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

Preface v

Part I: 
Creating and Using MATLAB Shared Library to Solve Mathematical Problems In C# 1

1 Introduction 3
  1.1 Introduction ........................................... 3
  1.2 Computer Software and Book Features ...................... 4
  1.3 Reference Manuals ..................................... 4

2 Generating a C Shared Library from MATLAB M-Files To Use in Microsoft Visual C# .Net 5
  2.1 Generating a C Shared Library from a MATLAB M-File .......... 5
  2.2 Calling a C Shared Library Function in C# .................. 7
  2.3 Generated Functions from MATLAB Compiler 4 ................ 9
  2.4 Using Functions of a C shared library in C# ............... 10

3 Transfer of Values between C# double and mxArray 11
  3.1 Transfer of Values between C# double and mxArray .......... 11
  3.2 The Utility File CsharpMatlab CompilerVer4.cs ............... 16

4 Matrix Computations 35
  4.1 Matrix Addition ........................................ 37
  4.2 Matrix Subtraction ..................................... 39
  4.3 Matrix Multiplication ................................... 40
  4.4 Matrix Determinant .................................... 41
  4.5 Inverse Matrix ........................................ 42
  4.6 Transpose Matrix ....................................... 42
# Linear System Equations

5.1 Linear System Equations ........................................... 45  
5.2 Sparse Linear System ............................................. 49  
5.3 Tridiagonal System Equations .................................... 52  
5.4 Band Diagonal System Equations .................................. 56

# Ordinary Differential Equations

6.1 First Order ODE ...................................................... 63  
6.2 Second Order ODE ................................................... 69  
  6.2.1 Analysis of second order ODE  ................................ 69  
  6.2.2 Using a second order ODE function .......................... 70

# Integration

7.1 Single Integration ................................................... 74  
7.2 Double-Integration .................................................. 78

# Curve Fitting and Interpolations

8.1 Polynomial Curve Fitting ........................................... 83  
8.2 One-Dimensional Polynomial Interpolation ........................ 87  
8.3 Two-Dimensional Polynomial Interpolation for Grid Points ....... 88

# Roots of Equations

9.1 Roots of Polynomials ............................................... 98  
9.2 The Root of a Nonlinear-Equation ................................ 101

# Fast Fourier Transform

10.1 One-Dimensional Fast Fourier Transform ....................... 105  
10.2 Two-Dimensional Fast Fourier Transform ........................ 109

# Eigenvalues and Eigenvectors

11.1 Eigenvalues and Eigenvectors ................................... 114

## Part II:

**Calling MATLAB Workspace in C# Functions**

Using MATLAB functions in C# Windows Form Applications 119

12 **Calling MATLAB Workspace in C# Functions**  .......................... 121
  12.1 Calling MATLAB Workspace with Input/Output as a Scalar ...... 121
12.2 Calling MATLAB Workspace with Input/Output as a Vector and a Matrix . . . 123
12.3 Generating a MATLAB Graphic from a C# Function . . . . . . . . . . . . . 125

13 Using MATLAB Functions In C# Windows Forms Applications 127

Part III:
Creating and Using COM From MATLAB COM Builder In C# 131

14 Using COM Created from MATLAB COM Builder in C# Applications 135
14.1 Creating COM From MATLAB COM Builder . . . . . . . . . . . . . . . . . . . 135
14.2 Using COM in C# functions . . . . . . . . . . . . . . . . . . . . . . . . . . . 138

15 Using COM Created from MATLAB COM Builder in C# Applications for
Matrix Manipulation & Linear System Equations 143
15.1 Creating COM From MATLAB COM Builder . . . . . . . . . . . . . . . . . . . 143
15.2 Using COM From MATLAB COM Builder To Calculate Matrix Multiplication . 144
15.3 Using COM From MATLAB COM Builder To Solve Linear System Equations . 146

References 149

Index 151
Preface

MATLAB provides the toolbox MATLAB Compiler to handle technical problems between MATLAB and programming languages. The popular task is to support MATLAB built-in functions for computer programmers in development. In recent years, Microsoft Corporation has been developed C# programming language in Microsoft Visual.Net. This C# language is a power language, easy to develop, easy to maintain, and other advanced features, specially in the memory handling. C# is one of programming languages can use the C shared library created from MATLAB M-files. In addition, the MATLAB COM Builder toolbox provides a special feature that the user can create Component Object Model (COM) from MATLAB M-files. The generated COM then can be use in the other programming languages that support COM applications. C# is one of language 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#, 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 call MATLAB built-in functions in the mathematical library created from MATLAB M-files to solve the mathematical problems.
- C# functions call the MATLAB Workspace to perform particular tasks then transfer results from the MATLAB Workspace to C# functions.
- C# functions use COM that is created from MATLAB M-files by using MATLAB COM Builder.

The 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.

LePhan Publishing
December 2004
Part I:
Creating and Using MATLAB Shared Library to Solve Mathematical Problems In C#
Chapter 1

Introduction

1.1 Introduction

MATLAB is a special mathematical software that includes many toolboxes. MATLAB Compiler is the most important toolbox that supports computer programmers. We can use MATLAB Compiler 4.0 to create dll functions from MATLAB M-files. These generated functions will then be called by C# functions. We also can use C# functions to call the MATLAB workspace to perform specific tasks then get results back in C# functions.

In addition, the MATLAB COM Builder toolbox provides a special feature that the user can create Component Object Model (COM) from MATLAB M-files. The generated COM then can be use in the other programming languages that support COM applications. C# is one of language 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#, implements the combination the advances of C# and MATLAB to solve the problems. The features of this book are designed to handle the following projects:

- C# functions call MATLAB built-in functions in the mathematical library created from MATLAB M-files to solve the mathematical problems.

- C# functions call the MATLAB Workspace to perform particular tasks then transfer results from the MATLAB Workspace to C# functions.

- C# functions use COM that is created from MATLAB M-files by using MATLAB COM Builder.
1.2 Computer Software and Book Features

MATLAB Compiler had some versions and there are some changes of its features in different versions. The focus of this book is only on MATLAB Compiler 4. The example codes in this book are developed, compiled, and tested in Windows 2000, Microsoft Visual C# .Net (2002 and 2003), MATLAB 7, MATLAB Compiler 4.0, and MATLAB COM Builer 1.1. These examples are intended to establish common works 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 reuse 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 the utility file to transfer values between C# double and mxArray. This file is very helpful in using MATLAB for C# programming.

1.3 Reference Manuals

In working with MATLAB Compiler 4 you may need more information to help your task. We refer here several manuals from the MATLAB website that you can download for more information.


If you couldn’t find these files at the time you are looking for, The MathWorks Inc. may change URL of these files, but you can find its somewhere in The MathWorks website www.mathworks.com.
Chapter 2

Generating a C Shared Library
from MATLAB M-Files To Use in
Microsoft Visual C# .Net

This chapter describes how to generate a C shared library from MATLAB M-files and use it in Microsoft Visual C# .Net (MSVC# .Net). From M-files, MATLAB Compiler 4 will generate functions in a C shared library as an dll file. This dll file then be used as a library in C# functions.

If we compiler all MATLAB mathematical functions from M-files we will have a mathematical library including all MATLAB functions for C# functions. The main steps of the procedure to generate a C shared library from M-files and use in Microsoft Visual C#. Net are:

1. Write the command to generate an dll-file from the MATLAB M-files.

2. Add the generated files in appropriate directories, and write the code to call the generated functions in the dll-file from C# functions.

The following sections describe how to generate a C share library from an M-file and using in C#. The procedure for another M-file is the same.

2.1 Generating a C Shared Library from a MATLAB M-File

The following are the steps of the procedure to generate a C shared library from a MATLAB M-file.

1. Create an M-file myplus.m as follows:

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

2. Open the command prompt, go to the current directory, and write the command (Fig. 2.1):

```
mcc -B csharedlib:mypluslib myplus.m
```

![Figure 2.1: Command of an dll-file generation](image)

This step will create eight files in the current folder:

- mypluslib.c
- mypluslib.exp
- mypluslib.lib
- mypluslib.ctf
- mypluslib.exports
- mypluslib_mcc_component_data.c
- mypluslib.dll
- mypluslib.h

3. Create a project in Microsoft Visual C#.Net (MSVC#.Net) as a Console Application Project by click File, New, Visual C#, Console Application (see Fig. 2.2).

![Figure 2.2: C# Console Application](image)

4. Copy two files mypluslib.dll and mypluslib.ctf into the folder `bin\Debug` of the project.

5. From the M-function `myplus.m`, MATLAB Compiler 4 has generated an implemental function (see this name in mypluslib.h file),

```
void mlfMyplus(int nargout, mxArray** y, mxArray* a, mxArray* b);
```

with the rule we’ll discuss in Section 2.3. In this function `mlfMyplus(.,.)`, the arguments are:
nargout : number of output (in this case nargout = 1)
y : output variable
a : input variable
b : input variable

Note that the MATLAB Compiler has capitalized M in the function name mlfMyplus.

2.2 Calling a C Shared Library Function in C#

The following is the code in an MSVC# .Net to call the function mlfMyplus(...).

Listing code

```csharp
using System;
using System.Runtime.InteropServices;
using UtilityMatlabCompilerVer4;

namespace MatlabCSharpExample
{
    class Example
    {
        //constructor
        public Example()
        {
            mypluslibInitialize();
        }

        public void CleanUp()
        {
            mypluslibTerminate();
        }

        /* declare dll functions */
        [DllImport( "mypluslib.dll ", CallingConvention = CallingConvention.Cdecl)]
        public static extern void mypluslibInitialize();

        [DllImport( "mypluslib.dll ", CallingConvention = CallingConvention.Cdecl)]
        public static extern void mypluslibTerminate();
    }
}
```
[DllImport("mypluslib.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern void mlfMyplus(int nargout, ref IntPtr y, IntPtr a, IntPtr b);

/* end dll functions */

[STAThread]
static void Main(string[] args)
{
    Console.WriteLine(" ");

    Example obj = new Example();

    Console.WriteLine("Generating a C Shared Library from MATLAB M-Files ");
    Console.Write("\n");

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

    obj.CleanUp();
}

public double CalculatePlus(double a, double b)
{
    /* declare mxArray variables */
    IntPtr mx_a = (IntPtr) null;
    IntPtr mx_b = (IntPtr) null;
    IntPtr mx_y = (IntPtr) null;

    /* convert Cs double to mxArray */
    mx_a = MatlabCSharp.double2mxArray_scalarReal(a);
    mx_b = MatlabCSharp.double2mxArray_scalarReal(b);

    /* call the implemental function */
    mlfMyplus(1, ref mx_y, mx_a, mx_b);

    /* convert back mxArray to Cs double */
    double result = MatlabCSharp.mxArray2double_scalarReal(mx_y);

    return result;
}
Remarks

1. See the code of the class MatlabCSharp in Chapter 3.

2. From the eight generated files, we used only two files mypluslib.dll and mypluslib.ctf.

3. After you built your project, MATLAB also created a folder mypluslib_mcr in the folder Debug.

4. To generate a C shared library from multiple M-files, we just normally add the adding M-files. For example:

   `mcc -B csharedlib:mymathlib myplus.m mymtimes.m`

2.3 Generated Functions from MATLAB Compiler 4

The C function generated by MATLAB Compiler 4 from an M-function has a form that depends on the M-function.

- With an M-functions with no return values, the C function has the form:

  ```c
  void mlf<function-name>(<list_of_input_variables>);
  ```

- With an M-function with at least one return value, the C function has the form:

  ```c
  void mlf<function-name>(int number_of_return_values,
      <list_of_pointer_to_return_variables>,
      <list_of_input_variables>);
  ```

for example:

```c
void mlfMyplus(int nargout, mxArray** y, mxArray* a, mxArray* b)
```

This generated C function has the pattern:
1. the return type is always *void*

2. the first argument, *nargout*, is the number of output variables in the original M-function.

3. the next argument(s) are output variables in the original M-function. These output variables have the type double-pointer to mxArray, for example *mxArray** y*.

4. the next argument(s) are input variables in the original M-function. These variables have the type pointer to mxArray, for example, *mxArray* *a*.

### 2.4 Using Functions of a C shared library in C#

To use a function of a C shared library we need to do:

1. Declaring the initial and terminal functions as the DLL import functions, for example,
   ```
   [DllImport( "mypluslib.dll ", CallingConvention = CallingConvention.Cdecl)]
   public static extern void mypluslibInitialize();
   
   [DllImport( "mypluslib.dll ", CallingConvention = CallingConvention.Cdecl)]
   public static extern void mypluslibTerminate();
   ```

2. Declaring the generated implement functions as the DLL import functions, for example,
   ```
   [DllImport( "mypluslib.dll ", CallingConvention = CallingConvention.Cdecl)]
   public static extern void mlfMyplus(int nargout, ref IntPtr y, IntPtr a, IntPtr b);
   ```

In the above import functions, we have done:

- changing *mxArray* type to *IntPtr* type.
- changing *mxArray** type to *ref IntPtr* type.

*IntPtr* type can be used by languages that support pointers, and as a common means of referring to data between languages that do and do not support pointers. To have more information about *IntPtr* refer to http://msdn.microsoft.com/library.

3. Calling the implement function in C# by transferring values between C# *double* type and mxArray type:

   (a) transferring values of inputs from C# *double* to mxArray inputs.
   
   (b) using these mxArray inputs as inputs in the implement function.
   
   (c) transferring values of outputs from mxArray to C# *double*

For convenience in transferring values between C# *double* and mxArray we write a class *MatlabCSharp* shown in Chapter 3.
Chapter 3

Transfer of Values between
C# double and mxArray

MATLAB Compiler has two principle types, mwArray and mxArray. mxArray is a type used in a C shared library. This chapter shows transfers values of variables between C# double and mxArray.

3.1 Transfer of Values between C# double and mxArray

This section shows how to transfer values between C# double type and mxArray through the example code. We write an utility file CsharpMatlabCompilerVer4.cs for convenience in using the transfer of values between C# double and mxArray. The following example code uses functions in this file CsharpMatlabCompilerVer4.cs to implement the transfers. The CsharpMatlabCompilerVer4.cs file is shown at the end of this chapter.

1. scalar transfer:
   a. real scalar

   double  db_scalar = 1.1   ;
   IntPtr  mx_scalar = (IntPtr) null   ;

   // transfer scalar value from C# double to a real mxArray
   mx_scalar = MatlabCSharp.double2mxArray_scalarReal(db_scalar)   ;

   // transfer scalar value from a real mxArray to a C# double scalar
   double db_scalarReturn = MatlabCSharp.mxArray2double_scalarReal(mx_scalar)   ;
// print out
Console.WriteLine(" db_scalarReturn = {0}" , db_scalarReturn.ToString() ) ;

b. complex scalar

double db_Real = 1.1 ;
double db_Imag = 2.2 ;

// transfer scalar values from C# double to a complex mxArray
IntPtr mx_Complex = (IntPtr) null ;
mx_Complex = MatlabCSharp.double2mxArray_scalarComplex(db_Real, db_Imag) ;

// transfer scalar values from a complex mxArray to C# double scalars
double db_returnReal = 0 ;
double db_returnImag = 0 ;
MatlabCSharp.mxArray2double_scalarComplex(
    mx_Complex, ref db_returnReal, ref db_returnImag) ;

// print out
Console.WriteLine(" db_returnReal = {0}" , db_returnReal.ToString() ) ;
Console.WriteLine(" db_returnImag = {0}" , db_returnImag.ToString() ) ;

2. vector transfer:

In this book, we define that:
   row vector has the number of row = 1
   column vector has the number of column = 1

a. real vector

double[] db_vector = { 1.1 , 2.2 , 3.3 } ;

// transfer vector value from C# double to a real mxArray

    /* row vector has number of row = 1 */
IntPtr mx_vectorRow = (IntPtr) null ;
mx_vectorRow = MatlabCSharp.double2mxArray_vectorRowReal(db_vector) ;
/* column vector has number of column = 1 */
IntPtr mx_vectorCol = (IntPtr) null;
mx_vectorCol = MatlabCSharp.double2mxArray_vectorColumnReal(db_vector);

// transfer vector value from a real mxArray to a C# double vector
double[] db_vectorRowReturn = MatlabCSharp.mxArray2double_vectorReal(mx_vectorRow);
double[] db_vectorColReturn = MatlabCSharp.mxArray2double_vectorReal(mx_vectorCol);

// print out
Console.WriteLine(" Row return vector :");
MatlabCSharp.printVector(db_vectorRowReturn);

Console.WriteLine(" Column return vector : ");
MatlabCSharp.printVector(db_vectorColReturn);

// Note : return vector db_vectorRowReturn or db_vectorColReturn
// from row vector or column mxArray vector is a vector,
// that’s not a matrix (3,1) or (1,3).
// In this case db_vectorRowReturn and db_vectorColReturn are identical

b. complex vector

double[] db_Real = { 1.1, 2.2, 3.3 };
double[] db_Imag = { 4.4, 5.5, 6.6 };
int vectorSize = 3;

// transfer vector values from C# double to a complex mxArray

    /* row vector has number of row = 1 */
IntPtr mx_complexRow = (IntPtr) null;
mx_complexRow = MatlabCSharp.double2mxArray_vectorRowComplex(db_Real, db_Imag);

    /* column vector has number of column = 1 */
IntPtr mx_complexCol = (IntPtr) null;
mx_complexCol = MatlabCSharp.double2mxArray_vectorColumnComplex(db_Real, db_Imag);

// transfer vector values from a complex mxArray to C# double vectors
double[] db_returnRowReal = new double[vectorSize];
double[] db_returnRowImag = new double[vectorSize];
MatlabCSharp.mxArray2double_vectorComplex(
    mx_complexRow, ref db_returnRowReal, ref db_returnRowImag);

Console.WriteLine("1. Real value ");
MatlabCSharp.printVector(db_returnRowReal);

Console.WriteLine("1a. Imaginary value ");
MatlabCSharp.printVector(db_returnRowImag);

//
double[] db_returnColReal = new double [vectorSize];
double[] db_returnColImag = new double [vectorSize];

MatlabCSharp.mxArray2double_vectorComplex(
    mx_complexCol, ref db_returnColReal, ref db_returnColImag);

Console.WriteLine("1. Real value ");
MatlabCSharp.printVector(db_returnColReal);

Console.WriteLine("1a. Imaginary value ");
MatlabCSharp.printVector(db_returnColImag);

3. matrix transfer:
   a. real matrix

double[,] db_A = {{ 1.1, 2.2, 3.3} , {4.4, 5.5, 6.6} , {7.7, 8.8, 9.9} } ;

   // transfer matrix values from C# double to a real matrix mxArray
   IntPtr mx_A = (IntPtr) null;
   mx_A = MatlabCSharp.double2mxArray_matrixReal(db_A);

   // transfer matrix values from a real mxArray to a C# double matrix
   double[,] db_ReturnA = MatlabCSharp.mxArray2double_matrixReal(mx_A);
   MatlabCSharp.printMatrix(db_ReturnA);

   b. complex matrix

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

int row = 3;
int col = 3;

// transfer matrix values from C# double to a complex matrix mxArray
IntPtr mx_complex = (IntPtr) null;
mx_complex = MatlabCSharp.double2mxArray_matrixComplex(db_Real, db_Imag);

// transfer matrix values from a complex mxArray to a C# double matrixes
double[,] db_returnReal = new double [row, col];
double[,] db_returnImag = new double [row, col];

MatlabCSharp.mxArray2double_matrixComplex(
    mx_complex, ref db_returnReal, ref db_returnImag);

// print out
Console.WriteLine("Real matrix : ");
MatlabCSharp.printMatrix(db_returnReal);

Console.WriteLine("Imaginary matrix : ");
MatlabCSharp.printMatrix(db_returnImag);
3.2 The Utility File CsharpMatlabCompilerVer4.cs

using System;
using System.Runtime.InteropServices;
namespace UtilityMatlabCompilerVer4
{

class MatlabCSharp // for MATLAB 7 and MATLAB Compiler 4
{

    [DllImport( "libmx.dll", CallingConvention = CallingConvention.Cdecl)]
    public static extern IntPtr mxCreateDoubleMatrix(int m, int n, mxComplexity flag);

    public enum mxComplexity
    { mxREAL, mxCOMPLEX };

    [DllImport( "libmx.dll", CallingConvention = CallingConvention.Cdecl)]
    public static extern void mxSetPr( IntPtr pa, IntPtr preal );

    [DllImport( "libmx.dll", CallingConvention = CallingConvention.Cdecl)]
    public static extern void mxSetPi( IntPtr pa, IntPtr pimag );

    [DllImport( "libmx.dll", CallingConvention = CallingConvention.Cdecl)]
    public static extern int mxGetM(IntPtr pa);

    [DllImport( "libmx.dll", CallingConvention = CallingConvention.Cdecl)]
    public static extern void mxSetM( IntPtr pa, int m );

    [DllImport( "libmx.dll", CallingConvention = CallingConvention.Cdecl)]
    public static extern int mxGetN( IntPtr pa );

    [DllImport( "libmx.dll", CallingConvention = CallingConvention.Cdecl)]
    public static extern void mxSetN( IntPtr pa, int n );

    [DllImport( "libmx.dll", CallingConvention = CallingConvention.Cdecl)]
    public static extern IntPtr mxGetPr( IntPtr pa );

    [DllImport( "libmx.dll", CallingConvention = CallingConvention.Cdecl)]
    public static extern IntPtr mxGetPi( IntPtr pa );
}
public static extern IntPtr mxDuplicateArray(IntPtr pa);

public static extern IntPtr mxCreateString(String str);

// DLL for MATLAB Engine
public static extern IntPtr engOpen(string startMATLAB);

public static extern int engClose(IntPtr ep);

public static extern int engPutVariable(IntPtr ep, string ML_name, IntPtr mx_name);

public static extern int engEvalString(IntPtr ep, string ML_command);

public static extern IntPtr engGetVariable(IntPtr ep, string ML_resultName);

// DLL for MATLAB Engine -- End

// SCALAR
public static IntPtr double2mxArray_scalarReal(double dbScalar)
{
    // convert a double scalar to an mxArray mxPointer

    double[] dbBuffer ;
    dbBuffer = new double[1];

    IntPtr mxPointer = (IntPtr)0 ;
    IntPtr mxPointerBuffer = (IntPtr)0 ;
    IntPtr mxPointerTemp ;

    try
    {
        // Insert your code here
    }
mxPointerBuffer = mxCreateDoubleMatrix(1, 1, MatlabCSharp.mxComplexity.mxREAL);
}
catch
{
    System.Console.WriteLine("We cannot create a pointer of mxArray. Please check again."");
    return (IntPtr)0;
}

dbBuffer[0] = dbScalar;

mxPointerTemp = Marshal.AllocHGlobal(dbBuffer.Length * Marshal.SizeOf(dbBuffer[0]));
Marshal.Copy(dbBuffer, 0, mxPointerTemp, 1);
mxSetPr(mxPointerBuffer, mxPointerTemp);

mxPointer = mxDuplicateArray(mxPointerBuffer);
Marshal.FreeHGlobal(mxPointerTemp);

return mxPointer ;

}

public static IntPtr double2mxArray_scalarComplex(double dbScalarReal, double dbScalarImag)
{
    // convert double scalars to a complex mxArray mxPointer

double[] dbBufferReal ;
    dbBufferReal = new double[1];

double[] dbBufferImag ;
    dbBufferImag = new double[1];

IntPtr mxPointer = (IntPtr)0 ;
IntPtr mxPointerBuffer = (IntPtr)0 ;

IntPtr mxPointerTempReal ;
IntPtr mxPointerTempImag ;

try
{

mxPointerBuffer = mxCreateDoubleMatrix(1, 1, MatlabChSharp.mxComplexity.mxCOMPLEX);
}
catch
{
    System.Console.WriteLine("We cannot create a pointer of mxArray. Please check again." ) ;
    return (IntPtr)0;
}

dbBufferReal[0] = dbScalarReal;
dbBufferImag[0] = dbScalarImag;

mxPointerTempReal = Marshal.AllocHGlobal(dbBufferReal.Length * Marshal.SizeOf(dbBufferReal[0]));
mxPointerTempImag = Marshal.AllocHGlobal(dbBufferImag.Length * Marshal.SizeOf(dbBufferImag[0]));

Marshal.Copy(dbBufferReal, 0, mxPointerTempReal, 1);
Marshal.Copy(dbBufferImag, 0, mxPointerTempImag, 1);

mxSetPr(mxPointerBuffer, mxPointerTempReal);
mxSetPi(mxPointerBuffer, mxPointerTempImag);

mxPointer = mxDuplicateArray(mxPointerBuffer);

Marshal.FreeHGlobal(mxPointerTempReal);
Marshal.FreeHGlobal(mxPointerTempImag);

return mxPointer ;
}

public static double mxArray2double_scalarReal(IntPtr mxPointer)
{
    // convert a real term of an mxArray mxPointer to a double scalar

double dbScalar ;

IntPtr PointerRealScalar;
double [] dbVector ;
dbVector = new double[1] ;
PointerRealScalar = mxGetPr(mxPointer);

Marshal.Copy(PointerRealScalar, dbVector, 0, 1);

dbScalar = dbVector[0] ;

return dbScalar;

}

public static void mxArray2double_scalarComplex( IntPtr mxPointer, ref double dbReal, ref double dbImag )
{

// convert a complex mxArray mxPointer to double scalars

IntPtr PointerScalarReal;
IntPtr PointerScalarImag = (IntPtr)null;

// for real term
double [] dbVectorReal ;
dbVectorReal = new double[1] ;

PointerScalarReal = mxGetPr(mxPointer);
Marshal.Copy(PointerScalarReal, dbVectorReal, 0, 1);

dbReal = dbVectorReal[0] ;

// for imaginary term

double [] dbVectorImag ;
dbVectorImag = new double[1] ;

try
{


PointerScalarImag = mxGetPi(mxPointer);
Marshal.Copy(PointerScalarImag, dbVectorImag, 0, 1);

dbImag = dbVectorImag[0] ;
}


catch
{  
    dbImag = 0.0;
}

// VECTOR

public static IntPtr double2mxArray_vectorColumnReal(double[] dbVector)
{
    // convert a double vector to an mxArray mxPointer
    // transfer to an mxArray vector with number of col = 1

    int vectorSize;
    vectorSize = dbVector.GetUpperBound(0) - dbVector.GetLowerBound(0) + 1;

    IntPtr mxPointer = (IntPtr)0;
    IntPtr mxPointerBuffer = (IntPtr)0;
    IntPtr mxPointerTemp;

    int row = vectorSize;
    double[] dbVectorBuffer;
    dbVectorBuffer = new double[row];

    try
    {
        mxPointerBuffer = mxCreateDoubleMatrix(row, 1, MatlabCSharp.mxComplexity.mxREAL);
    }
    catch
    {
        System.Console.Write("We cannot create a pointer mxArray. Please check again.");
        return (IntPtr)0;
    }

    mxPointerTemp = Marshal.AllocHGlobal(dbVector.Length * Marshal.SizeOf(dbVector[0]));

    Marshal.Copy(dbVector, 0, mxPointerTemp, dbVector.Length);
    mxSetPr(mxPointerBuffer, mxPointerTemp);
double2mxArray_vectorRowReal(double[] dbVector)
{
    // convert a double vector to an mxArray mxPointer
    // transfer to an mxArray vector with number of row = 1

    int vectorSize;
    vectorSize = dbVector.GetUpperBound(0) - dbVector.GetLowerBound(0) + 1;

    IntPtr mxPointer = (IntPtr)0;
    IntPtr mxPointerBuffer = (IntPtr)0;
    IntPtr mxPointerTemp;

    int col = vectorSize;
    double[] dbVectorBuffer;
    dbVectorBuffer = new double[col];

    try
    {
        mxPointerBuffer = mxCreateDoubleMatrix(1, col, MatlabCSharp.mxComplexity.mxREAL);
    }
    catch
    {
        System.Console.Write("We cannot create a pointer mxArray. Please check again."");
        return (IntPtr)0;
    }

    mxPointerTemp = Marshal.AllocHGlobal(dbVector.Length * Marshal.SizeOf(dbVector[0]));
    Marshal.Copy(dbVector, 0, mxPointerTemp, dbVector.Length);
    mxSetPr(mxPointerBuffer, mxPointerTemp);

    mxPointer = mxDuplicateArray(mxPointerBuffer);
public static IntPtr double2mxArray_vectorColumnComplex(
    double[] dbVectorReal, double[] dbVectorImag)
{
    // convert double vectors to a complex mxArray mxPointer
    // transfer to an mxArray vector with number of col = 1

    int vectorSize;
    vectorSize = dbVectorReal.GetUpperBound(0) - dbVectorReal.GetLowerBound(0) + 1;

    IntPtr mxPointer       = (IntPtr)0;
    IntPtr mxPointerBuffer = (IntPtr)0;

    IntPtr mxPointerTempReal;
    IntPtr mxPointerTempImag;

    int row = vectorSize;
    double[] dbVectorBuffer;
    dbVectorBuffer = new double[row];

    try
    {
        mxPointerBuffer = mxCreateDoubleMatrix(row, 1, MatlabCSharp.mxComplexity.mxCOMPLEX);
    }
    catch
    {
        System.Console.Write("We cannot create a pointer mxArray. Please check again."");
        return (IntPtr)0;
    }

    mxPointerTempReal = Marshal.AllocHGlobal(dbVectorReal.Length * Marshal.SizeOf(dbVectorReal[0]));
    mxPointerTempImag = Marshal.AllocHGlobal(dbVectorImag.Length * Marshal.SizeOf(dbVectorImag[0]));
Marshal.Copy(dbVectorReal, 0, mxPointerTempReal, dbVectorReal.Length);
Marshal.Copy(dbVectorImag, 0, mxPointerTempImag, dbVectorImag.Length);

mxSetPr(mxPointerBuffer, mxPointerTempReal);
mxSetPi(mxPointerBuffer, mxPointerTempImag);

mxPointer = mxDuplicateArray(mxPointerBuffer);

Marshal.FreeHGlobal(mxPointerTempReal);
Marshal.FreeHGlobal(mxPointerTempImag);

return mxPointer;

}

public static IntPtr double2mxArray_vectorRowComplex(double[] dbVectorReal, double[] dbVectorImag)
{
    // convert double vectors to a complex mxArray mxPointer
    // transfer to an mxArray vector with number of row = 1

    int vectorSize ;
    vectorSize = dbVectorReal.GetUpperBound(0)- dbVectorReal.GetLowerBound(0) + 1;

    IntPtr mxPointer = (IntPtr)0;
    IntPtr mxPointerBuffer = (IntPtr)0;

    IntPtr mxPointerTempReal ;
    IntPtr mxPointerTempImag ;

    int col = vectorSize ;
    double[] dbVectorBuffer;
    dbVectorBuffer = new double[col];

    try
    {
        mxPointerBuffer = mxCreateDoubleMatrix(1, col, MatlabCSharp.mxComplexity.mxCOMPLEX);
    }
    catch
    {
        System.Console.Write("We cannot create a pointer mxArray. Please check again."");
    }
return (IntPtr)0;
}

mxPointerTempReal = Marshal.AllocHGlobal(dbVectorReal.Length * Marshal.SizeOf(dbVectorReal[0]));
mxPointerTempImag = Marshal.AllocHGlobal(dbVectorImag.Length * Marshal.SizeOf(dbVectorImag[0]));

Marshal.Copy(dbVectorReal, 0, mxPointerTempReal, dbVectorReal.Length);
Marshal.Copy(dbVectorImag, 0, mxPointerTempImag, dbVectorImag.Length);

mxSetPr(mxPointerBuffer, mxPointerTempReal);
mxSetPi(mxPointerBuffer, mxPointerTempImag);

mxPointer = mxDuplicateArray(mxPointerBuffer);

Marshal.FreeHGlobal(mxPointerTempReal);
Marshal.FreeHGlobal(mxPointerTempImag);

return mxPointer;
}

public static double[] mxArray2double_vectorReal(IntPtr mxPointer)
{
    // convert a real term of an mxArray mxPointer to a double vector

    int vectorSize;

    int row = mxGetM(mxPointer);
    int col = mxGetN(mxPointer);
    if (row >col)  { vectorSize = row ;  }
    else  { vectorSize = col ;  }

    IntPtr PointerVector;

double [] dbVector ;
dbVector = new double[vectorSize] ;

    PointerVector = mxGetPr(mxPointer);
Marshal.Copy(PointerVector, dbVector, 0, vectorSize);

return dbVector;

}

public static void mxArray2double_vectorComplex(
    IntPtr mxPointer, ref double[] dbVectorReal, ref double[] dbVectorImag)
{
    // convert a complex mxArray mxPointer to double vectors
    int vectorSize;

    int row = mxGetM(mxPointer);
    int col = mxGetN(mxPointer);
    if (row > col) { vectorSize = row; }
    else { vectorSize = col; }

    // for real term
    IntPtr PointerVectorReal;
    PointerVectorReal = mxGetPr(mxPointer);
    Marshal.Copy(PointerVectorReal, dbVectorReal, 0, vectorSize);

    // for imaginary term
    IntPtr PointerVectorImag;
    int i;

    try
    {
        PointerVectorImag = mxGetPi(mxPointer);
        Marshal.Copy(PointerVectorImag, dbVectorImag, 0, vectorSize);
    }
    catch
    {
        for (i = 0; i < vectorSize; i++)
        {
            dbVectorImag[i] = 0.0;
        }
    }
public static IntPtr double2mxArray_matrixReal(double[,] dbMatrix) {
    // convert a double matrix to an mxArray mxPointer

    int i, j, row, col, index;
    row = dbMatrix.GetUpperBound(0) - dbMatrix.GetLowerBound(0) + 1;
    col = dbMatrix.GetUpperBound(1) - dbMatrix.GetLowerBound(1) + 1;

    IntPtr mxPointer = (IntPtr)0;
    IntPtr mxPointerBuffer = (IntPtr)0;
    IntPtr mxPointerTemp;

    try
    {
        mxPointerBuffer = mxCreateDoubleMatrix(row, col, MatlabCSharp.mxComplexity.mxREAL);
    }
    catch
    {
        System.Console.Write("We cannot create a pointer mxArray. Please check again."");
        return (IntPtr)0;
    }

    double[] dbVectorBuffer;
    dbVectorBuffer = new double[row * col];

    // transfer double matrix to double vector
    for (i = 0; i < row; i++)
    {
        for (j = 0; j < col; j++)
        {
            index = j * row + i;
            dbVectorBuffer[index] = dbMatrix[i, j];
        }
    }
}
public static IntPtr double2mxArray_matrixComplex(double[,] dbMatrixReal, double[,] dbMatrixImag)
{
    // convert a double matrixes to a complex mxArray mxPointer

    int i, j, row, col, index;
    row = dbMatrixReal.GetUpperBound(0) - dbMatrixReal.GetLowerBound(0) + 1;
    col = dbMatrixReal.GetUpperBound(1) - dbMatrixReal.GetLowerBound(1) + 1;

    IntPtr mxPointer = (IntPtr)0;
    IntPtr mxPointerBuffer = (IntPtr)0;

    IntPtr mxPointerTempReal;
    IntPtr mxPointerTempImag;

    try
    {
        mxPointerBuffer = mxCreateDoubleMatrix(row, col, MatlabCSharp.mxComplexity.mxCOMPLEX);
    }
    catch
    {
        System.Console.WriteLine("We cannot create a pointer mxArray. Please check again.");
        return (IntPtr)0;
    }

    return mxPointer;
}
double[] dbVectorBufferReal;
dbVectorBufferReal = new double[ row*col ];

double[] dbVectorBufferImag;
dbVectorBufferImag = new double[ row*col ];

// transfer double matrix to double vector
for( i=0; i<row; i++)
{
    for( j=0; j<col; j++)
    {
        index = j*row + i;
        dbVectorBufferReal[index] = dbMatrixReal[i, j];
        dbVectorBufferImag[index] = dbMatrixImag[i, j];
    }
}

mxPointerTempReal = Marshal.AllocHGlobal(
    dbVectorBufferReal.Length * Marshal.SizeOf(dbVectorBufferReal[0]));

mxPointerTempImag = Marshal.AllocHGlobal(
    dbVectorBufferImag.Length * Marshal.SizeOf(dbVectorBufferImag[0]));

Marshal.Copy(dbVectorBufferReal, 0, mxPointerTempReal, dbVectorBufferReal.Length);
Marshal.Copy(dbVectorBufferImag, 0, mxPointerTempImag, dbVectorBufferImag.Length);

mxSetPr(mxPointerBuffer, mxPointerTempReal);
mxSetPi(mxPointerBuffer, mxPointerTempImag);

mxPointer = mxDuplicateArray(mxPointerBuffer);

return mxPointer;
```csharp
public static double[,] mxArray2double_matrixReal(IntPtr mxPointer)
{
    // convert a real term of an mxArray mxPointer to a double matrix

    int i, j, index, row, col;
    row = mxGetM(mxPointer);
    col = mxGetN(mxPointer);

    IntPtr PointerDbArray;

    double[,] dbRealMatrix;
    dbRealMatrix = new double[row, col];

    PointerDbArray = mxGetPr(mxPointer);

    double[] dbBufferVector;
    dbBufferVector = new double[row*col];

    Marshal.Copy(PointerDbArray, dbBufferVector, 0, row*col);

    // convert vector to matrix
    for (i = 0; i < row; i++)
    {
        for (j = 0; j < col; j++)
        {
            index = j*row + i;
            dbRealMatrix[i, j] = dbBufferVector[index];
        }
    }

    return dbRealMatrix;
}

public static void mxArray2double_matrixComplex(
    IntPtr mxPointer, ref double[,] dbMatrixReal, ref double[,] dbMatrixImag)
{

    // code for matrix complex conversion
}
```
// convert a complex mxArray mxArray to double matrixes

int i, j, index, row, col;
row = mxGetM(mxPointer);
col = mxGetN(mxPointer);

// for real term
IntPtr PointerDbArrayReal;

PointerDbArrayReal = mxGetPr(mxPointer);

double[] dbBufferVectorReal;
dbBufferVectorReal = new double[row*col];

Marshal.Copy(PointerDbArrayReal, dbBufferVectorReal, 0, row*col);

// convert vector to matrix
for( i=0; i<row; i++)
{
    for( j=0; j<col; j++)
    {
        index = j*row + i;
        dbMatrixReal[i,j] = dbBufferVectorReal[index];
    }
}

// for imaginary term
IntPtr PointerDbArrayImag;

try
{
    PointerDbArrayImag = mxGetPi(mxPointer);

    double[] dbBufferVectorImag;
    dbBufferVectorImag = new double[row*col];
Marshal.Copy(PointerDbArrayImag, dbBufferVectorImag, 0, row*col);

// convert vector to matrix
for( i=0; i<row; i++)
{
    for( j=0; j<col; j++)
    {
        index = j*row + i ;
        dbMatrixImag[i,j] = dbBufferVectorImag[index];
    }
}

} catch
{
    for( i=0; i<row; i++)
    {
        for( j=0; j<col; j++)
        {
            dbMatrixImag[i,j] = 0.0 ;
        }
    }
}

}  /* ******************************** */
/* ******************************** */
/* ******************************** */
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() ;
}

/* ******************************** */
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 class

} // end class
Chapter 4

Matrix Computations

In this chapter we’ll generate a C shared library `matrixcomputationslib` from common M-files working on matrix computation problems. The generated functions of this library will be used in a MSVC# .Net project to solve matrix computation problems.

Following are steps to create a C shared library `matrixcomputationslib.dll` which will be used to solve matrix computation problems in the next sections. We will write the M-files as shown below. These files are used to generate the C shared library. These files are:

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

```matlab
function y = mydet(a)
    y = det(a) ;
end

function y = myinv(a)
    y = inv(a) ;
end

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

function y = mymtimes(a, b)
    y = a*b ;
end
```
function y = myplus(a, b)

    y = a + b ;

function y = mytranspose( x )

    y = x’ ;

The steps to create a C share library are:

1. Write the command in Windows Command Prompt as follows to create a C shared library

   `matrixcomputationslib`:

   ```
   mcc -B csharedlib:matrixcomputationslib mydet.m myinv.m myminus.m
   mymtimes.m myplus.m mytranspose.m
   ```

2. From this command MATLAB Compiler 4 will create eight files in this C shared library:

   ```
   matrixcomputationslib.c   matrixcomputationslib.ctf
   matrixcomputationslib.dll matrixcomputationslib.exp
   matrixcomputationslib.exports matrixcomputationslib.h
   matrixcomputationslib.lib matrixcomputationslib_mcc_component_data.c
   ```

   Add two files matrixcomputationsliblib.dll and matrixcomputationsliblib.ctf into the project directory `bin\Debug`.

3. In the following sections, we’ll use the following implemental functions in this library to solve
   the common problems in the matrix computations (open the file matrixcomputationslib.h to see the names of these functions):

   ```
   void mlfMydet (int nargout, mxArray** y, mxArray* a);
   void mlfMyinv (int nargout, mxArray** y, mxArray* a);
   void mlfMyminus (int nargout, mxArray** y, mxArray* a, mxArray* b);
   void mlfMymtimes (int nargout, mxArray** y, mxArray* a, mxArray* b);
   void mlfMyplus (int nargout, mxArray** y, mxArray* a, mxArray* b);
   void mlfMytranspose(int nargout, mxArray** y, mxArray* x);
   ```
4.1 Matrix Addition

Problem 1

input Matrix A and B

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

output Finding the matrix addition \( C = A + B \)

The function mlfMyplus(..) in the generated \textit{matrixcomputationslib} library will be used in the following code to solve Problem 1.

Listing code

```csharp
using System;
using System.Runtime.InteropServices;
using UtilityMatlabCompilerVer4;

namespace ExampleUsingMatlab
{
    public class Example
    {
        //constructor
        public Example()
        {
            matrixcomputationslibInitialize();
        }

        public void CleanUp()
        {
            matrixcomputationslibTerminate();
        }
    }
}
```
/* declare dll functions */

[DllImport("matrixcomputationslib.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern void matrixcomputationslibInitialize();

[DllImport("matrixcomputationslib.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern void matrixcomputationslibTerminate();

[DllImport("matrixcomputationslib.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern void mlfMydet(int nargout, ref IntPtr y, IntPtr a);

[DllImport("matrixcomputationslib.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern void mlfMyinv(int nargout, ref IntPtr y, IntPtr a);

[DllImport("matrixcomputationslib.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern void mlfMyminus(int nargout, ref IntPtr y, IntPtr a, IntPtr b);

[DllImport("matrixcomputationslib.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern void mlfMymtimes(int nargout, ref IntPtr y, IntPtr a, IntPtr b);

[DllImport("matrixcomputationslib.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern void mlfMyplus(int nargout, ref IntPtr y, IntPtr a, IntPtr b);

[DllImport("matrixcomputationslib.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern void mlfMytranspose(int nargout, ref IntPtr y, IntPtr x);

/* end dll functions */

[STAThread]
static void Main(string[] args)
{
    Example obj = new Example();

    Console.WriteLine("Matrix Computations");

    Console.WriteLine("Matrix addition");
    obj.addMatrix();

    obj.CleanUp();
}
/* ***************************************** */
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 mxArray variables */
    IntPtr mx_A = (IntPtr)null ;
    IntPtr mx_B = (IntPtr)null ;
    IntPtr mx_C = (IntPtr)null ;

    /* convert Cs matrix to mxArray */
    mx_A = MatlabCSharp.double2mxArray_matrixReal(A) ;
    mx_B = MatlabCSharp.double2mxArray_matrixReal(B) ;

    /* call an implemental function */
    mlfMyplus(1, ref mx_C, mx_A, mx_B );

    /* convert back to Cs double */
    double [,] C = MatlabCSharp.mxArray2double_matrixReal(mx_C) ;

    /* print out */
    MatlabCSharp.printMatrix(C) ;
}
} // end class

}
4.3 Matrix Multiplication

Problem 2

input Matrix A and B

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

output Finding the product matrix \( C = A \ast B \)

This following code describes how to use the function mlfMymtimes(..) in the generated matrixcomputationslib library to calculate the matrix multiplication in Problem 2.

Listing code

```csharp
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 } } ;

    IntPtr mx_A = (IntPtr)null ;
    IntPtr mx_B = (IntPtr)null ;
    IntPtr mx_C = (IntPtr)null ;

    mxArray mxArray_matrixReal(A) ;
    mxArray mxArray_matrixReal(B) ;

    mxArray mxArray_matrixReal(1, ref mx_C, mx_A, mx_B) ;

    mxArray mxArray2double_matrixReal(mx_C) ;

    /* print out */
```
4.4 Matrix Determinant

Problem 3

input Matrix A

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

output Finding the determinant of this matrix A

This following code describes how to use the function mlfMydet(..) in the generated matrixcomputationslib library to find the matrix determinant in Problem 3.

Listing code

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

    /* declare mxArray variables */
    IntPtr mx_A = (IntPtr)null ;
    IntPtr mx_detA = (IntPtr)null ;

    /* convert Cs matrix to mxArray */
    mx_A = MatlabCSharp.double2mxArray_matrixReal(A) ;

    /* call an implemental function */
    mlfMydet(1, ref mx_detA, mx_A ) ;

    /* convert back to Cs double */
    double detA = MatlabCSharp.mxArray2double_scalarReal(mx_detA) ;

    /* print out */
    Console.WriteLine( detA.ToString() ) ;
}
```
4.5 Inverse Matrix

This following code describes how to use the function mlfMyinv(..) in the generated `matrixcomputationslib` library to find a matrix inversion.

Listing code

```csharp
public void determinantMatrix()
{
    double[,] A = {{ 1.1, 2.2, 3.3}, {7.7, 4.4, 9.9} , {4.4, 5.5, 8.8} } ;
    /* declare mxArray variables */
    IntPtr mx_A = (IntPtr)null ;
    IntPtr mx_detA = (IntPtr)null ;

    /* convert Cs matrix to mxArray */
    mx_A = MatlabCSharp.double2mxArray_matrixReal(A) ;

    /* call an implemental function */
    mlfMydet(1, ref mx_detA, mx_A ) ;

    /* convert back to Cs double */
    double detA = MatlabCSharp.mxArray2double_scalarReal(mx_detA) ;

    /* print out */
    Console.WriteLine( detA.ToString() ) ;
}
```

4.6 Transpose Matrix

The function mlfMytranspose(..) in the generated `matrixcomputationslib` library will be used to find a transpose matrix. The code of using this function is identical to the section Inverse Matrix, except at the step *call an implemental function*, we write:

```csharp
/* call an implemental function */
mlfMytranspose(1, ref mx_transposeA, mx_A );
```
Chapter 5

Linear System Equations

The problem of linear system equations involves solving the equation $Ax = b$. This chapter focuses on linear system equations in which the matrix $A$ is a square matrix or a sparse matrix, $A \in \mathbb{R}^{n \times n}$, $x \in \mathbb{R}^{n}$, and $b \in \mathbb{R}^{n}$.

In this chapter we’ll generate a C shared library `linearsytemlib` from common M-files working on problems of linear system equations. The generated functions of this library will be used in MSC# .Net project to solve the linear system equations.

Following are steps to create a C shared library linearsytemlib.dll which will be used to solve problems in the next sections.

We will write the M-files as shown below. These files will be used to generate the shared library.

```plaintext
mydiag.m, myfull.m, mylu.m, mymldivide.m, mymdivide.m,
mysparse.m, and myspdiags.m
```

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

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

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

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

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

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

function S = mysparse(A)
S = sparse(A);

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

The steps to create a C share library are:

1. Write the command in Windows Command Prompt as follows to create a C shared library

   **linear system lib** :
   ```
   mcc -B csharedlib:linear system lib mydiag.m myextractmatrix.m myfull.m mylu.m
   mymldivide.m mymrdivide.m mysparse.m myspdiags.m
   ```

2. MATLAB Compiler 4 will create eight files:

   ```
   linearsystemlib.c  linearsystemlib.ctf  linearsystemlib.dll
   linearsystemlib.exp linearsystemlib.exports linearsystemlib.h
   linearsystemlib.lib linearsystemlib_mcc_component_data.c
   ```
Add two files linearsystemlib.dll and linearsystemlib.ctf into the project directory `bin\Debug`.

3. In the following sections, we’ll use the following implemental functions in this library to solve the common problems in the linear system equations (open the file linearsystemlib.h to see the names of these functions):

```c
void mlfMydiag (int nargout, mxArray** X, mxArray* v, mxArray* k);
void mlfMyextractmatrix(int nargout, mxArray** B, mxArray* A, mxArray* rowa, mxArray* rowb, mxArray* cola, mxArray* colb);

void mlfMyfull (int nargout, mxArray** A, mxArray* S);
void mlfMylu (int nargout, mxArray** L, mxArray** U, mxArray** P, mxArray* A);
void mlfMymldivide(int nargout, mxArray** x, mxArray* A, mxArray* b);
void mlfMymrdivide(int nargout, mxArray** x, mxArray* A, mxArray* b);
void mlfMysparse (int nargout, mxArray** S, mxArray* A);
void mlfMyspdiags (int nargout, mxArray** A, mxArray* B, mxArray* d, mxArray* m, mxArray* n);
```

5.1 Linear System Equations

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

\[
\begin{align*}
a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\
a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\
a_{31}x_1 + a_{32}x_2 + \cdots + a_{3n}x_n &= b_3 \\
\vdots & \quad \vdots \\
a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n 
\end{align*}
\]

(5.1)

**Problem 1**

**input** Matrix $A$ and vector $b$

\[
A = \begin{bmatrix} 1.1 & 5.6 & 3.3 \\ 4.4 & 12.3 & 6.6 \\ 7.7 & 8.8 & 9.9 \end{bmatrix}, \quad b = \begin{bmatrix} 12.5 \\ 32.2 \\ 45.6 \end{bmatrix}
\]
output. Finding the solution $x$ of linear system equations, $Ax = b$

. Finding the lower $L$ and upper $U$ of the matrix $A$ in the decompression method

The functions mlfMymldivide(..) and mlfMylu(..) in the generated linearsystemlib library will be used in the following code to solve Problem 1.

Listing code

```csharp
using System;
using System.Runtime.InteropServices;
using UtilityMatlabCompilerVer4;

namespace MatlabCSharpExample
{
    class Example
    {
        public Example()
        {
            linearsystemlibInitialize();
        }

        public void CleanUp()
        {
            linearsystemlibTerminate();
        }

        /* declare dll functions */

        [DllImport( "linearsystemlib.dll ", CallingConvention = CallingConvention.Cdecl)]
        public static extern void linearsystemlibInitialize();

        [DllImport( "linearsystemlib.dll ", CallingConvention = CallingConvention.Cdecl)]
        public static extern void linearsystemlibTerminate();

        [DllImport( "linearsystemlib.dll ", CallingConvention = CallingConvention.Cdecl)]
        public static extern void mlfMydiag(int nargout, ref IntPtr X, IntPtr v, IntPtr k);

        [DllImport( "linearsystemlib.dll ", CallingConvention = CallingConvention.Cdecl)]
        public static extern void mlfMymldivide(IntPtr A, IntPtr B, ref IntPtr X);
public static extern void mlfMyextractmatrix(int nargout, ref IntPtr B, IntPtr A
, IntPtr rowa, IntPtr rowb
, IntPtr cola, IntPtr colb);

[DllImport( "linearsystemlib.dll ", CallingConvention = CallingConvention.Cdecl)]
public static extern void mlfMyfull(int nargout, ref IntPtr A, IntPtr S);

[DllImport( "linearsystemlib.dll ", CallingConvention = CallingConvention.Cdecl)]
public static extern void mlfMylu(int nargout, ref IntPtr L, ref IntPtr U
, ref IntPtr P, IntPtr A);

[DllImport( "linearsystemlib.dll ", CallingConvention = CallingConvention.Cdecl)]
public static extern void mlfMymldivide(int nargout, ref IntPtr x, IntPtr A, IntPtr b);

[DllImport( "linearsystemlib.dll ", CallingConvention = CallingConvention.Cdecl)]
public static extern void mlfMymrdivide(int nargout, ref IntPtr x, IntPtr A, IntPtr b);

[DllImport( "linearsystemlib.dll ", CallingConvention = CallingConvention.Cdecl)]
public static extern void mlfMysparse(int nargout, ref IntPtr S, IntPtr A);

[DllImport( "linearsystemlib.dll ", CallingConvention = CallingConvention.Cdecl)]
public static extern void mlfMyspdiags(int nargout, ref IntPtr A, IntPtr B
, IntPtr d, IntPtr m, IntPtr n);

/* end dll functions */

[STAThread]
static void Main(string[] args)
{
    Console.WriteLine("Linear System Equations") ;

    Example obj = new Example() ;
    obj.LinearSystemEquations() ;

    Console.WriteLine() ;
    obj.LU_decompression() ;

    obj.CleanUp() ;
public void LinearSystemEquations()
{
    /* 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};

    /* declare mxArray variables */
    IntPtr mx_A = (IntPtr)null;
    IntPtr mx_b = (IntPtr)null;
    IntPtr mx_x = (IntPtr)null;

    /* convert Cs matrix to mxArray */
    mx_A = MatlabCSharp.double2mxArray_matrixReal(db_A);
    mx_b = MatlabCSharp.double2mxArray_vectorColumnReal(db_vectorb);

    /* call an implemental function */
    mlfMymldivide(1, ref mx_x, mx_A, mx_b);

    /* convert back to Cs double */
    double[] db_vectorx = MatlabCSharp.mxArray2double_vectorReal(mx_x);

    // print out
    MatlabCSharp.printVector(db_vectorx);
}

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}
    };

    /* declare mxArray variables */
    IntPtr mx_A = (IntPtr)null;
    IntPtr mx_b = (IntPtr)null;
    IntPtr mx_x = (IntPtr)null;

    /* convert Cs matrix to mxArray */
    mx_A = MatlabCSharp.double2mxArray_matrixReal(db_A);
    mx_b = MatlabCSharp.double2mxArray_vectorColumnReal(db_vectorb);

    /* call an implemental function */
    mlfMymldivide(1, ref mx_x, mx_A, mx_b);

    /* convert back to Cs double */
    double[] db_vectorx = MatlabCSharp.mxArray2double_vectorReal(mx_x);

    // print out
    MatlabCSharp.printVector(db_vectorx);
}
The code of the class MatlabCSharp in Chapter 3.

5.2 Sparse Linear System

The sparse linear system is a common system created to solve a 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.

**Problem 2**

**input**  Sparse matrix \( A \) and vector \( b \)

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

**output**  Finding the solution \( x \) of the sparse system equations \( Ax = b \)

The common steps to solve a sparse linear system using functions in the C shared library *matrixcomputations* are:

1. Establishing the spare matrix by using the function mlfMysparse(.

2. Solving the sparse system by using the function mlfMymldivide(.

The following is the code to solve Problem 2 by using the functions, mlfMysparse(.

Listing code

```java
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[,] db_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[] db_vectorb = {11.1, 0, 22.2, 0, 33.3};

/* declare mxArray variables */
IntPtr mx_A = (IntPtr)null;
IntPtr mx_b = (IntPtr)null;
IntPtr mx_x = (IntPtr)null;

/* note: we create mx_b and mx_x as column vectors */

/* convert Cs matrix to mxArray */
mx_A = MatlabCSharp.double2mxArray_matrixReal(db_A);
mx_b = MatlabCSharp.double2mxArray_vectorColumnReal(db_vectorb);

/* call an implemental function */
mlfMysparse(1, ref mx_A, mx_A);
mlfMysparse(1, ref mx_b, mx_b);
mlfMymldivide(1, ref mx_x, mx_A, mx_b);
mlfMyfull(1, ref mx_x, mx_x);

/* convert back to Cs double */
double[] db_vectorx = MatlabCSharp.mxArray2double_vectorReal(mx_x);

/* print out */
Console.WriteLine("Solution x :");
MatlabCSharp.printVector(db_vectorx);

}
5.3 Tridiagonal System Equations

This section focuses on finding a solution of tridiagonal linear system equations $Ax = d$, as follows:

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

(5.2)

Problem 3

**input**  Matrix $B$ includes column vectors $c$, $a$, and $b$
Vector right-hand side $d$

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

(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. Establishing a buffer matrix $bufferA$ (in Eq. 5.4) from the given matrix $B$ (in Eq. 5.3) by using the following function in the library *linarsytemlib*:

   ```c
   void mlfMyspdiags (int nargout, mxArray** bufferA, mxArray* B, mxArray* d,
   mxArray* m, mxArray* n);
   ```

   This function mlfMyspdiags(..) creates an m-by-n sparse matrix $bufferA$ by taking the
columns of \( B \) and placing them along the diagonals as follows:

\[
\begin{bmatrix}
  c_1 & a_1 & b_1 & 0 & \cdots & 0 & 0 \\
  0 & c_2 & a_2 & b_2 & \cdots & 0 & 0 \\
  0 & 0 & c_3 & a_3 & b_3 & \cdots & 0 \\
  \vdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
  0 & 0 & \cdots & 0 & c_{n-1} & a_{n-1} & b_{n-1} & 0 \\
  0 & 0 & \cdots & 0 & 0 & c_n & a_n & b_n
\end{bmatrix}
\]  

(5.4)

2. Obtaining the matrix \( A \) as in Eq. 5.2 by extracting from the matrix \( \text{bufferA} \)

3. Using the functions in the library \text{linearsytemlib} to solve the tridiagonal linear system equations.

The following is the code to solve Problem 3 by using the functions in the library \text{linearsytemlib}.

Listing code

```java
public void tridiagonalSystem()
{

double [,]B = new double [6,6] ;
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 ;

int one = 1 ;
int two = 2 ;
int seven = 7 ;

int row = 6 ;
int col = 6 ;

int band = 3 ; /* tridiagonal */
int m = row ;
int n = col + (band-1) ;

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

int rowB = row ;
int colB = band ;

/* declare mxArray variables */
IntPtr mx_B = (IntPtr) null ;
IntPtr mx_bufferA = (IntPtr) null ;
IntPtr mx_A = (IntPtr) null ;
IntPtr mx_d = (IntPtr) null ;
IntPtr mx_m = (IntPtr) null ;
IntPtr mx_n = (IntPtr) null ;
IntPtr mx_one = (IntPtr) null ;
IntPtr mx_two = (IntPtr) null ;
IntPtr mx_seven = (IntPtr) null;
IntPtr mx_row = (IntPtr) null;
IntPtr mx_vectord = (IntPtr) null;
IntPtr mx_x = (IntPtr) null;

/* note: we create mx_vectord and mx_x are column vectors
   when using in the following functions */

/* convert Cs matrix to mxArray */
mx_one = MatlabCSharp.double2mxArray_scalarReal(one);
mx_two = MatlabCSharp.double2mxArray_scalarReal(two);
mx_seven = MatlabCSharp.double2mxArray_scalarReal(seven);
mx_row = MatlabCSharp.double2mxArray_scalarReal(row);

mx_m = MatlabCSharp.double2mxArray_scalarReal(m);
mx_n = MatlabCSharp.double2mxArray_scalarReal(n);

mx_d = MatlabCSharp.double2mxArray_vectorColumnReal(d);
mx_B = MatlabCSharp.double2mxArray_matrixReal(B);

mx_vectord = MatlabCSharp.double2mxArray_vectorColumnReal(db_vectord);

/* call an implemental function */

/* create a sparse matrix mx_bufferA from column-matrix B */
mlfMyspdiags(1, ref mx_bufferA, mx_B, mx_d, mx_m, mx_n);
mlfMyfull(1, ref mx_bufferA, mx_bufferA);

/* plot to see */
double[,] db_bufferA = MatlabCSharp.mxArray2double_matrixReal(mx_bufferA);

Console.WriteLine("The buffer matrix A:");
MatlabCSharp.printMatrix(db_bufferA);

/* extract the need-matrix from the buffer matrix,
   from row 1 to row 6 and from column 2 to column 7 */
mlfMyextractmatrix(1, ref mx_A, mx_bufferA, mx_one, mx_row, mx_two, mx_seven);

/* plot to see */
double[,] db_A = MatlabCSharp.mxArray2double_matrixReal(mx_A); 

Console.WriteLine("The need-matrix A:" + mx_A); 
MatlabCSharp.printMatrix(db_A); 

/* solve the tridiagonal system equations */ 
mlfMymldivide(1, ref mx_x, mx_A, mx_vectord); 

/* convert back to Cs double */ 
double[] db_vectorx = MatlabCSharp.mxArray2double_vectorReal(mx_x); 

/* print out */ 
Console.WriteLine("Tridiagonal system solution:" + mx_x); 
MatlabCSharp.printVector(db_vectorx); 
}

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 \( Ax = d \) as follows:

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

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 \( d \)
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. Establishing a buffer matrix \( \text{bufferA} \) (in Eq. 5.7) from given matrix \( B \) (in Eq. 5.6) by using a function in the library.

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

(5.7)

2. Obtaining the matrix A as in Eq. 5.5 by extracting from the matrix \( \text{bufferA} \).

3. Using the functions in the library to solve the band system diagonal equations.

The following is the code to solve Problem 4 by using the functions in the library \texttt{linearsystemlib}.

Listing code

```java
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 ;

int i ;
int one = 1 ;
int two = 2 ;
int seven = 7 ;
int row = 6 ;
int col = 6 ;

int band = 4 ; /* band width */
int m = row ;
int n = col + (band-1) ;

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

int rowB = row ;
int colB = band ;

/* declare mxArray variables */
IntPtr mx_B = (IntPtr) null ;
IntPtr mx_bufferA = (IntPtr) null ;
IntPtr mx_A = (IntPtr) null ;
IntPtr mx_d = (IntPtr) null ;
IntPtr mx_m = (IntPtr) null ;
IntPtr mx_n = (IntPtr) null ;
IntPtr mx_one = (IntPtr) null ;
IntPtr mx_two = (IntPtr) null ;
IntPtr mx_seven = (IntPtr) null ;
IntPtr mx_row = (IntPtr) null ;
IntPtr mx_vectord = (IntPtr) null ;
IntPtr mx_x = (IntPtr) null ;

/* note: we create mx_vectord and mx_x are column vectors when using in the following functions */
/* convert Cs double to mxArray */
mx_one = MatlabCSharp.double2mxArray_scalarReal(one) ;
mx_two = MatlabCSharp.double2mxArray_scalarReal(two) ;
mx_seven = MatlabCSharp.double2mxArray_scalarReal(seven) ;
mx_row = MatlabCSharp.double2mxArray_scalarReal(row) ;

mx_m = MatlabCSharp.double2mxArray_scalarReal(m) ;
mx_n = MatlabCSharp.double2mxArray_scalarReal(n) ;
mx_d = MatlabCSharp.double2mxArray_vectorColumnReal(d);
mx_B = MatlabCSharp.double2mxArray_matrixReal(B);

mx_vectord = MatlabCSharp.double2mxArray_vectorColumnReal(db_vectord);

/* step 4 : call an implemental function */

/* create a sparse matrix, size mxn, from column-matrix B */
mlfMyspdiags(1, ref mx_bufferA, mx_B, mx_d, mx_m, mx_n);
mlfMyfull(1, ref mx_bufferA, mx_bufferA);

/* plot to see */
double[,] db_bufferA = MatlabCSharp mxArray2double_matrixReal(mx_bufferA);

Console.WriteLine("The buffer band-matrix A:");
MatlabCSharp.printMatrix(db_bufferA);

/* extract the need-matrix A from the buffer matrix,
from row 1 to row 6 and from column 2 to column 7 */
mlfMyextractmatrix(1, ref mx_A, mx_bufferA, mx_one, mx_row, mx_two, mx_seven);

/* plot to see */
double[,] db_A = MatlabCSharp mxArray2double_matrixReal(mx_A);

Console.WriteLine("The band-need-matrix A:");
MatlabCSharp.printMatrix(db_A);

/* solve the system equations */
mlfMymldivide(1, ref mx_x, mx_A, mx_vectord);

/* convert back to Cs double */
double[] db_vectorx = MatlabCSharp mxArray2double_vectorReal(mx_x);

/* convert back to Cs double */
Console.WriteLine("Band matrix system solution:");
MatlabCSharp.printVector(db_vectorx);

}
Chapter 6

Ordinary Differential Equations

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

The major MATLAB function of M-files used to generate the library to solve ODE problems is _ode45(_)_. There are another functions, _ode23(_)_, _ode113(_)_, _ode15s(_)_, and _ode23s(_)_, can be used to solve the ODE problems. Therefore you can choose a function with options that satisfies your problem requirements. For more information on these functions, refer to the manual [5].

Following are steps to create a C shared library _odelib.dll_ which will be used to solve ODE problems in the next sections.

We will write the M-files as shown below. These functions will be used to generate the C library.

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

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

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

```matlab
myode45secondorder.m, yoursecondfunc.m
```

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

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

```matlab
myode45secondorder.m, yoursecondfunc.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) ;

function [t, y] = myode45secondorder(strfunc, tspan, y0)
[t,y] = ode45(@yoursecondfunc, tspan, y0, [], strfunc) ;

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 steps to create a C share library are:

1. Write the command in Windows Command Prompt as follows to generate the C shared library *odelib*:
   
   ```
   mcc -B csharedlib:odelib myode45firstorder.m myode45secondorder.m
   ```

2. MATLAB Compiler 4.0 will create eight files of this C shared library:

   - `odelib.c`
   - `odelib.ctf`
   - `odelib.dll`
   - `odelib.exp`
   - `odelib.exports`
   - `odelib.h`
   - `odelib.lib`
   - `odelib_mcc_component_data.c`

   Add two files `odelib.dll` and `odelib.ctf` into the project directory `bin\Debug`.

3. In the following sections, we’ll use the following implemental functions in this library to solve the common ODE problems (open the file `odelib.h` to see the names of these functions):

   ```
   void mlfMyode45firstorder(int nargout, mxArray** t, mxArray** y
   , mxArray** lengthtime, mxArray* strfunc
   , mxArray* tspan, mxArray* y0);
   ```

   ```
   void mlfMyode45secondorder(int nargout, mxArray** t, mxArray** y
   , mxArray** lengthtime, mxArray* strfunc
   , mxArray* tspan, mxArray* y0);
   ```

### 6.1 First Order ODE

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

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

with initial condition :

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

**Note:**

1. In 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 $\mathbf{t}[]$ contains the values of time $t$, $t \in [a, b]$

• array $\mathbf{y}[]$ 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 you do not provide a time step.

3. The time step can be set by providing $\mathbf{tspan} = [t_0, t_2, \ldots, t_n]$ 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 $\mathbf{tspan}$.

4. The ODE function is passed as an expression string to the generated function mlfMyode45firstorder(..) or 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 mlfMyode45firstorder(..) in the generated odelib library with the time step set to default.

Listing code

```csharp
using System;
using System.Runtime.InteropServices;
using UtilityMatlabCompilerVer4;

namespace MatlabCSharpExample
{
    class Example
    {

        //constructor
        public Example()
        {
            odelibInitialize();
        }

        public void CleanUp()
        {
            odelibTerminate();
        }
    }
}
```
/* declare dll functions */

[DllImport( "odelib.dll ", CallingConvention = CallingConvention.Cdecl)]
public static extern void odelibInitialize();

[DllImport( "odelib.dll ", CallingConvention = CallingConvention.Cdecl)]
public static extern void odelibTerminate();

[DllImport( "odelib.dll ", CallingConvention = CallingConvention.Cdecl)]
public static extern void mlfMyode45firstorder(int nargout, ref IntPtr t
, ref IntPtr y, IntPtr strfunc
, IntPtr tspan, IntPtr y0);

[DllImport( "odelib.dll ", CallingConvention = CallingConvention.Cdecl)]
public static extern void mlfMyode45secondorder(int nargout, ref IntPtr t
, ref IntPtr y, IntPtr strfunc
, IntPtr tspan, IntPtr y0);

/* end dll functions */

[STAThread]
static void Main(string[] args)
{
    Example obj = new Example() ;
    Console.WriteLine("ODE problems.");

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

    obj.CleanUp();
}

/copyleft*****************************/
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 */
IntPtr mx_strfunc = MatlabCSharp.mxCreateString(strfunc) ;
IntPtr mx_y0 = (IntPtr) null ;
IntPtr mx_tspan = (IntPtr) null ;
IntPtr mx_length = (IntPtr) null ;
IntPtr mx_t = (IntPtr) null ;
IntPtr mx_y = (IntPtr) null ;
IntPtr mx_dum01 = (IntPtr) null ;
IntPtr mx_dum02 = (IntPtr) null ;

/* mx_tspan is a column vector when using in the following function */
/* convert Cs double to mxArray */
mx_strfunc = MatlabCSharp.mxCreateString(strfunc) ;

mx_y0 = MatlabCSharp.double2mxArray_scalarReal(db_y0) ;
mx_tspan = MatlabCSharp.double2mxArray_vectorRowReal(db_tspan) ;

/* call an implemental function */
mlfMyode45firstorder(2, ref mx_t, ref mx_y, mx_strfunc, mx_tspan, mx_y0);

/* convert back to Cs double */

double[ ] db_t = MatlabCSharp.mxArray2double_vectorReal(mx_t) ;
double[ ] db_y = MatlabCSharp.mxArray2double_vectorReal(mx_y) ;

Console.WriteLine("The column of time") ;
MatlabCSharp.printVector(db_t) ;

Console.WriteLine("The column of the function values y") ;
MatlabCSharp.printVector(db_y) ;

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

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

with initial condition:

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

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

```java
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

```java
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 */

    /*declare mxArray variables */
    IntPtr mx_strfunc ;
    IntPtr mx_y0 = (IntPtr) null ;
    IntPtr mx_tspan = (IntPtr) null ;
```
IntPtr mx_t = (IntPtr) null;
IntPtr mx_y = (IntPtr) null;

/* mx_tspan is a column vector when using in the following function */
/* convert Cs double to mxArray */
mx_strfunc = MatlabCSharp.mxCreateString(strfunc);

mx_y0 = MatlabCSharp.double2mxArray_scalarReal(db_y0);
mx_tspan = MatlabCSharp.double2mxArray_vectorColumnReal(db_tspan);

/* call an implemental function */
mlfMyode45firstorder(2, ref mx_t, ref mx_y, mx_strfunc, mx_tspan, mx_y);

/* convert back to Cs double */
double[] db_t = MatlabCSharp.mxArray2double_vectorReal(mx_t);
double[] db_y = MatlabCSharp.mxArray2double_vectorReal(mx_y);

Console.WriteLine("The column of time");
MatlabCSharp.printVector(db_t);

Console.WriteLine("The column of the function values y");
MatlabCSharp.printVector(db_y);
6.2 Second Order ODE

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

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

with initial conditions:

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

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:

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

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

and in MATLAB Command Window write:

```matlab
>> 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 from the provided equation:

\[
y = y_1 \\
y' = y_2 \\
y'' = y'_2
\]

Problem 3 then becomes:

\[
y = y_1 \\
y' = y_2 \\
y'' = \cos(3t) + 2y' + 6y \\
= \cos(3t) + 2y_2 + 6y_1
\]

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

3. The function \texttt{mysecondfunc(..)} , which is passed to the ode function \texttt{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
  \[
  \cos(3t) + 2y(2) + 6y(1) ;
  \]

The M-file `yoursecondfunc.m`, with which we use to create a C shared library `odelib` as shown in above, also has a return including two arrays:

• First array is the first derivative of the function \( y \), as follows:
  
  \[
  f0 = \text{inline('yy') ;}
  \]
  
  \[
  dy(1,:) = \text{feval( f0, y(2) ) ;}
  \]

• 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:
  
  \[
  f1 = \text{inline(strfunction)} ;
  \]
  
  \[
  dy(2,:) = \text{feval( f1, t , y(1), y(2) ) ;}
  \]

### 6.2.2 Using a second order ODE function

As explain in above, we have an easy way to use the generated function `mlfMyode45secondorder(..)` in the C shared library `odelib` to solve second order ODE problems by following the steps:

1. Write your 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:
   
   \[
   \text{String strfunc} = "\cos(3*t) + 2*yprime + 6*y" ;
   \]

The following is the code to solve Problem 3 by using the function `mlfMyode45secondorder(..)` in the generated `odelib` library.

Listing code

```java
public void SecondOrder()
{

```
```csharp
/* 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 */
IntPtr mx_strfunc ;
IntPtr mx_ybc = (IntPtr) null ;
IntPtr mx_tspan = (IntPtr) null ;
IntPtr mx_t = (IntPtr) null ;
IntPtr mx_y = (IntPtr) null ;

/* convert Cs double to mxArray */
mx_strfunc = MatlabCSharp.mxCreateString(strfunc) ;
mx_ybc = MatlabCSharp.double2mxArray_vectorRowReal(db_ybc) ;
mx_tspan = MatlabCSharp.double2mxArray_vectorRowReal(db_tspan) ;

/* call an implemental function */
mlfMyode45secondorder(2, ref mx_t, ref mx_y, mx_strfunc, mx_tspan, mx_ybc);

/* convert back to Cs double */
double[] db_t = MatlabCSharp.mxArray2double_vectorReal(mx_t) ;
double[,] db_y = MatlabCSharp.mxArray2double_matrixReal(mx_y) ;

Console.WriteLine("The column of time");
MatlabCSharp.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++)
{

```
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 your ode-function 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 as follow below will directly call the M-file mysecondfunc.m:

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

```matlab
function [t, y] = myotherode(tspan, y0)
    [t,y] = ode45(@mysecondfunc, tspan, y0) ;
```
Chapter 7

Integration

In this chapter we’ll generate a C shared library *integrationlib* 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 project to solve the integral problems.

Following are steps to create a C shared library integrationlib.dll which will be used to solve integral problems in the next sections.

We will write the M-files myquad.m and mydblquad.m. These functions will be used to generate the C shared library.

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

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

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

F = inline(strfunc) ;
dbint = dblquad(F, x1, x2, y1, y2) ;
```
The steps to create a C share library are:

1. Write the command in Windows Command Prompt as follows to create a C shared library
   
   integrationlib.

   mcc -B csharedlib:integrationlib myquad.m mydblquad.m

2. MATLAB Compiler 4.0 will create eight files of this library:

   integrationlib.c      integrationlib.ctf      integrationlib.dll
   integrationlib.exp    integrationlib.exports  integrationlib.h
   integrationlib.lib    integrationlib_mcc_component_data.c

   Add two files integrationlib.dll and integrationlib.ctf into the project directory bin\Debug.

In the following sections, we’ll use the following implemental functions in this library to solve the common problems in the integration (open the file integrationlib.h to see the names of these functions):

```c
void mlfMyquad(int nargout, mxArray** y, mxArray* strfunc
              , mxArray* a, mxArray* b);

void mlfMydblquad(int nargout, mxArray** dbint
                  , mxArray* strfunc, mxArray* x1
                  , mxArray* x2, mxArray* y1, mxArray* y2);
```

### 7.1 Single Integration

**Problem 1**  
Calculate the integration:

\[ I = \int_{0}^{3\pi} \left( \sin(x) + x^2 \right) dx \]

The following is the code to solve Problem 1 by using the functions mlfMyquad(..) in the C shared library integrationlib. The function mlfMyquad(..) used a MATLAB function quad(..) with default as shown in the M-file myquad.m. To use more options see this function quad(..) refer to [5].
using System;
using System.Runtime.InteropServices;
using UtilityMatlabCompilerVer4 ;

namespace MatlabCSharpExample
{
    class Example
    {
        //constructor
        public Example()
        {
            integrationlibInitialize();
        }

        public void CleanUp()
        {
            integrationlibTerminate();
        }

        /* declare dll functions */
        [DllImport( "integrationlib.dll ", CallingConvention = CallingConvention.Cdecl)]
        public static extern void integrationlibInitialize();

        [DllImport( "integrationlib.dll ", CallingConvention = CallingConvention.Cdecl)]
        public static extern void integrationlibTerminate();

        [DllImport( "integrationlib.dll ", CallingConvention = CallingConvention.Cdecl)]
        public static extern void mlfMyquad(int nargout, ref IntPtr y, IntPtr strfunc
           , IntPtr a, IntPtr b);

        [DllImport( "integrationlib.dll ", CallingConvention = CallingConvention.Cdecl)]
        public static extern void mlfMydblquad(int nargout, ref IntPtr dbint
           , IntPtr strfunc, IntPtr x1
           , IntPtr x2, IntPtr y1, IntPtr y2);

        /* end dll functions */

        [STAThread]
static void Main(string[] args)
{
    Console.WriteLine("Single integration:");

    Example obj = new Example();
    obj.singleIntegration();

    Console.WriteLine("Double-integration:");
    obj.dbIntegration();
    obj.CleanUp();
}

/* declare dll functions */
[DllImport( "integrationlib.dll ", CallingConvention = CallingConvention.Cdecl)]
public static extern void integrationlibInitialize();

[DllImport( "integrationlib.dll ", CallingConvention = CallingConvention.Cdecl)]
public static extern void integrationlibTerminate();

[DllImport( "integrationlib.dll ", CallingConvention = CallingConvention.Cdecl)]
public static extern void mlfMyquad(int nargout, ref IntPtr y, IntPtr strfunc,
    IntPtr a, IntPtr b);

[DllImport( "integrationlib.dll ", CallingConvention = CallingConvention.Cdecl)]
public static extern void mlfMydblquad(int nargout, ref IntPtr dbint,
    IntPtr strfunc, IntPtr x1, IntPtr x2, IntPtr y1, IntPtr y2);

/* end dll functions */

[STAThread]
static void Main(string[] args)
{
    Console.WriteLine("Single integration:");

    Example obj = new Example();
    obj.singleIntegration();
Console.WriteLine("Double-integration:");
obj.dbIntegration();
obj.CleanUp();
}

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 */
    IntPtr mx_strfunc;
    IntPtr mx_beginInterval = (IntPtr) null;
    IntPtr mx_endInterval = (IntPtr) null;
    IntPtr mx_y = (IntPtr) null;

    /* convert Cs double to mxArray */
    mx_strfunc = MatlabCSharp.mxCreateString(strfunc);
    mx_beginInterval = MatlabCSharp.double2mxArray_scalarReal(db_beginInterval);
    mx_endInterval = MatlabCSharp.double2mxArray_scalarReal(db_endInterval);

    /* call an implemental function */
    mlfMyquad(1, ref mx_y, mx_strfunc, mx_beginInterval, mx_endInterval);

    /* convert back to Cs double */
    double db_y = MatlabCSharp.mxArray2double_scalarReal(mx_y);
    Console.WriteLine(" I = ", db_y.ToString());
}

end code

Note

1. The generated function mlfMyquad(..) 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

**Problem 2** Calculate the double-integration:

\[
I = \int_0^\pi \int_0^{3\pi} (\sin(x) + x^2 + y^3) \, dx \, dy
\]

The following is the code to solve Problem 2 using the functions mlfMydblquad(..) in the C library `integrationlib`. The function mlfMydblquad(..) 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 [5].

Listing code

```csharp
public void dbIntegration()
{
    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 */
    IntPtr mx_strfunc ;
    IntPtr mx_x1 = (IntPtr) null ;
    IntPtr mx_x2 = (IntPtr) null ;
    IntPtr mx_y1 = (IntPtr) null ;
    IntPtr mx_y2 = (IntPtr) null ;
    IntPtr mx_II = (IntPtr) null ;

    /* convert Cs double to mxArray */
    mx_strfunc = MatlabCSharp.mxCreateString(strfunc) ;
    mx_x1 = MatlabCSharp.double2mxArray_scalarReal (db_x1) ;
    mx_x2 = MatlabCSharp.double2mxArray_scalarReal (db_x2) ;
    mx_y1 = MatlabCSharp.double2mxArray_scalarReal (db_y1) ;
```

mx_y2 = MatlabCSharp.double2mxArray_scalarReal (db_y2) ;

/* call an implemental function */
mlfMydblquad(1, ref mx_II, mx_strfunc, mx_x1, mx_x2, mx_y1, mx_y2) ;

/* convert back to Cs double */
double db_II = MatlabCSharp mxArray2double_scalarReal(mx_II) ;
Console.WriteLine(" II = {0}" , db_II.ToString() ) ;

}
Chapter 8

Curve Fitting and Interpolations

In this chapter we'll generate a C shared library `curvefittinglib` from common M-files working on curve fitting problems. The generated functions of these libraries will be used in MSVC#.Net project to solve common curve fitting problems.

Following are steps to create a C shared library curvefittinglib.dll which will be used to solve curve fitting problems in the next sections. We will write the M-files as shown below. These functions will be used to generate the C shared library.

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

```
function yi = myinterp1(x,y,xi)
    yi = interp1(x,y,xi);

function ZI = myinterp2(X,Y,Z,XI,YI,method)
    ZI = interp2(X,Y,Z,XI,YI,method);

function p = mypolyfit(x,y,n)
    p = polyfit(x,y,n);
```
function y = mypolyval(p,x)

    y = polyval(p,x) ;

function [X,Y] = mymeshgrid(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) ) ;

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

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

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 steps to create a C share library are:

1. Write the command in Windows Command Prompt to create a C shared library
   \textit{curvefittinglib} as follows:

   \texttt{mcc -B csharedlib:curvefittinglib myinterp1.m myinterp2.m mypolyfit.m mypolyval.m mymeshgrid.m mygriddata.m myfinemeshgrid.m}

2. MATLAB Compiler 4.0 will create eight files of this C shared library:
Add two files curvefittinglib.dll and curvefittinglib.ctf into the project directory `bin\Debug`. Note that, in this chapter we use typical functions to solve the problems. There are another options and MATLAB functions to solve the curve fitting problems, refer to [5].

3. In the following sections, we’ll use the following implemental functions in this library to solve the common curve fitting problem (open the file curvefittinglib.h to see the names of these functions):

   ```c
   void mlfMyinterp1(int nargout, mxArray** yi, mxArray* x , mxArray* y, mxArray* xi);
   void mlfMyinterp2(int nargout, mxArray** ZI, mxArray* X , mxArray* Y, mxArray* Z, mxArray* XI , mxArray* YI, mxArray* method);
   void mlfMypolyfit(int nargout, mxArray** p, mxArray* x , mxArray* y, mxArray* n);
   void mlfMypolyval(int nargout, mxArray** y, mxArray* p, mxArray* x);
   void mlfMymeshgrid(int nargout, mxArray** X, mxArray** Y , mxArray* x, mxArray* y);
   void mlfMygriddata(int nargout, mxArray** ZI, mxArray* x, mxArray* y , mxArray* z, mxArray* XI, mxArray* YI);
   ```

8.1 Polynomial Curve Fitting

This section describe 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:

\[
  f(x) = c_1x^n + c_2x^{n-1} + c_3x^{n-2} + \cdots + c_nx + c_{n+1}
\] (8.1)
**Problem 1** There are two arrays $X$ and $Y$ which have a relationship via a function, $y = f(x)$, $x \in X$ and $y \in Y$.

**input**

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:

$$y = c_1 x^3 + c_2 x^2 + c_3 x + c_4$$

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 *mlfMypolyfit(..)* with degree of $n = 3$ to obtain the coefficient array, and use the function *mlfMypolyval(..)* to calculate the function value.

Listing code

```csharp
using System;
using System.Runtime.InteropServices;
using UtilityMatlabCompilerVer4;

namespace MatlabCSharpExample
{
    class Example
    {
        // constructor
        public Example()
        {
            curvefittinglibInitialize();
        }

        public void CleanUp()
        {
            curvefittinglibTerminate();
        }

        /* declare dll functions */
        [DllImport( "curvefittinglib.dll ", CallingConvention = CallingConvention.Cdecl)]
        public static extern void curvefittinglibInitialize();

        [DllImport( "curvefittinglib.dll ", CallingConvention = CallingConvention.Cdecl)]
    }
```
public static extern void curvefittinglibTerminate();

[DllImport( "curvefittinglib.dll ", CallingConvention = CallingConvention.Cdecl)]
public static extern void mlfMyinterp1(int nargout, ref IntPtr yi, IntPtr x , IntPtr y, IntPtr xi);

[DllImport( "curvefittinglib.dll ", CallingConvention = CallingConvention.Cdecl)]
public static extern void mlfMyinterp2(int nargout, ref IntPtr ZI, IntPtr X , IntPtr Y, IntPtr Z, IntPtr XI , IntPtr YI, IntPtr method);

[DllImport( "curvefittinglib.dll ", CallingConvention = CallingConvention.Cdecl)]
public static extern void mlfMypolyfit(int nargout, ref IntPtr p, IntPtr x , IntPtr y, IntPtr n);

[DllImport( "curvefittinglib.dll ", CallingConvention = CallingConvention.Cdecl)]
public static extern void mlfMypolyval(int nargout, ref IntPtr y, IntPtr p, IntPtr x);

[DllImport( "curvefittinglib.dll ", CallingConvention = CallingConvention.Cdecl)]
public static extern void mlfMymeshgrid(int nargout, ref IntPtr X, ref IntPtr Y , IntPtr vectorstepx, IntPtr vectorstepy);

[DllImport( "curvefittinglib.dll ", CallingConvention = CallingConvention.Cdecl)]
public static extern void mlfMygriddata(int nargout, ref IntPtr ZI, IntPtr x, IntPtr y , IntPtr z, IntPtr XI, IntPtr YI);

[DllImport( "curvefittinglib.dll ", CallingConvention = CallingConvention.Cdecl)]
public static extern void mlfMyfinemeshgrid(int nargout, ref IntPtr row, ref IntPtr col , IntPtr vectorstepx, IntPtr vectorstepy);

/* end dll functions */

[STAThread]
static void Main(string[] args)
{
    Console.WriteLine("Curve Fitting:" );

    Example obj = new Example();
}
Console.WriteLine("1. Polynomial");
obj.PolynomialFittingCurve();

obj.CleanUp();
}

/* ************************** */

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_three = 3;
double db_oneValue = 2.2;

/* declare mxArray variables */
IntPtr mx_X = (IntPtr) null;
IntPtr mx_Y = (IntPtr) null;
IntPtr mx_three = (IntPtr) null;
IntPtr mx_oneValue = (IntPtr) null;
IntPtr mx_funcValue = (IntPtr) null;

IntPtr mx_three = (IntPtr) null;
IntPtr mx_oneValue = (IntPtr) null;
IntPtr mx_funcValue = (IntPtr) null;

/* convert Cs double to mxArray */
mx_X = MatlabCSharp.double2mxArray_vectorRowReal (db_X);
mx_Y = MatlabCSharp.double2mxArray_vectorRowReal (db_Y);

mx_three = MatlabCSharp.double2mxArray_scalarReal (db_three);
mx_oneValue = MatlabCSharp.double2mxArray_scalarReal (db_oneValue);

/* call an implemental function */
mlfMypolyfit(1, ref mx_coefs, mx_X, mx_Y, mx_three);

/* convert back to Cs double */
double[] db_coefs = MatlabCSharp.mxArray2double_vectorReal(mx_coefs);

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

Function value at oneValue */
mlfMypolyval(1, ref mx_funcValue, mx_coefs, mx_oneValue);

double db_funcValue = MatlabCSharp.mxArray2double_scalarReal(mx_funcValue) ;
Console.Write("The function value at 2.2 is: {0} ", db_funcValue.ToString() ) ;

} // end class

} // end code

8.2 One-Dimensional Polynomial Interpolation

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

**Problem 2** There are two arrays $X$ and $Y$ have a relationship via a function, $y = f(x)$, $x \in X$ and $y \in Y$.

**input**

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 mlfMyinterp1(..) with the default method (liner method) to solve the problem. To learn more about other possible methods see the MATLAB function inter1(..) in [4].

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 mxArray variables */
    IntPtr mx_X = (IntPtr) null;
    IntPtr mx_Y = (IntPtr) null;
    IntPtr mx_oneValue = (IntPtr) null;
    IntPtr mx_funcValue = (IntPtr) null;

    /* convert Cs double to mxArray */
    mx_X = MatlabCSharp.double2mxArray_vectorRowReal(db_X);
    mx_Y = MatlabCSharp.double2mxArray_vectorRowReal(db_Y);
    mx_oneValue = MatlabCSharp.double2mxArray_scalarReal(db_oneValue);

    /* call an implemental function */
    mlfMyinterp1(1, ref mx_funcValue, mx_X, mx_Y, mx_oneValue);

    /* convert back to Cs double */
    double db_funcValue = MatlabCSharp.mxArray2double_scalarReal(mx_funcValue);
    Console.WriteLine("The function value at 2.1 is: {0}", db_funcValue.ToString());
}

8.3 Two-Dimensional Polynomial Interpolation for Grid Points

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

**Problem 3** There are three matrixes, x, y, and z, that have a relationship via a function, $z = f(x, y)$, $x \in x$, $y \in y$, and $z \in z$. 
input  Grid points (x,y).
Representation of these grid points are 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 in the next page)

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 you have grid points as in Fig. 8.1.

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

\[
\begin{array}{cccccc}
-3 & -2 & -1 & 0 & 1 & 2 & 3 \\
-3 & -2 & -1 & 0 & 1 & 2 & 3 \\
-3 & -2 & -1 & 0 & 1 & 2 & 3 \\
-3 & -2 & -1 & 0 & 1 & 2 & 3 \\
-3 & -2 & -1 & 0 & 1 & 2 & 3 \\
-3 & -2 & -1 & 0 & 1 & 2 & 3
\end{array}
\text{ from point 1 to 7}
\]

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

\[
\begin{array}{cccccc}
-3 & -3 & -3 & -3 & -3 & -3 \\
-2 & -2 & -2 & -2 & -2 & -2 \\
-1 & -1 & -1 & -1 & -1 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 \\
2 & 2 & 2 & 2 & 2 & 2
\end{array}
\text{ from point 1 to 7}
\]

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

\[
\begin{array}{cccccc}
0.0001 & 0.0034 & -0.0299 & -0.2450 & -0.1100 & -0.0043 & -0.0000
\end{array}
\text{ (point 1 - 7)}
\]
B. Programming code

The following is the code to solve Problem 3. In the code, you will use the generated function mlfMyinterp2(..) with the cubic method to solve the problem. To learn more about other possible methods see the MATLAB function inter2(..) in [4]. The procedure of this code is:

1. Assign values for matrix x and matrix y

2. Assign values for matrix z

3. Call the two-dimensional interpolation function to evaluate the value at the point \((a, b)\)
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 mxArray variables */
    IntPtr mx_a = (IntPtr) null;
    IntPtr mx_b = (IntPtr) null;
    IntPtr mx_interp2z = (IntPtr) null;
    IntPtr mx_x = (IntPtr) null;
    IntPtr mx_y = (IntPtr) null;
    IntPtr mx_z = (IntPtr) null;
    IntPtr mx_vectorstepx = (IntPtr) null;
    IntPtr mx_vectorstepy = (IntPtr) null;
    IntPtr mx_method = null;
    
    /* convert Cs double to mxArray */
    mx_a = MatlabCSharp.double2mxArray_scalarReal(db_a);
    mx_b = MatlabCSharp.double2mxArray_scalarReal(db_b);
    
    /* same for two step-vectors */
    mx_vectorstepx = MatlabCSharp.double2mxArray_vectorRowReal(db_vectorstep);
    mx_vectorstepy = MatlabCSharp.double2mxArray_vectorRowReal(db_vectorstep);
    
    mx_z = MatlabCSharp.double2mxArray_matrixReal(z);
/* call implemental functions */
/* create values for the matrix x and matrix y */
mlfMymeshgrid(2, ref mx_x, ref mx_y, mx_vectorstepx, mx_vectorstepy); 

String interp2method = "cubic";
mx_method = MatlabCSharp.mxCreateString(interp2method);

mlfMyinterp2(1, ref mx_interp2z, mx_x, mx_y, mx_z, mx_a, mx_b, mx_method);

/* convert back to Cs double */
double [,] db_x = MatlabCSharp.mxArray2double_matrixReal(mx_x);
double [,] db_y = MatlabCSharp.mxArray2double_matrixReal(mx_y);

/* print out */
Console.WriteLine("Matrix x : ");
MatlabCSharp.printMatrix(db_x);

Console.WriteLine("Matrix y : ");
MatlabCSharp.printMatrix(db_y);

double db_interp2z = MatlabCSharp.mxArray2double_scalarReal(mx_interp2z);
Console.WriteLine("Interpolation in two dimensions with cubic method ");
Console.WriteLine(" z = {0}", db_interp2z.ToString());

} 

------------------------------ 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. 8.1, 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. 8.2), therefore to avoid mistakes and to have the convenience of visibility, you can set up your matrix data as in Fig. 8.1 (y axis direction from top to bottom).
Figure 8.2: 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 [4]. The following code finds the fine solution of Problem 3 by using the function mygriddata(..).

Listing code

```java
public void TwoDimensionsInterpolationFineSolution()
{
    /* 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_finevectorstep = {-3, 0.2, 3};

double db_a = 2.3;
double db_b = 0.7;

/* declare mxArray variables */
IntPtr mx_a       = (IntPtr) null ;
IntPtr mx_b       = (IntPtr) null ;
IntPtr mx_interp2z = (IntPtr) null ;

IntPtr mx_x       = (IntPtr) null ;
IntPtr mx_y       = (IntPtr) null ;
IntPtr mx_z       = (IntPtr) null ;

IntPtr mx_vectorstepx = (IntPtr) null ;
IntPtr mx_vectorstepy = (IntPtr) null ;

IntPtr mx_method ;
//
IntPtr mx_fine = (IntPtr) null ;
IntPtr mx_finey = (IntPtr) null ;
IntPtr mx_finez = (IntPtr) null ;

IntPtr mx_finerow = (IntPtr) null ;
IntPtr mx_finecol = (IntPtr) null ;

IntPtr mx_finevectorstepx = (IntPtr) null ;
IntPtr mx_finevectorstepy = (IntPtr) null ;

/* convert Cs double to mxArray */
mx_a = MatlabCSharp.double2mxArray_scalarReal (db_a) ;
mx_b = MatlabCSharp.double2mxArray_scalarReal (db_b) ;

/* same for two step-vectors */
mx_vectorstepx = MatlabCSharp.double2mxArray_vectorRowReal (db_vectorstep) ;
mx_vectorstepy = MatlabCSharp.double2mxArray_vectorRowReal (db_vectorstep) ;

mx_z = MatlabCSharp.double2mxArray_matrixReal (z) ;
//
mx_finevectorstepx = MatlabCSharp.double2mxArray_vectorRowReal(db_finevectorstep);
mx_finevectorstepy = MatlabCSharp.double2mxArray_vectorRowReal(db_finevectorstep);

/* get size for fine matrixes */
mlfMymeshgrid(2, ref mx_finerow, ref mx_finecol, mx_finevectorstepx, mx_finevectorstepy);

int finerow = (int) MatlabCSharp.mxArray2double_scalarReal(mx_finerow);
int finecol = (int) MatlabCSharp.mxArray2double_scalarReal(mx_finecol);

Console.WriteLine("fine row = {0}", finerow.ToString());
Console.WriteLine("fine col = {0}", finecol.ToString());

/* call implemental functions */
/* create values for the matrix x and matrix y */
mlfMymeshgrid(2, ref mx_x, ref mx_y, mx_vectorstepx, mx_vectorstepy);

/* create values for the fine matrix x and fine matrix y */
mlfMymeshgrid(2, ref mx_finex, ref mx_finey, mx_finevectorstepx, mx_finevectorstepy);

/* get a fine matrix mx_finez from mx_z */
mlfMygriddata(1, ref mx_finez, mx_x, mx_y, mx_z, mx_finex, mx_finey);

String interp2method = "cubic";
mx_method = MatlabCSharp.mxCreateString(interp2method);

mlfMyinterp2(1, ref mx_interp2z, mx_finex, mx_finey, mx_finez, mx_a, mx_b, mx_method);

/* convert back to Cs double */
double db_interp2finez = MatlabCSharp.mxArray2double_scalarReal(mx_interp2z);
Console.WriteLine("Interpolation in two dimensions with cubic method and a fine grid");
Console.WriteLine(" z = {0}", db_interp2finez.ToString());

}
Chapter 9

Roots of Equations

In this chapter we’ll generate a C shared library `rootslib` 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 project to find the roots of functions.

Following are steps to create a C shared library rootslib.dll which will be used in the next sections. We will write the M-files as shown below. These functions will be used to generate the C shared library.

`myfzero.m` and `myroots.m`

```
function r = myroots(c)
    r = roots(c) ;

function x = myfzero(strfunc, x0)
    F = inline(strfunc) ;
    x = fzero(F, x0) ;
```

The steps to create a C share library are:

1. Write the command in Windows Command Prompt as follows to create a C shared library
   
   ```
   rootslib:
mcc -B csharedlib:rootslib myfzero.m myroots.m
   ```

2. MATLAB Compiler 4 will create eight files of this C shared library:
   
   ```
   rootslib.c   rootslib.ctf   rootslib.dll
   rootslib.exp rootslib.exports rootslib.h
   rootslib.lib rootslib_mcc_component_data.c
   ```

   Add two files rootslib.dll and rootslib.ctf into the project directory `bin\Debug`.

3. In the following sections, we’ll use the following implemental functions in this library to solve the problems (open the file rootslib.h to see the names of these functions):
   
   ```
   void mlfMyroots(int nargout, mxArray** r, mxArray* c);
   void mlfMyfzero(int nargout, mxArray** x, mxArray* strfunc, mxArray* x0);
   ```

9.1 Roots of Polynomials

This section describes how to use the functions in the generated library to find the roots of a polynomial function.

**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):

\[
 f(x) = c_1 x^3 + c_2 x^2 + c_3 x + c_4
\]

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 `mlfMyroots(..)`. See the MATLAB manual [2] for more information on the MATLAB function `roots(..)`. 
using System;
using System.Runtime.InteropServices;
using UtilityMatlabCompilerVer4;

namespace MatlabCSharpExample
{
    class Example
    {
        //constructor
        public Example()
        {
            rootslibInitialize();
        }

        public void CleanUp()
        {
            rootslibTerminate();
        }

        /* declare dll functions */
        [DllImport("rootslib.dll", CallingConvention = CallingConvention.Cdecl)]
        public static extern void rootslibInitialize();

        [DllImport("rootslib.dll", CallingConvention = CallingConvention.Cdecl)]
        public static extern void rootslibTerminate();

        [DllImport("rootslib.dll", CallingConvention = CallingConvention.Cdecl)]
        public static extern void mlfMyfzero(int nargout, ref IntPtr x, IntPtr strfunc, IntPtr x0);

        [DllImport("rootslib.dll", CallingConvention = CallingConvention.Cdecl)]
        public static extern void mlfMyroots(int nargout, ref IntPtr r, IntPtr c);

        /* end dll functions */

        [STAThread]
        static void Main(string[] args)
        {
        }
Console.WriteLine("Roots of functions.")

Example obj = new Example();
obj.FindingRootsPolynomial();
obj.CleanUp();
}

/* ******************************* */

public void FindingRootsPolynomial()
{
    /*
    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 */
    IntPtr mx_coefs = (IntPtr) null;
    IntPtr mx_x = (IntPtr) null;

    /* convert Cs double to mxArray */
    mx_coefs = MatlabCSharp.double2mxArray_vectorColumnReal (db_coefs);

    /* step 4 : call an implemental function */
    mlfMyroots(1, ref mx_x, mx_coefs);

    /* step 5 : convert back to C/C++ double */
    double[] db_xReal = new double[order];
    double[] db_xImag = new double[order];

    MatlabCSharp.mxArray2double_vectorComplex(mx_x, ref db_xReal, ref db_xImag);

    /* print out */
    Console.WriteLine("Solutions of the polynomial function : ");
    int i;
    for (i=0; i<order; i++)
    {
Note
The roots of a polynomial function may have imaginary terms. Therefore we need to use the function `mxArray2double_vectorComplex(..)` to convert to `double` type.

### 9.2 The Root of a Nonlinear-Equation

This section describes how to use the function `mlfMyfzero(..)` in the generated `rootslib` library 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 [4].

**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 `mlfMyfzero(..).

Listing code

```csharp
public void FindingZeroFunction ( )
{
    /* Find the solution of the function f(x) = \sin(2x) + \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 */
IntPtr mx_initialGuess = (IntPtr) null ;
IntPtr mx_x = (IntPtr) null ;
IntPtr mx_strfunc ;

/* convert Cs double to mxArray */
mx_initialGuess = MatlabCSharp.double2mxArray_scalarReal (db_initialGuess) ;
mx_strfunc = MatlabCSharp.mxCreateString(strfunc) ;

/* call an implemental function */
mlfMyfzero(1, ref mx_x, mx_strfunc, mx_initialGuess);

/* convert back to Cs double */
double db_xReal ;
db_xReal = MatlabCSharp.mxArray2double_scalarReal(mx_x) ;

/* print out */
Console.WriteLine("Solutions of the given function : ");
Console.WriteLine( db_xReal.ToString() ) ;

}
Chapter 10

Fast Fourier Transform

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:

$$F(s) = \int_{-\infty}^{+\infty} f(x)e^{-i(2\pi s)x} dx$$  \hspace{1cm} (10.1a)

and its inverse

$$f(x) = \int_{-\infty}^{+\infty} F(s)e^{i(2\pi x)s} ds$$  \hspace{1cm} (10.1b)

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 C shared library in the following sections.

In this chapter we’ll generate a C shared library fftlib from common M-files working on FFT problems. The generated functions of these libraries will be used in a MSVC# .Net project to solve the integral problems.

Following are steps to create a C shared library fftlib.dll which will be used to solve FFT problems in the next sections. We will write the M-files as shown below. These functions will be used to generate the C shared library.

`myfft.m, myifft.m, myfft2.m, and myifft2.m`
function Y = myfft(X)
Y = fft(X);

function Y = myifft(X)
Y = ifft(X);

function Y = myfft2(X)
Y = fft2(X);

function Y = myifft2(X)
Y = ifft2(X);

The steps to create a C share library are:

1. Write the command in Windows Command Prompt as follows to create a C shared library `fftlib`:
   
mcc -B csharedlib:fftlib myfft.m myifft.m myfft2.m myifft2.m

2. MATLAB Compiler 4.0 will create eight files of this C shared library:
   
   - fftlib.c
   - fftlib.ctf
   - fftlib.dll
   - fftlib.exp
   - fftlib.exports
   - fftlib.h
   - fftlib.lib
   - fftlib_mcc_component_data.c

   Add two files fftlib.dll and fftlib.ctf into the project directory `bin\Debug`.

3. In the following sections, we’ll use the following implemental functions in this library to solve the common problems in the fast Fourier transform (open the file fftlib.h to see the names of these functions):

   ```
   void mlfMyfft (int nargout, mxArray** Y, mxArray* X);
   void mlfMyifft (int nargout, mxArray** Y, mxArray* X);
   void mlfMyfft2 (int nargout, mxArray** Y, mxArray* X);
   void mlfMyifft2 (int nargout, mxArray** Y, mxArray* X);
   ```
10.1 One-Dimensional Fast Fourier Transform

The MATLAB functions implement FFT and its inverse for a vector $X$ with length $N$ as follows:

$$Y_k = \sum_{n=1}^{N} X_n e^{-i2\pi \left( \frac{n-1}{N} \right)(k-1)} \quad 1 \leq k \leq N$$  \hspace{1cm} (10.2a)

and its inverse

$$X_n = \frac{1}{N} \sum_{k=1}^{N} Y_k e^{i2\pi \left( \frac{k-1}{N} \right)(n-1)} \quad 1 \leq n \leq N$$  \hspace{1cm} (10.2b)

Remarks

1. The vectors $X$ and $Y$ in equation 10.2 are represented by functions $f(x)$ and $F(s)$ in equation 10.1, respectively.

2. The first index of a vector in MATLAB starts at 1, therefore in the equation 10.2 we have term $(n-1)$ and $(k-1)$. This produces identical results as the traditional Fourier equations from 0 to $(N-1)$.

Problem 1

**input**  Vector $X$,

$X = \{ 6, 3, 7, -9, 0, 3, -2, 1 \}$

**output**  Finding the FFT vector $Y$ in Eq. 10.2a

The following is the code to solve Problem 1 by using the function mlfMyfft().

Listing code

```csharp
using System;
using System.Runtime.InteropServices;
using UtilityMatlabCompilerVer4;

namespace MatlabCSharpExample
{
    class Example
    {
        // constructor
        public Example()
        {
```
fftlibInitialize();
}

public void CleanUp()
{
    fftlibTerminate();
}

/* declare dll functions */
[DllImport( "fftlib.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern void fftlibInitialize();

[DllImport( "fftlib.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern void fftlibTerminate();

[DllImport( "fftlib.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern void mlfMyfft(int nargout, ref IntPtr Y, IntPtr X);

[DllImport( "fftlib.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern void mlfMyifft(int nargout, ref IntPtr Y, IntPtr X);

[DllImport( "fftlib.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern void mlfMyfft2(int nargout, ref IntPtr Y, IntPtr X);

[DllImport( "fftlib.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern void mlfMyifft2(int nargout, ref IntPtr Y, IntPtr X);

/* end dll functions */

[STAThread]
static void Main(string[] args)
{
    Console.WriteLine("Fast Fourier Transform.");

    Example obj = new Example();
    obj.FastFourierTrans1D();
    obj.CleanUp();
}
public void FastFourierTrans1D()
{

double[] db_X = { 6, 3, 7, -9, 0, 3, -2, 1 };
int vectorSize = 8;

/* declare mxArray variables */
IntPtr mx_X = (IntPtr) null;
IntPtr mx_Y = (IntPtr) null;

/* convert Cs double to mxArray */
mx_X = MatlabCSharp.double2mxArray_vectorColumnReal (db_X);

/* call an implemental function */
mlfMyfft(1, ref mx_Y, mx_X);

/* convert back to Cs double */
double[] db_YReal = new double [vectorSize];
double[] db_YImag = new double [vectorSize];

MatlabCSharp.mxArray2double_vectorComplex(mx_Y, ref db_YReal, ref db_YImag);

/* print out */
Console.WriteLine("Fast Fourier Transform of X :");
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");

}

} // end class

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. 10.2b

The following is the code to solve Problem 1B by using the function mlfMyift(..).

Listing code

```csharp
public void InverseFastFourierTrans1D()
{

double[] db_YReal = { 9.00, 13.0711, 1.0, -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 */
IntPtr mx_X = (IntPtr) null;
IntPtr mx_Y = (IntPtr) null;

/* convert Cs double to mxArray */
mx_Y = MatlabCSharp.double2mxArray_vectorColumnComplex(db_YReal, db_YImag);

/* call an implemental function */
mlfMyift(1, ref mx_X, mx_Y);

/* convert back to Cs double */
double[] db_XReal = new double [vectorSize];
double[] db_XImag = new double [vectorSize];
MatlabCSharp.mxArray2double_vectorComplex(mx_X, ref db_XReal, ref db_XImag);

/* 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") ;

10.2 Two-Dimensional Fast Fourier Transform

The MATLAB function fft2(..) (\( Y = fft2(X) \)) 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 \( Y = fft2(X, m, n) \), refer to the MATLAB manual [4].

Problem 2

input       Matrix \( X \),

\[
X = \begin{bmatrix}
4 & 3.2 & 6.8 & 9.1 \\
-4 & 1.2 & 4.3 & 5.4 \\
2.2 & -6.7 & 8 & 12
\end{bmatrix}
\]

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 mlfMyfft2(..) in the generated fftlib.

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 */
IntPtr mx_X = (IntPtr) null;
IntPtr mx_Y = (IntPtr) null;

/* convert Cs double to mxArray */
mx_X = MatlabCSharp.double2mxArray_matrixReal(X);

/* call an implemental function */
mlfMyfft2(1, ref mx_Y, mx_X);

/* convert back to Cs double */
double[,] db_YReal = new double[row, col];
double[,] db_YImag = new double[row, col];
MatlabCSharp.mxArray2double_matrixComplex(mx_Y, ref db_YReal, ref db_YImag);

/* 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" );
    }
    Console.WriteLine("\n");
}

Console.WriteLine("\n");

}
Problem 2B

input  Matrix $Y$,

\[
Y = \begin{bmatrix}
(45.70 + 0i) & (-16.9000 + 29.0000i) & (-3.10 + 0i) & (-16.9000 - 29.0000i) \\
(11.80 + 7.621i) & (-8.4806 - 3.4849i) & (-0.70 + 9.5263i) & (16.9806 + 7.8151i) \\
(11.80 - 7.621i) & (16.9806 - 7.8151i) & (-0.70 - 9.5263i) & (-8.4806 + 3.4849i)
\end{bmatrix}
\]

output  Finding the matrix $X$ which is an inverse FFT matrix of $Y$.

The following is the code to solve Problem 2B by using the function mlfMyiifft2(..).

Listing code

```csharp
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 */
IntPtr mx_X = (IntPtr) null ;
IntPtr mx_Y = (IntPtr) null ;

/* convert Cs double to mxArray */
mx_X = MatlabCSharp.double2mxArray_matrixReal(X) ;

/* call an implemental function */
mlfMyfft2(1, ref mx_Y, mx_X); 

/* convert back to Cs double */
double [,] db_YReal = new double [row, col] ;
double [,] db_YImag = new double [row, col] ;

MatlabCSharp.mxArray2double_matrixComplex(mx_Y, ref db_YReal, ref db_YImag) ;
```
/* 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");

}
Chapter 11

Eigenvalues and Eigenvectors

In this chapter we’ll generate a C shared library `eigenlib` from common M-files working on problems of eigenvectors and eigenvalues. The generated functions of this library will be used in a MSVC# .Net project to solve the eigen problems. 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 [3].

Following are steps to create a C shared library `eigenlib.dll` which will be used to solve problems in the next sections. We will write the M-file `myeig.m` as shown below. This function will be used to generate the C shared library.

```matlab
function [V,D] = myeig(A)

[V,D] = eig(A) ;
```

The steps to create a C share library are:

1. Write the command in Windows Command Prompt as follows to create a C shared library `eigenlib`:
   ```
   mcc -B csharedlib:eigenlib myeig.m
   ```

2. MATLAB Compiler 4 will create eight files of this C shared library:
   ```
   eigenlib.c   eigenlib.ctf   eigenlib.dll
   eigenlib.exp eigenlib.exports eigenlib.h
   eigenlib.lib eigenlib_mcc_component_data.c
   ```

   Add two files `eigenlib.dll` and `eigenlib.ctf` into the project directory bin\Debug.

3. In the next sections, we’ll use the following implemental function in this library to solve the common eigen-problems (open the file `eigenlib.h` to see the name of this function):
11.1 Eigenvalues and Eigenvectors

Problem 1

input  a square matrix $A$

\[
A = \begin{bmatrix}
0 & -6 & -1 \\
6 & 2 & -16 \\
-5 & 20 & -10
\end{bmatrix}
\]

output  Finding eigenvalues and eigenvectors of $A$

The following is the code to solve Problem 1 by using function `mlfMyeig(..)` in the generated `eigenlib` library to find eigenvalues and eigenvectors of a square matrix.

Listing code

```csharp
using System;
using System.Runtime.InteropServices;
using UtilityMatlabCompilerVer4 ;

namespace MatlabCSharpExample
{
    class Example
    {
        public Example()
        {
            eigenlibInitialize();
        }

        public void CleanUp()
        {
            eigenlibTerminate();
        }

        /* declare dll functions */

        [DllImport( "eigenlib.dll ", CallingConvention = CallingConvention.Cdecl)]
```
public static extern void eigenlibInitialize();

[DllImport( "eigenlib.dll ", CallingConvention = CallingConvention.Cdecl)]
public static extern void eigenlibTerminate();

[DllImport( "eigenlib.dll ", CallingConvention = CallingConvention.Cdecl)]
public static extern void mlfMyeig(int nargout, ref IntPtr V, ref IntPtr D, IntPtr A);

امية end dll functions */

[STAThread]
static void Main(string[] args)
{
    Console.WriteLine(" ");

    Example obj = new Example();

    Console.WriteLine("Eigenvalues and eigenvectors ");
    obj.EigValueVector();

    obj.CleanUp();
}

/* **************************** */
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 //
IntPtr mx_A = (IntPtr) null;
IntPtr mx_eigenvectors = (IntPtr) null;
IntPtr mx_eigenvalues = (IntPtr) null;
/ convert Cs double to mxArray */
mx_A = MatlabCSharp.double2mxArray_matrixReal(A);

/* call an implemental function */
mlfMyeig(2, ref mx_eigenvectors, ref mx_eigenvalues, mx_A);

/* 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];
MatlabCSharp.mxArray2double_matrixComplex(
    mx_eigenvectors, ref db_eigenvectorsReal, ref db_eigenvectorsImag );
MatlabCSharp.mxArray2double_matrixComplex( 
    mx_eigenvalues , ref db_eigenvaluesReal , ref db_eigenvaluesImag );

/* 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++)
    {
    
    
}
The MATLAB function 
\[ [V,D] = \text{eig}(A) \] 
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:

\[
eigenvalues = \begin{bmatrix}
-0.30710 & 0.0000 & 0.0000 \\
0.0000 & -0.24645 + 1.76008i & 0.0000 \\
0.0000 & 0.0000 & -0.24645 - 1.76008i
\end{bmatrix}
\]

then the eigenvalues of the matrix \( A \) are:

\[
eigenvalue_1 = -0.30710 \\
eigenvalue_2 = -0.24645 + 1.76008i \\
eigenvalue_3 = -0.24645 - 1.76008i
\]

The MATLAB function \( [V,D] = \text{eig}(A) \) 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:

\[
eigenvector_1 = \begin{bmatrix} -0.83261 \\ -0.35534 \\ -0.42485 \end{bmatrix}, \quad \eigenvector_2 = \begin{bmatrix} 0.20027 - 0.13936i \\ -0.21104 - 0.64472i \\ -0.69301 \end{bmatrix}
\]

, and \( \eigenvector_3 = \begin{bmatrix} 0.20027 + 0.13936i \\ -0.21104 + 0.64472i \\ -0.69301 \end{bmatrix} \)
Part II:
Calling MATLAB Workspace in C# Functions

Using MATLAB functions in C# Windows Form Applications
Chapter 12

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 transfer inputs from a C# function to the MATLAB workspace and then transfer 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.

12.1 Calling MATLAB Workspace with Input/Output as a Scalar

Problem 1

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
. Get the result \( c \) in the MATLAB workspace and transfer back to the C# function

The following is the code to solve Problem 1 by using MATLAB Engine in MATLAB Compiler 4.

Listing code

```csharp
using System;
using System.Runtime.InteropServices;
using UtilityMatlabCompilerVer4;
using System.IO;
```
namespace MatlabCSharpExample
{
    class Example
    {
        IntPtr ep;
        public Example()
        {
            ep = MatlabCSharp.engOpen(null);
        }

        public void CleanUp()
        {
            MatlabCSharp.engClose(ep);
        }
    }

    [STAThread]
    static void Main(string[] args)
    {
        Console.WriteLine("Using MATLAB Engine to call the MATLAB workspace ");
        Console.WriteLine("to perform the particular tasks ");

        Example obj = new Example();
        obj.SimplePlus();

        obj.CleanUp();
    }

    /* ****************************************** */
    public void SimplePlus()
    {
        double db_a = 1.2;
        double db_b = 2.5;

        // declare mxArray variables to use
        IntPtr mx_a = (IntPtr) null;
        IntPtr mx_b = (IntPtr) null;
        IntPtr mx_c = (IntPtr) null;

        // convert Cs double to mxArray
        mx_a = MatlabCSharp.double2mxArray_scalarReal(db_a);
The result from the simple plus: 123.2
Call the MATLAB workspace to perform the tasks as following in a C# function:

a) Finding the solution $x$ of linear system equations $Ax = b$
b) Calculating the upper matrix $U$ with the LU decompression method
c) Getting results in the MATLAB workspace and converting to C# double

The following is the code to solve Problem 2 by using MATLAB Engine in MATLAB Compiler 4.

Listing code

```csharp
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};

    /* declare mxArray variables */
    IntPtr mx_A = (IntPtr) null;  
    IntPtr mx_b = (IntPtr) null;  
    IntPtr mx_vectorX = (IntPtr) null;  
    IntPtr mx_U = (IntPtr) null;  

    /* convert Cs double to mxArray */
    mx_b = MatlabCSharp.double2mxArray_vectorColumnReal(b);
    mx_A = MatlabCSharp.double2mxArray_matrixReal(A);

    /* begin performing tasks in the Matlab workspace
    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. */

    MatlabCSharp.engPutVariable(ep, "ml_A", mx_A);
    MatlabCSharp.engPutVariable(ep, "ml_b", mx_b);

    MatlabCSharp.engEvalString(ep, "ml_vectorX = mldivide(ml_A, ml_b) ; " );
    MatlabCSharp.engEvalString(ep, "[ml_L, ml_U, ml_P] = lu(ml_A) ; " );

    /* end tasks in the Matlab workspace */

    /* get results in the Matlab workspace then convert to mxArray */
```
12.3 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. The steps are:

1. Transfer values of C# double to mxArray.
2. Change mxArray variable names to MATLAB Workspace names.
3. Perform tasks in the MATLAB Workspace.

Following is the code to create the graphic. The created figure is shown in Fig. 13.1.

Listing code

```csharp
public void MatlabEnginePlot()
{
    double[] X = { 12.5, 32.2, 45.6 } ;
    double[] Y = { 12.5, 32.2, 45.6 } ;

    /* declare mxArray variables */
    IntPtr mx_X = (IntPtr) null ;
    IntPtr mx_Y = (IntPtr) null ;

    /* convert mxArray to Cs double */
    double[] db_vectorX = MatlabCSharp.mxArray2double_vectorReal(mx_vectorX) ;
    double[,] db_matrixU = MatlabCSharp.mxArray2double_matrixReal(mx_U) ;

    /* close Matlab workspace */
    MatlabCSharp.engEvalString(ep, "close") ;

    /* print out */
    Console.WriteLine("Solution of the equation Ax = b is: ") ;
    MatlabCSharp.printVector(db_vectorX) ;

    Console.WriteLine("The upper matrix U:") ;
    MatlabCSharp.printMatrix(db_matrixU) ;
}
```
// Convert Cs double to mxArray
mx_X = MatlabCSharp.double2mxArray_vectorColumnReal(X) ;
mx_Y = MatlabCSharp.double2mxArray_vectorColumnReal(Y) ;

/* transfer name mx_X to the name, ml_X */
MatlabCSharp.engPutVariable(ep, "ml_X" , mx_X ) ;
MatlabCSharp.engPutVariable(ep, "ml_Y" , mx_Y ) ;

/* perform tasks in the Matlab workspace */
MatlabCSharp.engEvalString(ep, " plot(ml_X, ml_Y ,'r'); " ) ;

/* keep the figure */
Console.WriteLine("Hit Enter key to close the figure and continue") ;
System.Console.Read() ;

/* close the Matlab workspace */
MatlabCSharp.engEvalString(ep, "close") ;

Figure 12.1: The figure is created by using the MATLAB workspace

**Remark**

- With directly creating the graphic by MATLAB Engine, 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() ;`
Chapter 13

Using MATLAB Functions In C# Windows Forms Applications

This chapter describes how to use MATLAB 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 of C# double type.
3. Use MATLAB built-in functions in the dll file to perform the task.
4. Transfer back result values to C# double type.
5. Put the result to the Windows Form application.

The Windows Forms application in this chapter is an application of the matrix multiplication. The user will enter the row and column number of matrixes A and B (see Fig. 13.1), then click Set Number button. The application will reset the grid for entering values of A and B. The user then enters data for a matrix A and a matrix B, then click Calculation button. The application will use the MATLAB built-in function mlfMymtimes(..) in a C shared library to calculate the multiplication matrix C, then put back result to the Windows application. The function mlfMymtimes(..) is created as described in Chapter 4.

The procedure to create this Windows Form application is:

1. Create a function mlfMymtimes(..) as described in Chapter 4.
2. Open a new C# project of Windows Application.
3. Copy the files matrixcomputationslib.ctf and matrixcomputationslib.dll to the folder bin\Debug.
4. Copy the file CsharpMatlabCompilerVer4.cs in Chapter 3 to the C# project.
5. Use the file `mlMymtimes(..)` as an dll import file.

The complete code of this example is in this CD-Rom. Here we just put a part of code that implements the Windows Form application.

![MATLAB in a C# Windows Forms Application](image)

Figure 13.1: MATLAB in a C# Windows Forms Application

I. Get input data from the Windows Form application and assign to variables of C# `double` type

Listing code

```csharp
public void GetMatrixA()
{
    int rowA, colA;

    try
    {
```
rowA = Convert.ToInt32(tbMatrixA_row.Text);
colA = Convert.ToInt32(tbMatrixA_col.Text);
}
catch
{
    MessageBox.Show("Please enter the number") ;
    return ;
}

int i, j ;

try
{
    for( i=0; i<m_numRowMatricATextBox; i++)
    {
        for( j=0; j<m_numColMatricATextBox; j++)
        {

        }
    }
}
catch
{
    MessageBox.Show("Please enter the number in Matrix A") ;
    return ;
}

public void GetMatrixB()
{
    int rowB, colB ;
    try
rowB = Convert.ToInt32(tbMatrixB_row.Text);
colB = Convert.ToInt32(tbMatrixB_col.Text);

try
{
    for( i=0; i<m_numRowMatrixBTextBox; i++)
    {
        for( j=0; j<m_numColMatrixBTextBox; j++)
        {
            db_MatrixB[i,j] = Convert.ToDouble(arrTextBoxMatrixB[i,j].Text);
        }
    }
}

catch
{
    MessageBox.Show("Please enter the number") ;
    return ;
}

int i, j ;

try
{
    for( i=0; i<m_numRowMatrixBTextBox; i++)
    {
        for( j=0; j<m_numColMatrixBTextBox; j++)
        {
            db_MatrixB[i,j] = Convert.ToDouble(arrTextBoxMatrixB[i,j].Text);
        }
    }
}

catch
{
    MessageBox.Show("Please enter the number in Matrix B") ;
    return ;
}

end code
II. Use MATLAB built-in functions to calculate the multiplication matrix

Listing code

```csharp
public void GetMatrixC()
{
    // calculate matrix multiplication

    /* declare mxArray variables */
    IntPtr mx_A = (IntPtr)null;
    IntPtr mx_B = (IntPtr)null;
    IntPtr mx_C = (IntPtr)null;

    /* convert C# matrix to mxArray */
    mx_A = MatlabCSharp.double2mxArray_matrixReal(db_MatrixA);
    mx_B = MatlabCSharp.double2mxArray_matrixReal(db_MatrixB);

    /* call an implemental function */
    mlfMymtimes(1, ref mx_C, mx_A, mx_B);

    /* convert back to C# double */
    db_MatrixC = MatlabCSharp.mxArray2double_matrixReal(mx_C);

    // display to application
    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();
        }
    }

    MessageBox.Show(“It has done”);
}
```

end code
Part III:
Creating and Using COM From MATLAB COM Builder In C#
Chapter 14

Using COM Created from MATLAB COM Builder in C# Applications

This chapter describes how to use COM created from MATLAB COM Builder in a simple C# Console Application. In this application we use MATLAB COM Builder version 1.1, MATLAB 7.1, and MATLAB Compiler 4.1 to create COM from an M-file then use it in a simple C# Console Application. The following are steps to create COM from an M-file and use it in C# functions:

1. Writing an M-file, say myplus.m

2. Use MATLAB COM Builder to create COM from the M-file myplus.m (the steps show how to do this task are in Section 14.1)

3. Using COM in C# functions

14.1 Creating COM From MATLAB COM Builder

To create COM from MATLAB, we do:

1. Write an M-file, for example myplus.m, as follows:

   function y = myplus(a, b)

   y = a + b;

2. Open MATLAB, type the command comtool, a new project setting box is appeared as shown in Fig. 14.1
3. In Fig. 14.1, click **File** and click **New Project**. The new project setting box is appeared as shown in Fig. 14.2.

4. In the **Component name** text box (Fig 14.2), enter the component name `myplusCOM`.

5. Click on the button **Add>>,** a default of a class name `myplusCOMclass` is created in the **Classes** text box (Fig 14.3). You can change another name by entering the name in the
Class name text box.

![Figure 14.3: Setting COM class name](image)

6. In Fig 14.3, click the button **Browse** to choose the directory where the COM will be created.

7. Click OK to finish the setting. We obtain Fig 14.4

![Figure 14.4: Setting COM Project](image)
8. In Fig 14.4, click **M-files** then click the button **Add File** to choose M-files for creating COM. In this example, we choose the M-file `myplus.m` (see Fig. 14.5).

![Image of choosing an M-file to create COM](image)

**Figure 14.5: Choosing an M-file to create COM**

9. In Fig. 14.4 click **Build** then click **COM Object**, the MATLAB COM Builder will create a COM `myplusCOM_1_0.dll` and `myplusCOM.ctf` in a folder `distrib` which is located in the same directory of the M-file.

### 14.2 Using COM in C# functions

In this section we'll describe the steps to use the created COM `myplusCOM_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 (MCRInstaller.exe is located in the directory `..\MATLAB7\toolbox\compiler\deploy\win32\`).

2. Create a normal C# Console Application, named `SimpleCOMbuilder`.

3. Copy two created files `myplusCOM_1_0.dll` and `myplusCOM.ctf` to the directory `bin\Debug` of the C# project (Fig. 14.6).

4. Register the COM `myplusCOM_1_0.dll` with Microsoft Windows:

   (a) Open a DOS window, change directories to `bin\Debug` where the COM `myplusCOM_1_0.dll` located.

   (b) Run the command `mwregsvr myplusCOM_1_0.dll` (Fig. 14.7).

5. Come back to the project `SimpleCOMbuilder`, right click on **References** (Fig. 14.8) then click **Add Reference**. The Add Reference dialog appears (Fig. 14.9)
Figure 14.6: Choosing an M-file to create COM

Figure 14.7: Registration of COM

Figure 14.8: Adding COM to References

Figure 14.9: Adding COM to References (cont.)
6. In Fig. 14.9, click on **Browse** then click the COM **myplusCOM_1_0.dll** (Fig. 14.10). Click OK to choose this COM.

![Figure 14.10: Adding COM to References (cont.)](image)

**Note:** Before going to the code, we want to mention some thing here:

(a) When we add the COM **myplusCOM_1_0.dll** to references, a class is created and named **myplusCOMclassClass**, see Fig. 14.11. This name is added a suffix **Class** to our predefined name **myplusCOMclass**, see Fig. 14.4

![Figure 14.11: Class in COM](image)

(b) The created function from the M-function **myplus(a, b)** is **myplus(int, ref object, object, object)** also shown in Fig. 14.11.

7. The following code demonstrates how to call the function myplus(..) in a C# function.

**Listing code**
using System;
using myplusCOM;

namespace SimpleCOMbuilder
{
    /// <summary>
    /// Summary description for Class1.
    /// </summary>
    class Class1
    {
        [STAThread]
        static void Main(string[] args)
        {
            Console.WriteLine("Hello COM Builder");
            Class1 obj = new Class1();
            obj.CallCOM();
        }

        public void CallCOM()
        {
            myplusCOMClassClass obj2 = new myplusCOMClassClass();

            double a = 1.2;
            double b = 3.4;

            object obj_a = a;
            object obj_b = b;

            object obj_y = (object) 0;

            obj2.myplus(1, ref obj_y, obj_a, obj_b);

            double y = (double) obj_y;
            Console.WriteLine(y.ToString());
        }
    }
}

end code
Chapter 15

Using COM Created from MATLAB COM Builder in C# Applications for Matrix Manipulation & Linear System Equations

This chapter describes how to use COM created from MATLAB COM Builder in a C# Console Application. This application is matrix multiplication and solving system linear equations, and we use MATLAB COM Builder version 1.1, MATLAB 7.1, and MATLAB Compiler 4.1 to create COM from M-files then use it in a C# Console Application.

15.1 Creating COM From MATLAB COM Builder

We'll write the following M-files to create a COM:

```matlab
function transposeA = mytranspose(A) ;

transposeA = transpose(A) ;

function C = mytimes(A, B)
```

function x = mymldivide(matrixA, vectorb)

% solve equation Ax = b
x = mldivide(matrixA, vectorb) ;

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

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

From these M-files we’ll use MATLAB COM Builder to create a COM, named MatrixCalculation_1_0.dll. The creating method and the procedure to add COM to C# project are the same as described in the previous chapter.

15.2 Using COM From MATLAB COM Builder To Calculate Matrix Multiplication

Problem 1

input Matrix A and B

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

output Finding the product matrix \( C = A \times B \)

This following code describes how to use the created COM MatrixCalculation_1_0.dll to calculate the matrix multiplication in Problem 1.
public void MatrixMultiplication()
{
    MatrixCalculationClass obj2 = new MatrixCalculationClass();

    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;
    obj2.mytimes(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);
}

end code
15.3 Using COM From MATLAB COM Builder To Solve Linear System Equations

Problem 2

input Matrix \( A \) and vector \( b \)

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

output . Finding the solution \( x \) of linear system equations, \( Ax = b \)
. Finding the lower \( L \) and upper \( U \) of the matrix \( A \)

This following code describes how to use the created COM MatrixCalculation_1_0.dll to solve Problem 2.

Listing code

```java
public void LinearSystemEquations()
{
    MatrixCalculationClass obj3 = new MatrixCalculationClass();

    /* 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 column vector \( b \)
    // note: type of obj_Transb is Double[,]
obj3.mytranspose(1, ref obj_Transb, obj_b);

Object obj_x = (object) 0;
obj3.mymldivide(1, ref obj_x, obj_A, obj_Transb);

// note: type of obj_x is Double[,]

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 */
    MatrixCalculationclass obj4 = new MatrixCalculationclass();
    obj4.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) ;

} /* ****************************************************** */
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() ;
    }
    Console.WriteLine() ;
}

end code

Note:
. Don't forget register COM before using.
. The created functions are shown in Fig. 15.1

Figure 15.1: Class in COM
Bibliography


Index

Band diagonal matrix, 56

call
  MATLAB workspace, 121
curve fitting, 81
eigenvalues, 113
eigenvectors, 113
Fast Fourier Transform, 103
feval, 61, 70
function-function, 64, 77, 102
fzero, 101
generate
dll-file, 5
functions, 5
inline function, 73
integrations, 73
interpolation, 87, 88
least-squares, 83
linear system equations, 43, 46, 124
LU
decompression method, 124
manuals, 4
MATLAB Engine, 121
matrix
  addition, 37
determinant, 41
inversion, 42
multiplication, 40, 144
subtraction, 39
transpose, 42
mwArray, 11
mxArray, 11
mxUtilityCompilerVer4.h, 16
ODE
  second order, 70
ordinary differential equations, 61
roots
  of a nonlinear, 101
  of a polynomial, 97, 98
sparse matrix, 49
tested, 4
tridiagonal, 52