") ;
  Console.WriteLine("\t Real Part: {0}", cReal.ToString() ) ;
  Console.WriteLine("\t Imag Part: {0}", cImag.ToString() ) ;

}
```

end code

18.3 Calling MATLAB Workspace with Input/Output as a Vector and a Matrix

In this section, we will use MATLAB functions to solve the following problems.

Problem 3

input . A matrix **A** and a vector **b**,

$$\mathbf{A} = \begin{bmatrix} 1.1 & 5.6 & 3.3 \\ 4.4 & 12.3 & 6.6 \\ 7.7 & 8.8 & 9.9 \end{bmatrix} \quad , \quad \mathbf{b} = \begin{bmatrix} 12.5 \\ 32.2 \\ 45.6 \end{bmatrix}$$

output

. Call the MATLAB workspace to perform the tasks as following in a C# function:

a) Finding the solution **x** of linear system equations $\mathbf{Ax} = \mathbf{b}$

b) Calculating the upper matrix U with the LU decompression method

c) Getting results of matrixes **U** and **L** in the MATLAB workspace and converting to C# double

The following subroutine is used to solve Problem 3.

Listing code

```
public void LinearSolve()
{
  double[,] A = { {1.1, 5.6, 3.3},  {4.4, 12.3, 6.6} , { 7.7, 8.8, 9.9} } ;
  double[]  b = { 12.5, 32.2 , 45.6 } ;

  // Put values to MATLAB variables
  MatlabCSharpAPI.PutMatrixReal("ml_A", A) ;
  MatlabCSharpAPI.PutVectorReal("ml_b", b) ;

  //MATLAB workspace tasks : Solve Ax = b, and get a vector x and an upper matrix U.
  //The problem Ax = b is solved here as the purpose
  //of showing the use of the MATLAB workspace in a C# function.
  MatlabCSharpAPI.Execute(" ml_vectorX = mldivide(ml_A, ml_b) ; ") ;
  MatlabCSharpAPI.Execute(" [ml_L, ml_U, ml_P] = lu( ml_A )   ; ") ;

  //Get results from MATLAB workspace
  Array vectorX = new double [b.Length] ;
  MatlabCSharpAPI.GetVectorReal("ml_vectorX", ref vectorX) ;

  int row, col;
  row = A.GetUpperBound(0) - A.GetLowerBound(0) + 1;
  col = A.GetUpperBound(1) - A.GetLowerBound(1) + 1;

  Array matrixU = Array.CreateInstance(typeof(double), row, col) ;
  MatlabCSharpAPI.GetMatrixReal("ml_U", ref matrixU) ;

  // Print out
  Console.WriteLine("Linear system: result of the vector solution") ;
  MatlabCSharpAPI.PrintVector(vectorX) ;

  Console.WriteLine() ;
  Console.WriteLine("Linear system: result of the U matrix") ;
  MatlabCSharpAPI.PrintMatrix(matrixU) ;

  // Or Convert to a double array
  Console.WriteLine("Section linear solve: Convert to double array: ") ;
  double[] db_vectorX = new double [b.Length] ;
  vectorX.CopyTo(db_vectorX, 0) ;
  MatlabCSharpAPI.PrintVector(db_vectorX) ;

  // Or convert to a double matrix
  Console.WriteLine("Section Linear solve: Convert to double matrix") ;
  double[,] db_matrixU = new double[row, col] ;
  Array.Copy(matrixU, db_matrixU, row*col) ;
  MatlabCSharpAPI.PrintMatrix(db_matrixU) ;
}
```

end code

Problem 4 Put and get complex vectors (one-dimensional array) between C# and MATLAB workspace.

input

. Two given complex vectors a and b:

Real values of vector a are: 0.1, 1.1, 11.1

Imaginary values of vector a are: 100.1, 101.1, 111.1

Real values of vector b are: 0.2, 2.2, 22.2

Imaginary values of vector b are: 200.2, 202.2, 222.2

output

. Call the MATLAB workspace to perform the task, $c = a + b$, in a C# function

. Get the result of the complex vector c in the MATLAB workspace and put back to the C# function

The following subroutine is used to solve Problem 4.

Listing code

```
public void PlusComplexVectors()
{
  double [] vecAReal = {  0.1,  1.1,   11.1 } ;
  double [] vecAImag = {100.1, 101.1, 111.1 } ;

  double [] vecBReal = {  0.2,  2.2,   22.2 } ;
  double [] vecBImag = {200.2, 202.2, 222.2 } ;

  MatlabCSharpAPI.PutVectorComplex("ml_vecAComp", vecAReal, vecAImag) ;
  MatlabCSharpAPI.PutVectorComplex("ml_vecBComp", vecBReal, vecBImag) ;

  MatlabCSharpAPI.Execute( " ml_vecCComp = ml_vecAComp + ml_vecBComp ; " ) ;

  Array vectorC_Real = new double [vecAReal.Length] ;
  Array vectorC_Imag = new double [vecAImag.Length] ;

  MatlabCSharpAPI.GetVectorComplex("ml_vecCComp", ref vectorC_Real, ref vectorC_Imag);

  // Print out
  Console.WriteLine("Result of a complex vector:") ;
  Console.WriteLine("Real vector:") ;
  MatlabCSharpAPI.PrintVector(vectorC_Real) ;

  Console.WriteLine("Imag vector:") ;
  MatlabCSharpAPI.PrintVector(vectorC_Imag) ;

  // Convert to a double array
  double[] db_VectorC_Real = new double [vecAReal.Length] ;
  double[] db_vectorC_Imag = new double [vecAImag.Length] ;

  vectorC_Real.CopyTo(db_VectorC_Real,0) ;
  vectorC_Imag.CopyTo(db_vectorC_Imag,0) ;
  Console.WriteLine("Section complex vector: Convert vector to a double array:") ;
  MatlabCSharpAPI.PrintVector(db_VectorC_Real) ;
  MatlabCSharpAPI.PrintVector(db_vectorC_Imag) ;
}
```

end code

18.4 Generating a MATLAB Graphic from a C# Function

In this section we will directly create a MATLAB graphic by performing a plotting task in the MATLAB workspace .

Problem 5 Plot graphic from two arrays:

```
X = 0, 50, 100, 150, 200, 250, 300, 350, 400, 450, 500, 550, 600
Y = 0.4000, 0.2426, 0.1472, 0.0893, 0.0541, 0.0328, 0.0199,
            0.0121, 0.0073, 0.0044, 0.0027, 0.0016,  0.0010
```

Following is the code to create the graphic. The created figure is shown in Fig. 18.5 .

Listing code

```
public void API_Plot()
{
    double[]X = {0, 50, 100, 150, 200, 250, 300, 350, 400, 450, 500, 550, 600} ;

    double[]Y = {0.4000, 0.2426, 0.1472, 0.0893, 0.0541, 0.0328, 0.0199,
                 0.0121, 0.0073, 0.0044, 0.0027, 0.0016,  0.0010 } ;

    String myColor = "blue";

    MatlabCSharpAPI.PutVectorReal("ml_X", X) ;
    MatlabCSharpAPI.PutVectorReal("ml_Y", Y) ;

    MatlabCSharpAPI.PutString("ml_plotColor", myColor) ;

    MatlabCSharpAPI.Execute( "plot(ml_X, ml_Y, ml_plotColor); ") ;
    MatlabCSharpAPI.Execute( "grid on; ") ;
    Console.WriteLine("Hit Enter key to close the figure and continue.") ;
    Console.ReadLine() ; // Keep figure staying
}
```

end code

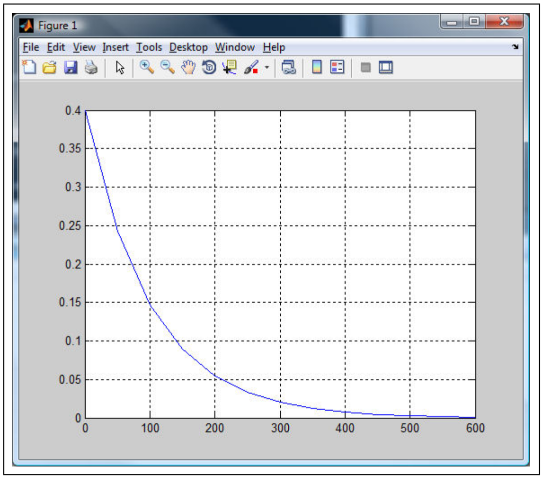


Figure 18.5: The figure is created by using the MATLAB workspace

Remark

- With directly creating the graphic by MATLAB workspace, you can use the graphic features as Insert, Tool, etc.
- You need to hit Enter key to close the figure as shown in the line of the code **System.Console.Read() ;**.

The following is the full code for this chapter.

Listing code

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using UtilityMatlabCSharpAPIVer4p11 ;
```

```
namespace Example
{
  class Program
  {
    public MatlabCSharpAPI CSharpObj = new MatlabCSharpAPI() ;

    /* ******************************** */
    static void Main(string[] args)
    {
      Program progObj = new Program() ;

      // I . Scalar
      // 1. Put scalars in MATLAB Workspace
      // 2. Get a scalar from MATLAB Workspace
      Console.WriteLine() ;
      progObj.SimplePlus() ;

      Console.WriteLine() ;
      progObj.PlusComplexNumbers() ;



      // II. Vector
      // 1. Put a vector in MATLAB API
      // 2. Get a vecotr from MATLAB API
      Console.WriteLine() ;
      progObj.PlusRealVectors() ;

      Console.WriteLine() ;
      progObj.PlusComplexVectors() ;

      // III. Matrix
      // 1. Put a matrix in MATLAB  Workspace
      // 2. Get a matrix from MATLAB Workspace
      Console.WriteLine() ;
      progObj.LinearSolve() ;
      // IV. Plot
      // 1. Put a string in MATLAB Workspace
      // 2. Plot graphic from MATLAB Workspace
      Console.WriteLine() ;
      progObj.API_Plot() ;

    }

    /* ******************************** */
    public void SimplePlus()
    {
      double db_a = 1.1 ;
      double db_b = 2.2 ;

      MatlabCSharpAPI.PutScalarReal("ml_a", db_a ) ;
      MatlabCSharpAPI.PutScalarReal("ml_b", db_b ) ;

      MatlabCSharpAPI.Execute(" ml_c = ml_a + ml_b ; ") ;

      double db_c = 0 ;
      MatlabCSharpAPI.GetScalarReal("ml_c", ref db_c) ;

      //// print out
      Console.WriteLine("The result from simple addition: ") ;
      Console.WriteLine( db_c.ToString() ) ;
    }

    /* **************************** */
    public void PlusComplexNumbers()
    {
      MatlabCSharpAPI.PlusComplexNumbers() ;
    }

    /* **************************** */
    public void LinearSolve()
    {
```

```
double[,] A = { {1.1, 5.6, 3.3},  {4.4, 12.3, 6.6} , { 7.7, 8.8, 9.9} } ;
double[]  b = { 12.5, 32.2 , 45.6 } ;

// Put values to MATLAB variables
MatlabCSharpAPI.PutMatrixReal("ml_A", A) ;
MatlabCSharpAPI.PutVectorReal("ml_b", b) ;

//MATLAB workspace tasks : Solve Ax = b, and get a vector x and an upper matrix U.
//The problem Ax = b is solved here as the purpose
//of showing the use of the MATLAB workspace in a C# function.
MatlabCSharpAPI.Execute(" ml_vectorX = mldivide(ml_A, ml_b) ; ") ;
MatlabCSharpAPI.Execute(" [ml_L, ml_U, ml_P] = lu( ml_A )   ; ") ;

//Get results from MATLAB workspace
Array vectorX = new double [b.Length] ;
MatlabCSharpAPI.GetVectorReal("ml_vectorX", ref vectorX) ;

int row, col;
row = A.GetUpperBound(0) - A.GetLowerBound(0) + 1;
col = A.GetUpperBound(1) - A.GetLowerBound(1) + 1;

Array matrixU = Array.CreateInstance(typeof(double), row, col) ;
MatlabCSharpAPI.GetMatrixReal("ml_U", ref matrixU) ;

// Print out
Console.WriteLine("Linear system: result of the vector solution") ;
MatlabCSharpAPI.PrintVector(vectorX) ;

Console.WriteLine() ;
Console.WriteLine("Linear system: result of the U matrix") ;
MatlabCSharpAPI.PrintMatrix(matrixU) ;

// Or Convert to a double array
Console.WriteLine("Section linear solve: Convert to double array: ") ;
double[] db_vectorX = new double [b.Length] ;
vectorX.CopyTo(db_vectorX, 0) ;
```

```
  MatlabCSharpAPI.PrintVector(db_vectorX) ;

  // Or convert to a double matrix
  Console.WriteLine("Section Linear solve: Convert to double matrix") ;
  double[,] db_matrixU = new double[row, col] ;
  Array.Copy(matrixU, db_matrixU, row*col) ;
  MatlabCSharpAPI.PrintMatrix(db_matrixU) ;
}

/* *************************** */
public void PlusRealVectors()
{
  double [] vecAReal = {  0.1,  1.1,   11.1 } ;
  double [] vecBReal = {  0.2,  2.2,   22.2 } ;

  MatlabCSharpAPI.PutVectorReal("ml_vecA", vecAReal) ;
  MatlabCSharpAPI.PutVectorReal("ml_vecB", vecBReal) ;

  MatlabCSharpAPI.Execute( " ml_vecC = ml_vecA + ml_vecB ; " ) ;

  Array vectorC_Real = new double [vecAReal.Length] ;
  MatlabCSharpAPI.GetVectorReal("ml_vecC", ref vectorC_Real);

  // Print out
  Console.WriteLine("Result of a real vector") ;
  MatlabCSharpAPI.PrintVector(vectorC_Real) ;

  // Or convert to a double array
  double[] db_VectorC_Real = new double [vecAReal.Length] ;
  vectorC_Real.CopyTo(db_VectorC_Real,0) ;

  Console.WriteLine("Section real vector: Convert vector to a double array:") ;
  MatlabCSharpAPI.PrintVector(db_VectorC_Real) ;
}

/* *************************** */
```

```
public void PlusComplexVectors()
{
  double [] vecAReal = {  0.1,  1.1,   11.1 } ;
  double [] vecAImag = {100.1, 101.1, 111.1 } ;

  double [] vecBReal = {  0.2,  2.2,   22.2 } ;
  double [] vecBImag = {200.2, 202.2, 222.2 } ;

  MatlabCSharpAPI.PutVectorComplex("ml_vecAComp", vecAReal, vecAImag) ;
  MatlabCSharpAPI.PutVectorComplex("ml_vecBComp", vecBReal, vecBImag) ;

  MatlabCSharpAPI.Execute( " ml_vecCComp = ml_vecAComp + ml_vecBComp ; " ) ;

  Array vectorC_Real = new double [vecAReal.Length] ;
  Array vectorC_Imag = new double [vecAImag.Length] ;

  MatlabCSharpAPI.GetVectorComplex("ml_vecCComp", ref vectorC_Real, ref vectorC_Imag);

  // Print out
  Console.WriteLine("Result of a complex vector:") ;
  Console.WriteLine("Real vector:") ;
  MatlabCSharpAPI.PrintVector(vectorC_Real) ;

  Console.WriteLine("Imag vector:") ;
  MatlabCSharpAPI.PrintVector(vectorC_Imag) ;

  // Convert to a double array
  double[] db_VectorC_Real = new double [vecAReal.Length] ;
  double[] db_vectorC_Imag = new double [vecAImag.Length] ;

  vectorC_Real.CopyTo(db_VectorC_Real,0) ;
  vectorC_Imag.CopyTo(db_vectorC_Imag,0) ;

  Console.WriteLine("Section complex vector: Convert vector to a double array:") ;
  MatlabCSharpAPI.PrintVector(db_VectorC_Real) ;
  MatlabCSharpAPI.PrintVector(db_vectorC_Imag) ;
```

```
    }

    /* ****************************** */
    public void API_Plot()
    {
      double[]X = {0, 50, 100, 150, 200, 250, 300, 350, 400, 450, 500, 550, 600} ;

      double[]Y = {0.4000, 0.2426, 0.1472, 0.0893, 0.0541, 0.0328, 0.0199,
             0.0121, 0.0073, 0.0044, 0.0027, 0.0016,  0.0010 } ;

      String myColor = "blue";

      MatlabCSharpAPI.PutVectorReal("ml_X", X) ;
      MatlabCSharpAPI.PutVectorReal("ml_Y", Y) ;

      MatlabCSharpAPI.PutString("ml_plotColor", myColor) ;

      MatlabCSharpAPI.Execute( "plot(ml_X, ml_Y, ml_plotColor); ") ;
      MatlabCSharpAPI.Execute( "grid on; ") ;

      Console.WriteLine("Hit Enter key to close the figure and continue.") ;
      Console.ReadLine() ; // Keep figure staying
    }

/* ****************************** */
/* ****************************** */
/* ****************************** */
  }
}
```

end code

Utility file ***MatlabCSharpAPIVer4p11.cs***.

Listing code

```
using System;
```

```
// For MATLAB Version 7.9 (2009b) and MATLAB Compiler 4.11
namespace UtilityMatlabCSharpAPIVer4p11
{
  class MatlabCSharpAPI
  {
    // Instantiate MATLAB Engine Interface through COM
    public static MLApp.MLAppClass matlab = new MLApp.MLAppClass();

    /* ***************************** */
    /* ********** I. Scalar ******** */
    /* ***************************** */
    public static void PutScalarReal(String ml_a, double a)
    {
      matlab.PutWorkspaceData(ml_a, "base", (object) a) ;
    }

    /* ******************************** */
    public static void PlusComplexNumbers() {

      // 1. Set number
      double aReal = 1.1    ;
      double aImag = 101.1  ;

      double bReal = 2.2    ;
      double bImag = 202.2  ;

      // 2. Perform tasks in Matlab Workspace
      matlab.PutWorkspaceData("ml_aReal", "base", aReal) ;
      matlab.PutWorkspaceData("ml_aImag", "base", aImag) ;
      matlab.Execute(" ml_aComp = complex(ml_aReal, ml_aImag) ; ") ;

      matlab.PutWorkspaceData("ml_bReal", "base", bReal) ;
      matlab.PutWorkspaceData("ml_bImag", "base", bImag) ;
      matlab.Execute(" ml_bComp = complex(ml_bReal, ml_bImag) ; ") ;

      matlab.Execute(" ml_cComp = ml_aComp + ml_bComp ; " ) ;
      matlab.Execute(" ml_cReal = real(ml_cComp) ; ") ;
      matlab.Execute(" ml_cImag = imag(ml_cComp) ; ") ;

      // 3. Get data from Matlab Workspace
      object obj_cReal, obj_cImag ;
      matlab.GetWorkspaceData("ml_cReal", "base", out obj_cReal) ;
      matlab.GetWorkspaceData("ml_cImag", "base", out obj_cImag) ;

      // 4. Print out
      double cReal = 0 ;
      double cImag = 0 ;

      cReal = Convert.ToDouble(obj_cReal) ;
      cImag = Convert.ToDouble(obj_cImag) ;

      Console.WriteLine("Result of a complex number :") ;
      Console.WriteLine("\t Real Part: {0}", cReal.ToString() ) ;
      Console.WriteLine("\t Imag Part: {0}", cImag.ToString() ) ;

    }

    /* ***************************** */
    public static void GetScalarReal(String ml_a, ref double a)
    {
      object obj_a ;
      matlab.GetWorkspaceData(ml_a, "base", out obj_a) ;

      a = Convert.ToDouble(obj_a) ;
    }

    /* ****************************** */
    /* *************II. Vector ******** */
    /* ****************************** */
    public static void PutVectorReal(String ml_vector, Array vector)
    {
      Array pi = new double [vector.Length] ;
      Array.Clear(pi, 0, pi.Length);

      matlab.PutFullMatrix(ml_vector, "base", vector, pi) ;
    }

    /* ******************************** */
    public static void PutVectorComplex(String ml_vectorComplex,
                                            Array vectorReal, Array vectorImag)
    {
      matlab.PutFullMatrix(ml_vectorComplex, "base", vectorReal, vectorImag) ;
    }

    /* ******************************** */
    public static void GetVectorReal(String ml_vector, ref Array vector)
    {
      Array pi = new double [vector.Length];
      matlab.GetFullMatrix(ml_vector, "base", ref vector, ref pi) ;
    }

    /* ******************************** */
    public static void GetVectorComplex(String ml_vectorComplex, ref Array vectorReal,
                                                        ref Array vectorImag)
    {
      matlab.GetFullMatrix(ml_vectorComplex, "base", ref vectorReal, ref vectorImag) ;
    }

    /* ******************************** */
    /* ********** III. Matrix ********* */
    /* ******************************** */
    public static void PutMatrixReal(String ml_matrix, Array matrix)
    {
    int row, col;
    row = matrix.GetUpperBound(0) - matrix.GetLowerBound(0) + 1;
    col = matrix.GetUpperBound(1) - matrix.GetLowerBound(1) + 1;
```

```
  Array pi = Array.CreateInstance( typeof(double), row, col ) ;
  Array.Clear(pi, 0, pi.Length) ;

  matlab.PutFullMatrix(ml_matrix, "base", matrix, pi) ;
}

/* ******************************** */
public static void PutMatrixComplex(String ml_matrixComplex, Array matrixReal,
                                                              Array matrixImag)
{
  matlab.PutFullMatrix(ml_matrixComplex, "base", matrixReal, matrixImag) ;
}

 /* ******************************** */
public static void GetMatrixReal(String ml_matrix, ref Array matrix)
{
int row, col;
row = matrix.GetUpperBound(0) - matrix.GetLowerBound(0) + 1;
col = matrix.GetUpperBound(1) - matrix.GetLowerBound(1) + 1;

  Array pi = Array.CreateInstance( typeof(double), row, col ) ;
  Array.Clear(pi, 0, pi.Length) ;

  matlab.GetFullMatrix(ml_matrix, "base", ref matrix, ref pi) ;
}

/* ******************************** */
public static void GetMatrixComplex(String ml_matrixComplex, ref Array matrixReal,
                                                              ref Array matrixImag)
{
  matlab.GetFullMatrix(ml_matrixComplex, "base", ref  matrixReal, ref matrixImag) ;
}

/* ******************************** */
/* ************* Print out ******* */
/* ******************************** */
```

```
public static void PrintVector(Array vector)
{
  int i, row ;
  row = vector.GetUpperBound(0) - vector.GetLowerBound(0) + 1;

  for (i=0; i<row; i++)
  {
    Console.Write("{0} \t", (String) vector.GetValue(i).ToString() ) ;
    Console.WriteLine() ;
  }
}

/* ******************************** */
public static void PrintMatrix(Array matrix)
{
int i, j, row, col ;
row = matrix.GetUpperBound(0) - matrix.GetLowerBound(0) + 1;
col = matrix.GetUpperBound(1) - matrix.GetLowerBound(1) + 1;

for (i=0; i<row; i++)
{
for (j=0; j<col; j++)
{
  Console.Write("{0} \t", (String)matrix.GetValue(i,j).ToString() ) ;
}
Console.WriteLine() ;
}
}

/* ******************************** */
public static void Execute(String commandName)
{
  matlab.Execute(commandName) ;
}

/* ******************************** */
```

```
    public static void PutString(String ml_str, String str)
    {
      matlab.PutCharArray(ml_str, "base", str) ;
    }

/* ******************************** */
/* ******************************** */
/* ******************************** */
  }
}
```

end code

Bibliography

[1] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C.* Cambridge University Press, 1992.

[2] Inc. The MathWork. MATLAB compiler version 4, user guide. The Language of Technical Computing. URL address:
www.mathworks.com/access/helpdesk/help/pdf_doc/compiler/compiler.pdf.

[3] Inc. The MathWork. MATLAB Builder for .Net, The Language of Technical Computing. URL address:
www.mathworks.com/access/helpdesk/help/pdf_doc/dotnetbuilder/dotnetbuilder.pdf.

[4] Inc. The MathWork. MATLAB Function Reference. Volume 1, The Language of Technical Computing. URL address:
www.mathworks.com/access/helpdesk/help/pdf_doc/matlab/refbook.pdf.

[5] Inc. The MathWork. MATLAB Function Reference. Volume 2, The Language of Technical Computing. URL address:
www.mathworks.com/access/helpdesk/help/pdf_doc/matlab/refbook2.pdf.

[6] Inc. The MathWork. MATLAB Function Reference. Volume 3, The Language of Technical Computing. URL address:
www.mathworks.com/access/helpdesk/help/pdf_doc/matlab/refbook3.pdf.

Index