C++ PROGRAM TO DRIVE AGILENT MOMENTUM

J.W. Bandler and M.A. Ismail

SOS-01-23-R

August 2001

C++ PROGRAM TO DRIVE AGILENT MOMENTUM

J.W. Bandler and M.A. Ismail

Simulation Optimization Systems Research Laboratory
and Department of Electrical and Computer Engineering
McMaster University, Hamilton, Canada L8S 4L7

Abstract This report presents a software tool to drive the Agilent Momentum™ electromagnetic simulator. The concept presented here can be used as a basis to drive any Windows based EM simulator. This tool mimics the steps performed by a microwave designer to simulate a microwave passive structure using Agilent Momentum. The user should prepare a text input file with the microwave structure dimensions and the frequency ranges. Then he runs Momentum-Driver executable file with the input file name as an argument. The Momentum-Driver calls Agilent Momentum simulator and saves the simulated results in a text output file, which contains the S-parameters of the structure of interest. Momentum-Driver has been used in various CAD programs in our laboratory.

I. INTRODUCTION

Momentum_Driver is a Windows based program to drive Agilent Momentum™ [1] from any programming environment such as C++, Matlab™, Fortran, etc. It is an object-oriented program written in Visual C++ [2]. It automatically performs the same steps performed by a user to simulate a microwave structure by Agilent Momentum. The Momentum-Driver program takes an input file (to be prepared by the user or by the CAD system) with the necessary information to simulate a microwave structure such as the structure's dimensions and the frequency ranges. Then it calls Agilent Momentum to perform electromagnetic simulation. Finally, it saves the simulated results (S-parameters) in an output file accessible to the user or the CAD system.

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada under Grants OGP0007239 and STR234854-00, through the Micronet Network of Centres of Excellence and Bandler Corporation. M.A. Ismail is supported by a Nortel Networks Ontario Graduate Scholarship in Science and Technology.

Momentum-Driver is an important tool for microwave design automation. It has been used in several microwave CAD programs [3,4]. In this report, we describe the general operation of the Momentum-Driver program as well as the user interface. We also provide an overview of the C++ source code.

II. MOMENTUM-DRIVER OPERATION

The operation of the program is illustrated in Fig. 1. First the user or the CAD program (we assume Microsoft Windows based programming environment) runs the executable file "Momentum_Driver.exe" which opens the input file "Input.dat" (this file is created by the user or the CAD program). The input file "Input.dat" contains the necessary information to simulate a microwave structure of interest such as the project name and directory, the design parameters and the frequency bands. Momentum_Driver.exe then runs ADS™[5] and launches Momentum™ with the specified microwave project. Then it creates a structure with the specified parameters. Next it opens the "Simulation" window, fills in frequency bands and launches Momentum simulator. Finally "Momentum_Driver.exe" commands Momentum™ to export the simulated results (S-parameters) to the file "Momentum_Output.dat". Then, it reads this output file and saves its contents in a certain format in the file "Output.dat".

The following command line runs Momentum_Driver

Momentum_Driver [-h] <Input.dat> <Output.dat>

where [-h] is optional and is used for help on how to use Momentum_Driver, "Input.dat" is a text input file (other names are allowed) containing all the necessary information about the Momentum project and it takes the following format

[Momentum Project Directory]
[Design File Name]
[Number of Parameters] [Number of Points]
[Point Number 1]
[Point Number 2]
⋮
[Point Number n]
[Number of Ports] [Number of Frequency Bands]

[Freq_Start] [Freq_Stop] [Freq_Step]
[Freq_Start] [Freq_Stop] [Freq_Step]
⋮
[Freq_Start] [Freq_Stop] [Freq_Step]

where Freq_Start, Freq_Stop and Freq_Step stand for the lower band edge frequency, the upper band edge frequency and the incremental frequency step for the band.

The output file "Output.dat" is generated by the Momentum_Driver program and it contains the real and imaginary parts of S-parameters at all points and over all frequency bands. In this file the real and imaginary part of the microwave structure at a single frequency takes the form

$RS_{11}\ IS_{11}\ldots\ RS_{1M}\ IS_{1M}\ RS_{21}\ IS_{21}\ldots RS_{2M}\ IS_{2M}\ RS_{M1}\ IS_{M1}\ldots RS_{MM}\ IS_{MM}$

where RS_{11} and IS_{11} stand for the real and imaginary part of the scattering parameter S_{11} and M is the number of ports.

III. MOMENTUM-DRIVER PROGRAMMING DETAILS

In this Section, we give a detailed description of the Momentum-Driver C++ code. The main object in the Momentum-Driver program is the class "Momentum" which is derived from the abstract class "Simulator" in [3]. The "Simulator" class is an abstract data type for all possible simulators. The implementation of this class is given in Appendix A. The reader is referred to [3] for details about the "Simulator" class. The "Simulator" class uses the matrix class "Matrix" in Appendix B for vectors and matrix manipulations.

Momentum Class

A Momentum object is used to drive Agilent Momentum™ [1] to simulate a microwave structure of interest. The source code for the Momentum class is given in Appendix C. The data members and their description are given in Table I. The public class members and private class members are given in Table II and Table III, respectively. Most of the member functions are easy to understand by the aid of the comments attached to them.

Momentum::FindWindow

int Momentum::FindWindow(char *WindClass, char *WindTitle, CWnd *&WndPtr, double DurationInSec)

Return Value

return 0 if a Window of interest is found otherwise return 1

input parameters

WindClass: the class name of the Window of interest.

WindTitle: the title of the Window of interest.

WndPtr: a pointer to the Window of interest. This pointer is to be obtained by the member function.

DurationInSec: the function keeps looking for the Window for a time period of DurationInSec.

Remarks

The function FindWindow obtains the pointer of the Window of interest and it assigns it to the input parameter WndPtr (note that this parameter is passed by reference to the member function). The Window's class name and title are obtained using the Microsoft Visual C++ tool "Spy ++". To get the class name and title of an open Window using Spy++ we perform the following steps

Step 1 Open the Window of interest.

Step 2 Run the Microsoft Visual C++ tool "Spy++".

Step 3 Press Ctrl+F or choose the "Spy" menu and the "Find Window" item.

Step 4 Drag the "Finder Toll" icon and move it to the top of the Window of interest.

Step 5 Read the Window title from the "Caption" edit box and the class name from the "Class" edit box.

Spy++ is also used to get the identification number ID of a Button Dialog box or an Edit Dialog box or any other dialog box. This ID is needed for pressing a Button or writing in an Edit box. To get the ID of a specific Dialog box choose the "Spy" from the main menu and click on

the “Windows” item. Spy will open a window with all dialog boxes. Just click on the dialog box of interest and it will show a window with all information about the dialog box.

IV. EXAMPLE

In this example, we consider the microstrip bandstop filter with open stubs in Fig. 2. The filter parameters are W_1, W_2, L_0, L_1 and L_2. The filter is parameterized in Agilent Momentum and the nominal layout is shown in Fig. 3. The input file for Momentum-Driver program is

```
E:\examples\BStopF_mom_prj
BStopF_nom.dsn
5   1
5   10   120   120   120
2   3
5   8   1
9   11   0.25
12  15   1
```

where the microstrip bandstop filter is simulated at the point W_1=5 mil, W_2=10 mil, L_0=120 mil, L_1=120 mil and L_2= 120 mil over the frequency bands 5 GHz to 8 GHz with a frequency step of 1 GHz, 9 GHz to 11 GHz with a frequency step of 0.25 GHz and 12 GHz to 15 GHz with a frequency step of 1 GHz.

The content of the output file generated by Momentum_Driver is

```
-0.009119  0.010952 -0.775801 -0.626588 -0.775801  -0.626588  -0.009109   0.010961
 0.003383 -0.184941 -0.978425 -0.0177     -0.978425 -0.0177      0.003404 -0.184941
-0.180784 -0.177166 -0.660442  0.694359 -0.660442   0.694359   -0.180754 -0.177198
 0.265666 -0.681888  0.564753  0.264993  0.564753   0.264993    0.265551 -0.681936
-0.832124 -0.520346  0.010263  0.000544  0.010263  0.000544    -0.832166 -0.52028
-0.915579 -0.354741  0.002534  0.015021 0.002534  0.015021 -0.915606 -0.354671
-0.963203 -0.189628  0.003808  0.017706 0.003808  0.017706 -0.963218 -0.189554
-0.980776 -0.023784  0.006914  0.012963 0.006914  0.012963 -0.980778 -0.023707
-0.969209  0.144564  0.008633  0.005253 0.008633  0.005253 -0.969197   0.144644
-0.925085  0.317589  0.009219  1.9e-005  0.009219  1.9e-005  -0.925056   0.317673
-0.838206  0.498008  0.015747  0.005309 0.015747   0.005309 -0.838154  0.498094
-0.68088    0.68758   0.052362  0.029807 0.052362   0.029807 -0.680792  0.687665
-0.367492  0.857775  0.201777  0.054805 0.201777   0.054805 -0.367341  0.857836
-0.258549  0.366671 -0.699751 -0.511677 -0.699751 -0.511677 -0.258594 0.366637
 0.077751   0.177318 -0.889097  0.35538  -0.889097   0.35538    0.07773   0.177327
-0.092225 -0.050086 -0.390055  0.88509  -0.390055  0.88509  -0.092236 -0.050063
-0.243701   0.034075  0.17604   0.920942  0.17604   0.920942 -0.243697  0.034103
```

V. CONCLUSIONS

We have presented a software tool to drive the Agilent Momentum™ electromagnetic simulator. The operation of the driver is explained as well as the user interface. An example of how to use the Momentum-Driver program is also presented. Relevant programming details are addressed.

ACKNOWLEDGEMENT

The authors would like to thank their colleague Q.S. Cheng for useful discussion on Microsoft Visual C++ tool "Spy++". The authors also thank Agilent Technologies, Santa Rosa, CA, for making Momentum™ available.

REFERENCES

[1] Momentum™ Version 3.5, Agilent Technologies, 1400 Fountaingrove Parkway, Santa Rosa, CA 95403-1799, 1999.

[2] Microsoft Visual C++ Version 6.0, Microsoft Corporation.

[3] M.H. Bakr, J.W. Bandler, Q.S. Cheng, M.A. Ismail and J.E. Rayas-Sánchez, "SMX—A novel object-oriented optimization system," *IEEE MTT-S Int. Microwave Symp. Digest* (Phoenix, AZ), 2001, pp. 2083-2086.

[4] J.W. Bandler, M.A. Ismail and J.E. Rayas-Sánchez, "Expanded space mapping design framework exploiting preassigned parameters," *IEEE MTT-S Int. Microwave Symp. Digest* (Phoenix, AZ), 2001, pp. 1151-1154.

[5] ADS™, Agilent Technologies, 1400 Fountaingrove Parkway, Santa Rosa, CA 95403-1799, 1999.

TABLE I

DATA MEMBERS OF THE MOMENTUM CLASS

Data Member	Data Type	Description
m_sOutputFileName	CString	Contains the output file name
m_sInputFileName	CString	Contains the input file name
m_sLayoutCaption	CString	Contains the layout caption (to be used to capture the window of the layout)
m_sProjectName	CString	contains the full path of the project
m_sDesignName	CString	contains the design file name (nominal file name)
m_iHelpFlag	int	Show help window if it is 1
m_iCurrentPoint	int	The current point index (a point is a set of parameter values)
m_iNofPoints	int	Total number of points to be simulated
m_iLayoutNumber	int	Contains the layout number (this number is written in the layout window caption and it is used to capture the window of the created layout
m_iErrorFlag	int	0 if there is no error otherwise it takes a specific value depending on the error (see Momentum.h in Appendix C)
m_pAllParameters	double *	Contains the value of all design parameters

TABLE II

PUBLIC CLASS MEMBERS OF THE MOMENTUM CLASS

Class Member (public)	Description
Momentum	The class construction
ReadInput	Read input file
SetAllParameters	Set the parameters of the current design
SetNofPoints	Set the number of points to be simulated
UncheckDlgButton	Uncheck the dialog point
PressButton	Press a button
FillEditBox	Fill in an edit box with numbers or text
FindWindowExt	Find a Microsoft window extension
FindChildWindow	Find a child window
Test	This function was used during the debugging of the Momentum class
SetMultiplePointOption	Set the multiple point option to simulate the structure at different points
WriteInputFiles	Write the necessary input files
Simulate	Launch Momentum™ simulator to simulate the microwave structure
GetResult	Get the simulated results

TABLE III

PUBLIC CLASS MEMBERS OF THE MOMENTUM CLASS

Class Member (private)	Description
CallHelp	Show the help dialog box
GetMomDataFileName	Get the file name of the generated data file which takes the form "simdata.s$p" where $ stands for the port number
DesignNotFound	check if the nominal file does not exist
WriteOutputFile	Write the output file which contains the simulated data
SaveResults	Save the simulated results
WaitForSimulation	Wait for the Momentum until the simulation ends
SetLayoutWndCaption	Set the caption of the layout window in order to capture this window when it appears
OpenDesignFile	open the nominal design file which is used to generate a new structure
SetSimulationWindow	this function fills in the frequencies of interest and launches the Momentum simulator
SetIniLayoutNumber	set the layout number which is used to open the parameter and the simulator windows
CannotOpenDesign	return 1 if the program cannot open the layout design otherwise return 0
PressSubMenuItem	this function activates an item in a sub menu
FindWindow	Keep looking for a window until finding it or exits after some time
OpenAdsPrj	Open the ADS™ project
SetParametersWindow	set the values of the design parameters
RunAds	Run Agilent ADS
WriteAdsMacroFile	Write the Agilent ADS Macro file (this Macro file just opens the project of interest)
CloseAds	Exit from Agilent ADS

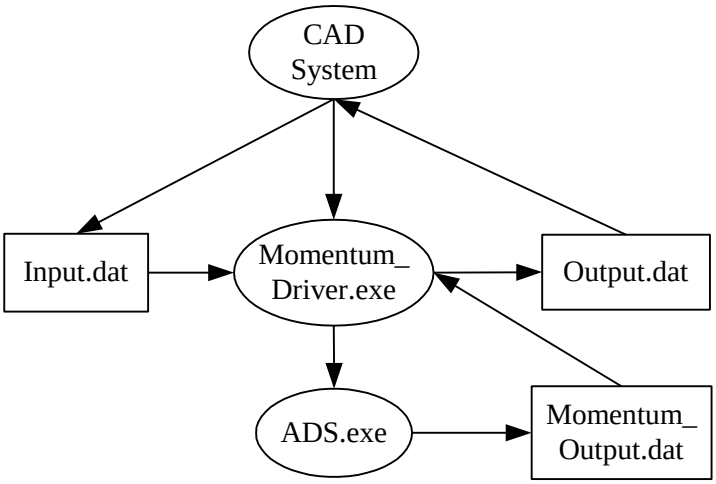

Fig. 1. Driving Momentum™ from a microwave CAD system.

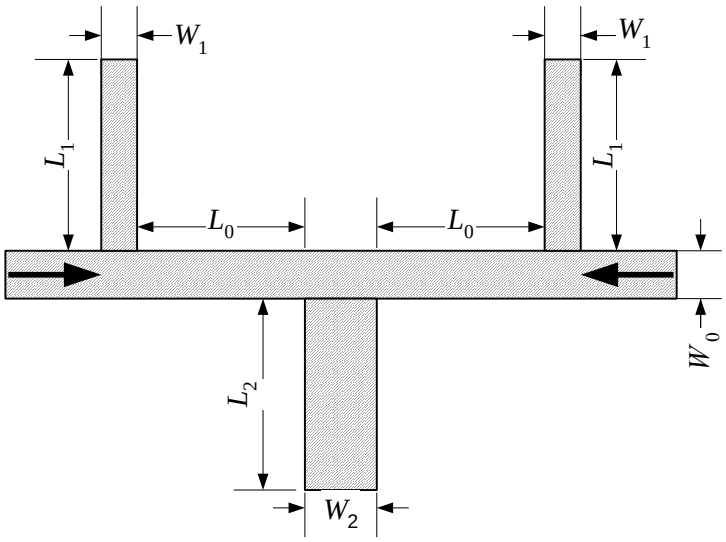

Fig. 2. Microstrip bandstop filter with open stubs.

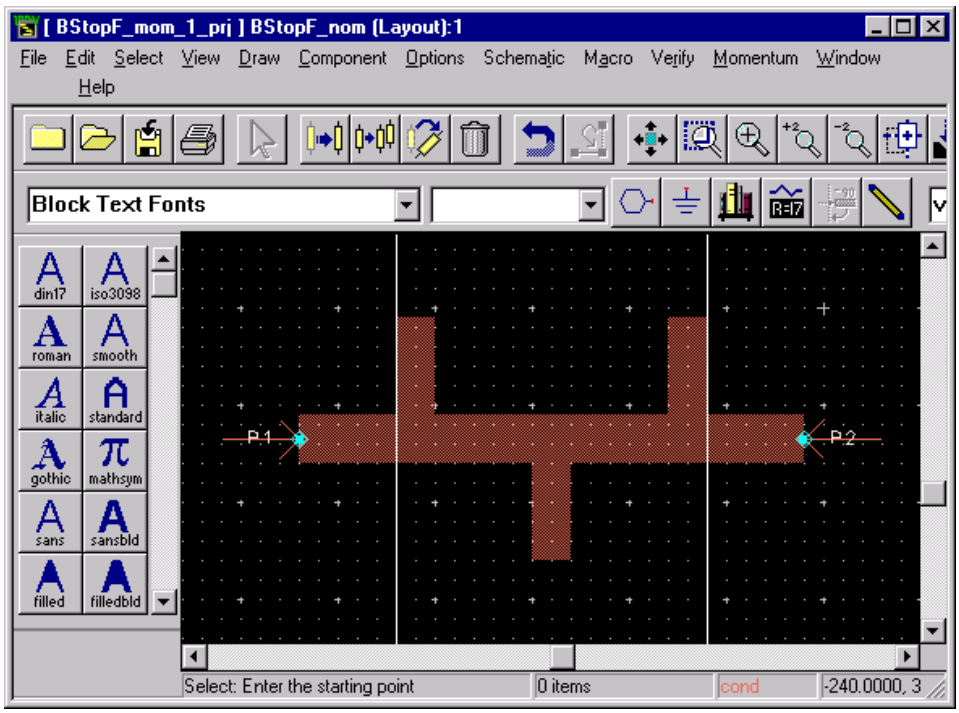

Fig. 3. The nominal layout of the microstrip bandstop filter with open stubs in Fig. 2.

APPENDIX A

Simulator.h

```
//Simulator.h: interface for the Simulator class
//This class is an Abstract Data Type for all possible simulators
//Any Future Simulator should be derived class
//Only the purely virtual functions will have to be overwritten in
//the new class
//////////////////////////////////////////////////////////////////////
#if !defined(AFX_SIMULATOR_H__C3173FDE_0A71_11D4_835E_006094EB08FA__INCLUDED_)
#define AFX_SIMULATOR_H__C3173FDE_0A71_11D4_835E_006094EB08FA__INCLUDED_
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
#include "Matrix.h"
#include <fstream.h>
const WM_PLOTOBJECTIVE=WM_USER+100;
const WM_PRINTSTRING=WM_USER+101;
const WM_RESETPROGRESSBAR=WM_USER+102;
const WM_INCREMENTPROGRESSBAR=WM_USER+103;
class Simulator : public CObject
{
public:
                sMatrix* GetRequiredResponse(sMatrix& RealS, sMatrix& ImagS, sMatrix& RealZ, sMatrix& ImagZ,
                sMatrix& RealY, sMatrix& ImagY);
                int ConvertZToY(sMatrix& sRealZMatrix,sMatrix& sImagZMatrix, sMatrix& sRealYMatrix, sMatrix&
                sImagYMatrix,int iNoPorts);
                int ConvertSToZ(sMatrix& SReal,sMatrix& SImag, sMatrix& ZReal,sMatrix& ZImag,int iNoPorts, double
                dZRefence);
        int SetPathName(CString *pPathName);
        int SetResponses(int iNoResponses, CString *pResponses);
        int SetFilesList(int , CString* );
        int SetPoint(int iNoParameters, double *pParameters);
        void SetNoParameters(int iNoParameters);
        int SetFrequencies(int NoFrequencyBands, int NoFrequencies, sMatrix *pFrequencyBands);
        int GetNoFrequencies();
        int GetNoResponses();
        int GetNoParameters();
        virtual int GetResult(sMatrix&)=0;//This function obtains the simulation results
        virtual int Simulate()=0;//This function calls the simulator
        virtual int WriteInputFiles()=0;//This function prepares the input files
        virtual void SetMultiplePointOption(BOOL bMultiplePoints)=0;
        Simulator();//Constructor
        virtual ~Simulator();//Destructor
        void SetNoPorts(int iNoPorts);
protected:
        CString* m_pPathName;//The Path Name where all project files are aved
        int m_iNoAssistantFiles; // Number of assistant files
        CString * m_pAssistantFiles;//List of assistant files
        double * m_pParameters;//Parameters
        int m_iNoParameters;//Number of Parameters
        int m_iNoFrequencies;//No of simulation frequencies
        double * m_pFrequencies;//List of simulation frequencies
        int m_iNoFrequencyBands;//Number of Frequency Bands
        sMatrix *m_pFrequencyBands;// Matrix containing the definition of frequency Bands
        CString * m_pResponses;// Pointer to the names of the responses
        CPtrArray m_pSimulatedPoints;
        int m_iNoResponses;//Number of Simulation responses
        int m_iNoPorts;

};
#endif // !defined(AFX_SIMULATOR_H__C3173FDE_0A71_11D4_835E_006094EB08FA__INCLUDED_)
```

Simulator.cpp

```
// Simulator.cpp: implementation of the Simulator class.
//////////////////////////////////////////////////////////////////////
#include "stdafx.h"
#include "Simulator.h"
#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif
//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////
//The Constructor Initializes all the pointer members to NULL
Simulator::Simulator()
{
	m_pPathName=NULL;
	m_pAssistantFiles=NULL;
	m_pParameters=NULL;
	m_pFrequencies=NULL;
	m_pFrequencyBands=NULL;
	m_pResponses=NULL;
}

//The destructor clears the memory assigned to the pointer
Simulator::~Simulator()
{
	if(m_pPathName!=NULL)
	{
		delete m_pPathName;
		m_pPathName=NULL;
	}
	if(m_pAssistantFiles!=NULL)
	{
		delete [] m_pAssistantFiles;
		m_pAssistantFiles=NULL;
	}
	if(m_pParameters!=NULL)
	{
		delete [] m_pParameters;
		m_pParameters=NULL;
	}
	if(m_pFrequencies!=NULL)
	{
		delete [] m_pFrequencies;
		m_pFrequencies=NULL;
	}
	if(m_pFrequencyBands!=NULL)
	{
		delete m_pFrequencyBands;
		m_pFrequencyBands=NULL;
	}
	if(m_pResponses!=NULL)
	{
		delete [] m_pResponses;
		m_pResponses=NULL;
	}
}

//The function SetFrequencies sets the simulation frequencies of the simulator
//it recieves the Number of frequency bands, the total number of frequencies and
// a pointer to a matrix that contains the band definitions

int Simulator::SetFrequencies(int NoFrequencyBands, int NoFrequencies, sMatrix *pFrequencyBands)
{
	int i,j,NoBandFrequencies;
	double *pTempPtr1;
	double dTempDouble,dBandStart,dBandStop,dBandStep;
```

```
	if(m_pFrequencies!=NULL) //simulation frequncies has been set before
	{
		delete [] m_pFrequencies;//Clear previously allocated memory
		delete m_pFrequencyBands;
	}
	m_pFrequencies=NULL;
	m_pFrequencyBands=NULL;
	m_pFrequencies=new double[NoFrequencies];//Allocate new memory
	m_pFrequencyBands=new sMatrix(mDOUBLE,NoFrequencyBands,3);//
	if((m_pFrequencies==NULL)||(m_pFrequencyBands==NULL))
		return (-1); //-1 means memory allocation has failed
	m_iNoFrequencies=NoFrequencies;//Set Number of simulation frequencies
	m_iNoFrequencyBands=NoFrequencyBands;//Set Number of frequency bands
	//The following loop copies the band matrix
	for(i=1;i<=NoFrequencyBands;i++)
	{
		for(j=1;j<=3;j++)//There are three columns; BandStart, BandStop and BandStep
		{
			pFrequencyBands->GetElement(&dTempDouble,i,j);
			m_pFrequencyBands->Populate(&dTempDouble,i,j);
		}
	}
	//The following loop sets the frequency vector
	pTempPtr1=m_pFrequencies;
	for(i=1;i<=NoFrequencyBands;i++) //Repeat for all frequency bands
	{
		m_pFrequencyBands->GetElement(&dBandStart,i,1);//Get band start
		m_pFrequencyBands->GetElement(&dBandStop,i,2);//Get band stop
		m_pFrequencyBands->GetElement(&dBandStep,i,3);//Get band step
		if(dBandStep==0)
			NoBandFrequencies=1; //if BandStep=0 then The band contains one frequency
		else
			NoBandFrequencies=((int)((dBandStop-dBandStart)/dBandStep))+1;

		for(j=0;j<NoBandFrequencies;j++)
		{
			(*pTempPtr1)=dBandStart+(j*dBandStep);//Assign the jth frequency inside the band
			pTempPtr1++;//Move to set the next frequency in the band
		}
	}
	return 0;
}

//This function sets the number of designable parameters and assigns value for each
//It recieves two inputs; the first one is the number of parameters and a pointer
//to the parameters list.  It returns a negatve value if error occurs
//Notice that this vector may correspond to multiplt points in case of surrogate
//model optimization
int Simulator::SetPoint(int iNoParameters, double *pParameters)
{
	int i;
	double *pTempPtr1, *pTempPtr2;
	if(m_pParameters!=NULL) //Parameters have  been set before
		delete [] m_pParameters;//Clear previously allocated memory
	m_pParameters=NULL;
	m_pParameters=new double[iNoParameters];//Allocate new memory
	if(m_pParameters==NULL)
		return (-1); //-1 means memory allocation has failed
	//The following loop sets the values of the parameters
	pTempPtr1=pParameters;
	pTempPtr2=m_pParameters;
	for(i=0;i<iNoParameters;i++)
	{
		(*pTempPtr2)=(*pTempPtr1);
		pTempPtr1++;
		pTempPtr2++;
	}
	return 0;
}
//The following functions sets a list for the assistant files needed
```

```
//to get the responses;  The utility of these files is problem dependent
//This function recieves two input; the number of files and a pointer
// to the file list

int Simulator::SetFilesList(int iNoAssistantFiles, CString *pAssistantFiles)
{
	int i;
	CString *pTempPtr1, *pTempPtr2;
	if(m_pAssistantFiles!=NULL) //Files have  been set before
		delete [] m_pAssistantFiles;//Clear previously allocated memory
	m_pAssistantFiles=NULL;
	m_pAssistantFiles=new CString[iNoAssistantFiles];//Allocate new memory
	if(m_pAssistantFiles==NULL)
		return (-1); //-1 means memory allocation has failed
	m_iNoAssistantFiles=iNoAssistantFiles;//Set Number of Files
	//The following loop sets the contents of the files list
	pTempPtr1=pAssistantFiles;
	pTempPtr2=m_pAssistantFiles;
	for(i=0;i<m_iNoAssistantFiles;i++)
	{
		(*pTempPtr2)=(*pTempPtr1);
		pTempPtr1++;
		pTempPtr2++;
	}
	return 0;
}

//This function sets the Number of simulated responses and their names
// It recieves two inputs, the first one is the Number of responses
//The second input is the list of response names

int Simulator::SetResponses(int iNoResponses, CString *pResponses)
{
	int j;
	//CString *pTempString;
	if(m_pResponses!=NULL) //Responses names have been set before?
		delete [] m_pResponses;//Clear previously allocated memory
	m_pResponses=NULL;
	m_pResponses=new CString [iNoResponses];// Allocate new memory
	if(m_pResponses==NULL)
		return(-1);//Memory Could not be allocated
	m_iNoResponses=iNoResponses;//Set Number of Responses
	//pTempString=pResponses;
	for(j=0;j<iNoResponses;j++)
		m_pResponses[j]=pResponses[j];//Copy the response names
	return 0;
}

//This function sets the Path name of all the files that will be generated
//Through the simulator
int Simulator::SetPathName(CString *pPathName)
{
	if(m_pPathName!=NULL) //Path Name was allocated before?
		delete m_pPathName; //Clear allocated memory
	m_pPathName=NULL;// assign cleared vector to zero
	m_pPathName=new CString;// assign new memory
	if(m_pPathName==NULL)
		return -1; //Memory allocation failed
	(*m_pPathName)=(*pPathName);//Copy path name
	return 0;
}

// This function convert from the S matrix to the Z matrix
// It recieves the Real(S) and Imag(S) matrix and then calculates Real(Z) and Imag(Z)

	int Simulator::ConvertSToZ(sMatrix &SReal, sMatrix &SImag, sMatrix &ZReal, sMatrix &ZImag,int iNoPorts, double
	dZReference)
{
	int i,j;
	double dComp1, dComp2,dComp3;
```

```
	double dRS,dIS;
	sMatrix *A=new sMatrix(mDOUBLE,2*iNoPorts,2*iNoPorts);
	if(A==NULL)
		return -1;
	sMatrix *B=new sMatrix(mDOUBLE,2*iNoPorts,iNoPorts);
	if(B==NULL)
	{
		delete A;
		return -1;
	}
	//The next loop fills the matrices A and B
	for(i=1;i<=iNoPorts;i++)
	{
		for(j=1;j<=iNoPorts;j++)
		{
			SReal.GetElement(&dRS,i,j);//This is  ReSij
			SImag.GetElement(&dIS,i,j);//This is Im(Sij)
			if(i==j) //Aii=1-RSii
			{
				dComp1=1-dRS;
				dComp3=1+dRS;//Bii=1+ReSii
			}
			else
			{
				dComp1=-1*dRS;//location Aij=-ReSij
				dComp3=dRS;//Bij=dRSij
			}
			dComp2=dIS;
			A->Populate(&dComp1,i,j);//Top Left Block of A=I-RS
			A->Populate(&dComp1,(i+iNoPorts),(j+iNoPorts));//Bottom Right Block of A=(i-RS)
			A->Populate(&dComp2,i,(j+iNoPorts));//Top Right Block=IS
			dComp2=-1.0*dComp2;
			A->Populate(&dComp2,(i+iNoPorts),j);//Bottom Left Block=-1*IS
			B->Populate(&dComp3,i,j);//Fill top block of B=I+RS
			B->Populate(&dIS,(i+iNoPorts),j);//Fill bottom block of B=IS
		}
	}
	sMatrix *X=&(A->SolveEquationsSystem(B));//Solve the system of equations AX=B
	//The matrix X=[ ZR
	//		ZI] normalized with respect to the reference impedance
	//The following loop copies the contents of the X matrix to the ZR and ZI matrices
	for(i=1;i<=iNoPorts;i++)
	{
		for(j=1;j<=iNoPorts;j++)
		{
			X->GetElement(&dComp1,i,j);
			X->GetElement(&dComp2,(i+iNoPorts),j);
			dComp1=dComp1*dZReference;//Multiply by referenece impedance
			dComp2=dComp2*dZReference;//Multiply by referenece impedance
			ZReal.Populate(&dComp1,i,j);//Store scaled impedances
			ZImag.Populate(&dComp2,i,j);//Store scaled impedances
		}
	}
	delete A;//Free Allocated memory
	delete B;//Free Allocated memory
	delete X;//Free Allocated
	return 0;
}

	int Simulator::ConvertZToY(sMatrix &sRealZMatrix, sMatrix &sImagZMatrix, sMatrix &sRealYMatrix, sMatrix&
	sImagYMatrix, int iNoPorts)
{
	int i,j;
	double dComp1, dComp2,dComp3,dComp4;
	double dRZ,dIZ;
	sMatrix *A=new sMatrix(mDOUBLE,2*iNoPorts,2*iNoPorts);
	if(A==NULL)
		return -1;
	sMatrix *B=new sMatrix(mDOUBLE,2*iNoPorts,iNoPorts);
	if(B==NULL)
```

```
	{
		delete A;
		return -1;
	}
	//The next loop fills the matrices A and B
	for(i=1;i<=iNoPorts;i++)
	{
		for(j=1;j<=iNoPorts;j++)
		{
			sRealZMatrix.GetElement(&dRZ,i,j);//This is  Re(Zij)
			sImagZMatrix.GetElement(&dIZ,i,j);//This is Im(Zij)
			dComp1=dRZ;
			dComp2=-1*dIZ;
			A->Populate(&dComp1,i,j);//Top Left Block of A=ZR
			A->Populate(&dComp1,(i+iNoPorts),(j+iNoPorts));//Bottom Right Block of A=ZR
			A->Populate(&dComp2,i,(j+iNoPorts));//Top Right Block=-IZ
			dComp2=-1.0*dComp2;
			A->Populate(&dComp2,(i+iNoPorts),j);//Bottom Left Block=IZ
			if(i==j)
				dComp3=1;
			else
				dComp3=0;
			dComp4=0;
			B->Populate(&dComp3,i,j);//Fill top block of B=I
			B->Populate(&dComp4,(i+iNoPorts),j);//Fill bottom block of B=0
		}
	}
	sMatrix *X=&(A->SolveEquationsSystem(B));//Solve the system of equations AX=B
	//The matrix X=[ YR
	//		YI] normalized with respect to the reference impedance
	//The following loop copies the contents of the X matrix to the YR and YI matrices
	for(i=1;i<=iNoPorts;i++)
	{
		for(j=1;j<=iNoPorts;j++)
		{
			X->GetElement(&dComp1,i,j);
			X->GetElement(&dComp2,(i+iNoPorts),j);
			sRealYMatrix.Populate(&dComp1,i,j);//Store scaled impedances
			sImagYMatrix.Populate(&dComp2,i,j);//Store scaled impedances
		}
	}

	delete A;//Free Allocated memory
	delete B;//Free Allocated memory
	delete X;//Free Allocated
	return 0;
}
void Simulator::SetNoPorts(int iNoPorts)
{
	m_iNoPorts=iNoPorts;
}

int Simulator::GetNoFrequencies()
{
	return(m_iNoFrequencies);
}
int Simulator::GetNoResponses()
{
	return(m_iNoResponses);
}
int Simulator::GetNoParameters()
{
	return m_iNoParameters;
}

// This function examines the strings of required response and returns a vector
//
	sMatrix* Simulator::GetRequiredResponse(sMatrix &RealS, sMatrix &ImagS, sMatrix &RealZ, sMatrix &ImagZ, sMatrix
	&RealY, sMatrix &ImagY)
{
```

```
	int i;
	sMatrix *Result=new sMatrix(mDOUBLE,1,m_iNoResponses);//Assigne result vector
	if(Result==NULL)
		exit(-1);
	sMatrix *sTempMatrixReal,*sTempMatrixImag;
	CString *cTempString=m_pResponses;//A pointer to the strings list
	char char1,char2;
	int index1,index2;
	double dReal,dImag, dModulus,dValue;
	for(i=1;i<=m_iNoResponses;i++)//Repeat for all responses
	{
		if((*cTempString)[1]=='z')//it is assumed here that responses are converted to
		{		//lower case
			sTempMatrixReal=&(RealZ);
			sTempMatrixImag=&(ImagZ);
		}
		if((*cTempString)[1]=='s')
		{
			sTempMatrixReal=&(RealS);
			sTempMatrixImag=&(ImagS);
		}
		if((*cTempString)[1]=='y')
		{
			sTempMatrixReal=&(RealY);
			sTempMatrixImag=&(ImagY);
		}
		char1=(*cTempString)[2];//Third digit
		char2=(*cTempString)[3];//Fourth digit
		index1=atoi(&char1);
		index2=atoi(&char2);
		sTempMatrixReal->GetElement(&dReal,index1,index2);
		sTempMatrixImag->GetElement(&dImag,index1,index2);
		dModulus=sqrt(dReal*dReal+dImag*dImag);
		if((*cTempString)[0]=='r')//Required is the real
			dValue=dReal;
		if((*cTempString)[0]=='i')//Required is the imaginary
			dValue=dImag;
		if((*cTempString)[0]=='m')//Required is the modulus
			dValue=dModulus;
		Result->Populate(&dValue,1,i);
		cTempString++;
	}
	return(Result);
}
//This is a virtual function that is going to be overwritten by child classes
void Simulator::SetMultiplePointOption(BOOL bMultiplePoints)
{
  return;
}
//This function sets the number of parameters
void Simulator::SetNoParameters(int iNoParameters)
{
	m_iNoParameters=iNoParameters;
}
```

APPENDIX B

Matrix.H

```
/*****************************************************************
**  Matrix.H
**      Definition of sMatrix Class
**      April 21-1998
*****************************************************************/


#ifndef __SMATRIX_H__
   #define __SMATRIX_H__
#include <math.h>
#include <iostream.h>
#define mCHAR                   0
#define mINT                    1
#define mLONG                   2
#define mFLOAT                  3
#define mDOUBLE                 4
#define mMAX                    5

#ifdef WIN32
   #pragma warning( disable : 4244 )
#endif

class sMatrix{

   /*  constructors and destructors */

   public:
                sMatrix();
      sMatrix(int );          //constructor which takes type as an argument;
      sMatrix(int, int, int); //constructor take type and sizes;
                        sMatrix(double *,int,int);//convert from pointer to a matrix
                virtual ~sMatrix();

   /*  member functions */

   public:

      /* Functions for filling elements */
      int Populate(void *, int, int);  //Function to fill a matrix element;
      int PopulateRow(void *, int);    //Function to fill an entire row;
      int PopulateColumn(void *, int); //Funciton to fill a column;
      int PopulateArea(void *, int, int, int, int);  //Fill an area see note1
      int PopulateArea(sMatrix *, int, int);  //see note 4

      /* Functions for retrieving specific elements see note3*/
      int GetElement(void *, int, int);   //Functions act like their populate
      int GetRow(void *, int);            //counterparts, except returning
      int GetColumn(void *, int);         //information.
                           int GetColumn(sMatrix *,int);
      int GetArea(void *, int, int, int, int);
      int GetArea(sMatrix *, int, int);   //see note 5

      /* Functions for relating to the data type */
      int SetType(int );      //Set type of data
      int GetType(int *);     //Get the type of data

      /* Functions relating to size of the matrix */
      int SetSize(int, int);      //Set Matrix Size
      int GetSize(int *, int *);      //Get Matrix Size
      int CompareSize(int, int, int*);  //Compare Matrix see note2
      int CompareSize(sMatrix *, int*);  //Compare Matrix see note2
      //matrix resizing functions should be added here.

      /* Math Operations */
         /* overloaded operators for simple math */
```

```
        sMatrix& operator+ (sMatrix& );  //overloaded addition
                                        sMatrix& operator= (sMatrix&);//Equality operator
                                        void operator+=(sMatrix &);//Increment Operator
        sMatrix& operator- (sMatrix& );  //overloaded subtraction
        sMatrix& operator* (sMatrix& );  //overloaded multiplication
        sMatrix& operator* (double& );   //overloaded multiplication for a constant
        sMatrix& operator* (float& );    //overloaded multiplication for a constant
        sMatrix& operator* (int& );      //overloaded multiplication for a constant
        sMatrix& operator* (long& );     //overloaded multiplication for a constant
        sMatrix& operator* (char& );     //overloaded multiplication for a constant
        /* non-operator based math functions */
        int Transpose(sMatrix& );       //function to transpose matricies
      //   int SVD(sMatrix & /*u*/, sMatrix & /*w*/, sMatrix & /*vT*/);
            //See notes in code for description of modifications to SVD
       //  int Psuedoinv(sMatrix &);       //function to calculate non-destructive
                                //psuedoinverse
        // int Chol(sMatrix &);           //see note 6 at the end of this file
                                        int BiDiagonalize(sMatrix *Bptr, sMatrix *Wptr, sMatrix *rv1, double *aNormPtr);
                                        int AccmlteLeft(sMatrix *U, sMatrix *WPtr);
                                        int AccmlteRight(sMatrix *V, sMatrix *WPtr);
                                        int SVDA(sMatrix *WPtr, sMatrix *rv1, sMatrix *UPtr,sMatrix *VPtr,int m, int n,double aNorm);
                                        int PrintMatrix(int startx, int starty, int endx, int endy);
                                        int PsuedoInverse(sMatrix *BInverse);
                                        int BroydenUpdate(sMatrix* RNew, sMatrix* ROld, sMatrix* Step);
                                        double GetL2Norm();
    private:

    protected:

       /*Helper functions.. these are only called by the class */

       inline int HGetElement(void *, int, int);  //function to get elements
       inline int HSetElement(void *, int, int);  //function to set elements
                               double GetSign(double a, double b);
                               double GetMax(double a, double b);
                               double GetPythag(double a, double b);
                               int Rotate(double *aOut, double *bOut, double aIn, double bIn, double c, double s);

    /* data members */

    public:
                    void * GetPointer();
                    sMatrix& SolveEquationsSystem(sMatrix* b);

    private:

    protected:
       void * m_data;      //Matrix data container
       int m_x;          //Matrix X dimension (1 based)
       int m_y;          //Matrix Y dimension (1 based)
       int m_datatype;     //Type of data stored in matrix
       int m_datatypesize; //storage requirements in memory for data type


    };

#endif

/*****************************************************************************
**  Note 1:  PopulateArea's arguments are as follows:
**  PopulateArea(   void *,     Data to fill area with (1 value)
**                  int,        Starting row
**                  int,        Starting column
**                  int,        Number of rows
**                  int,        Number of columns
**
**  All population is inclusive, that is if:
**      Row = 2, Column = 2, rows =3 and Columns = 3,
**      Elements:       (2,2), (3,2), (4,2)
**                      (2,3), (3,3), (4,3)
```

```
**                (2,4), (3,4), (4,4)
**
**  all get set making 3 rows and columns that were set
*****************************************************************************/

/*****************************************************************************
**  Note 2:  Matrix Comparisons via CompareSize function
**
**  This function compares the size of the matrix against the provided
**  sizes (int, int).  If the sizes are equal the return value is zero.
**
**  The following table should be consulted to deteremine the return
**  value.  == means equal, < and > mean smaller and larger respectively.
**  All comparison are to the source matrix provided.
**
**       1     :     2     : Return
**    ++++++++++++++++++++++++++++++++++++++
**       ==    :     ==    :  0
**       ==    :     <     :  -1
**       ==    :     >     :  +1
**       <     :     ==    :  -2
**       <     :     <     :  -3
**       <     :     >     :  +4
**       >     :     ==    :  +2
**       >     :     >     :  +3
**       >     :     <     :  +5
**
**  The third variable provided is an error variable.  An error of -1
**  is placed here on failure.  Otherwise this value is 0.  If there is an
**  error the return value is undefined.
*****************************************************************************/

/*****************************************************************************
**  Note 3:  Get* Functions
**
**  The get functions assume that you have already correctly allocated
**  the necessary amount of memory and only the exact amount of memory.
**  Allocating the correct amount of memory (i.e. not too much) is
**  VERY important for the GetArea function (void * version) as the matrix
**  provided must be parsed in vector format.
*****************************************************************************/

/*****************************************************************************
**  Note 4:  PopulateArea (sMatrix version)
**
**  The sMatrix must be initalize before hand.  The checking done is
**  simply a call to GetSize.  If GetSize returns successfully then
**  it is assumed that everything is fine and can continue.
**
**  All population is inclusive, that is if:
**     Row = 2, Column = 2 and the size of the sMatrix provided as an
**     argument 3,3
**     Elements:      (2,2), (3,2), (4,2)
**                    (2,3), (3,3), (4,3)
**                    (2,4), (3,4), (4,4)
**
**  all get set making 3 rows and columns that were set
*****************************************************************************/

/*****************************************************************************
**  Note 5:  GetArea (sMatrix version)
**
**  The sMatrix must be newly created (that is, not initialized) but it
**  must exist (i.e. allocated).  No allocation is performed and a check
**  is done against NULL in an attempt to avoid a bad pointer, but
**  one should exercise caution when using this function.
**
**  In addition, the types of data stored in the two matricies MUST match.
*****************************************************************************/
```

```
/*****************************************************************************
**  Note 6:  Coll (sMatrix & R)
**
**  This is the collesky (sp) decomposition routine.  R must be provided
**  in an unitialized form.
**
**  This is all original code, so don't blame me if it doesn't work ;-)
**
**  The matrix on which the Collesky is performed must be square (nxn).
**  This is checked in the code and some other restrictions are also
**  checked.
*****************************************************************************/
```

Matrix.Cpp

```
/*****************************************************************
**  Matrix.Cpp
**      Implements sMatrix Class
**      April 21-1998
*****************************************************************/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <memory.h>
#include "matrix.h"

/*Things that are required from NRinC */

double at, bt, ct;
#define PYTHAG(a,b) ((at=fabs(a)) > (bt=fabs(b)) ? \
   (ct=bt/at,at*sqrt(1.0+ct*ct)) : (bt ? (ct=at/bt,bt*sqrt(1.0+ct*ct)) : 0.0))

#include "StdAfx.h"
#include "matrix.h"
double maxarg1, maxarg2;
#define MAX(a,b) (maxarg1=(a), maxarg2=(b), (maxarg1) > (maxarg2) ?\
   (maxarg1) : (maxarg2))
#define SIGN(a,b) ((b) >= 0.0 ? fabs(a) : -fabs(a))


/*****************************************************************
**  Constructors and Destructors
*****************************************************************/

sMatrix::sMatrix(){

   m_x = m_y = 0;
   m_datatype = -1;
   m_data = NULL;
   m_datatypesize = 0;
   }

sMatrix::sMatrix(int Datatype){

   m_x = m_y = 0;
   m_data = NULL;
   if (Datatype >=0 && Datatype < mMAX){
      m_datatype = Datatype;

      switch (m_datatype){

         case mCHAR:
            m_datatypesize = sizeof(char);
            break;

         case mINT:
         case mLONG:
         case mFLOAT:
            m_datatypesize = sizeof(long);
            break;

         case mDOUBLE:
            m_datatypesize = sizeof(double);
            break;
         } //end switch
      }//endif datatype >0 && < max

   else {
      m_datatype = -1;
      m_datatypesize = 0;
      }
```

```
    }

sMatrix::sMatrix(int Datatype, int Rows, int Columns){

    m_data = NULL;
    if (Datatype >=0 && Datatype <mMAX){

        m_datatype = Datatype;
        if (Rows >0 && Columns >0){     //ensure a valid number!

            m_x = Rows;
            m_y = Columns;

            switch (m_datatype){
                case mCHAR:
                    m_data = (void *)new char[Rows * Columns];
                                                            m_datatypesize = sizeof(char);
                    break;
                case mINT:
                    m_data = (void *)new int[Rows * Columns];
                    m_datatypesize = sizeof(int);
                    break;
                case mLONG:
                    m_data = (void *)new long[Rows * Columns];
                    m_datatypesize = sizeof(long);
                    break;
                case mFLOAT:
                    m_data = (void *)new float[Rows * Columns];
                    m_datatypesize = sizeof(float);
                    break;
                case mDOUBLE:
                    m_data = (void *)new double[Rows * Columns];
                    m_datatypesize = sizeof(double);
                    break;
                }  //end switch
                                        if(m_data==NULL)//These two lines were not here
                                                exit(-1);
            }// end if rows && columns
        }//end if datatype

    else{  //Datatype not valid
        m_datatype = -1;
        m_datatypesize = 0;
        m_x = 0;
        m_y = 0;

        }

    }
//The following constructor is used to convert from vector type to matrix type
sMatrix::sMatrix(double *pDataList, int iRows, int iColumns)
{
                int i;
                double *pTempDouble1, *pTempDouble2;
                m_datatype=mDOUBLE;//set data type
                m_x=0;//initialize number of rows
                m_y=0;//initialize number of columns
                m_data=NULL;//initialize data pointer
                if((pDataList!=NULL)&&(iRows>0)&&(iColumns>0))
                {
                                m_x=iRows;
                                m_y=iColumns;
                                m_data=new double[iRows*iColumns];//allocate memory for matrix
                                if(m_data==NULL)//did memory allocation failed
                                        exit(-1);//exit
                                //the following loops copies the contents of the receieved vector into the
                                //matrix
                                pTempDouble1=(double *) m_data;
                                pTempDouble2=pDataList;
                                for(i=1;i<=(iRows*iColumns);i++)
```

```
                        {
                                (*pTempDouble1)=(*pTempDouble2);
                                pTempDouble1++;
                                pTempDouble2++;
                        }
                        m_datatypesize = sizeof(double);;
                }
}


sMatrix::~sMatrix(){

   if ((m_data != NULL)){

      delete[] m_data;//This line was commented out
      m_data = NULL;

      }

   m_datatype = 0;
   m_datatypesize = 0;
   m_x = 0;
   m_y = 0;

   }

/*****************************************************************
**  Functions for filling elements
*****************************************************************/

/*****************************************************************
**  Populate allows you to set a single element in an array.
**      For the real guts of this code you need to see
**      HSetElement.  It contains excellent examples of pointer
**      manipulation and casting.
*****************************************************************/

int sMatrix::Populate(void * data, int Row, int Column){

   /*  First verify everything is inplace to work on the matrix */

   if (data == NULL) return (-1);                //invalid data ptr
   if (Row <=0 || Row >m_x) return (-2);      //invalid row arg
   if (Column <=0 || Column >m_y) return (-3); //invalid column arg
   if (m_x == 0 || m_y ==0) return (-4);   //size not set and not alloced

   /*  Everything is fine for use of the matrix, we may proceed  */

   if (HSetElement(data, Row, Column) < 0) return (-5);

   return (0);
   }

/*****************************************************************
**  PopulateRow allows you to set a row in an array to a single
**      value.
*****************************************************************/

int sMatrix::PopulateRow(void * data, int Row){

   int x;

   /*  First verify everything is inplace to work on the matrix */

   if (data == NULL) return (-1);                //invalid data ptr
   if (Row <=0 || Row >m_x) return (-2);      //invalid row arg

   /*  Everything is cool, let's set the row */

   for (x=1 ; x<=m_y ; x++){
```

```
      if (HSetElement(data, Row, x) <0) return (-5);
      }

   return (0);
   }

/*****************************************************************
**  PopulateColumn allows you to set a row in an array to a
**      single value.
*****************************************************************/

int sMatrix::PopulateColumn(void * data, int Column){

   int y;

   /*  First verify everything is inplace to work on the matrix */

   if (data == NULL) return (-1);                  //invalid data ptr
   if (Column <=0 || Column >m_y) return (-3); //invalid column arg

   /*  Everything is cool, let's set the row */

   for (y=1 ; y<=m_x ; y++){
      if (HSetElement(data, y, Column) <0) return (-5);
      }

   return (0);
   }

/*****************************************************************
**  PopulateArea allows you to set an area in an array to a
**      single value.
*****************************************************************/

int sMatrix::PopulateArea(void * data, int Row, int Column, int Rows,
                     int Columns){

   int x, y;

   /*  First verify everything is inplace to work on the matrix */

   if (data == NULL) return (-1);              //invalid data ptr
   if (Row <=0 || Row >m_x) return (-2);       //invalid row arg
   if (Column <=0 || Column >m_y) return (-3); //invalid column arg
   if (Rows <=0) return (-6);
   if (Columns <=0) return (-7);

   /*  Everything is cool, let's set the row */
   for (x=Row ; x<=m_x && x<=Row+(Rows-1) ; x++){
      for (y=Column ; y<=m_y && y<=Column+(Columns-1) ; y++){
         if (HSetElement(data, x, y) <0) return (-5);
         }
      }

   return (0);
   }

/*****************************************************************
**  PopulateArea allows you to set an area in an array to be
**      set from another sMatrix (see Note 4 in the header).
*****************************************************************/

int sMatrix::PopulateArea(sMatrix * source, int Row, int Column){

   int source_x, source_y, source_type;
   int x,y;
   void * tempdata;

   if (source == NULL) return (-1);           //invalid data ptr
   if (Row <=0 || Row >m_x) return (-2);      //invalid row arg
```

```
   if (Column <=0 || Column >m_y) return (-3); //invalid column arg
   if (source->GetSize(&source_x, &source_y) < 0)
      return (-10);   //source is not ready

   // now check that the data types of the two matricies match

   if (source->GetType(&source_type) < 0)
      return (-10);

   if (source_type != m_datatype)
      return (-11);   //type mismatch

   /*  All the necessary checks have been performed, let's move on */

   tempdata = malloc(m_datatypesize);  //allocate the temporary storage

   for (x=Row ; x<=m_x ; x++){
      for (y=Column ; y<=m_y ; y++){

         /* Get the source element */
         if (source->GetElement(tempdata, (x+1)-Row, (y+1)-Column) < 0){
            free (tempdata);
            return (-5);
            }

         /* If there are no errors, write it into this matrix */
         if (HSetElement(tempdata, x, y) < 0){
            free (tempdata);
            return (-5);
            }

         }
      }

   free(tempdata);

   return (0);
   }


/*****************************************************************
** Functions for retrieving specific elements
*****************************************************************/

/*****************************************************************
**  GetElement
**      This function is identical to populate, except it
**      returns the value contained in the particular element
*****************************************************************/

int sMatrix::GetElement(void * data, int Row, int Column) {

   if (data == NULL) return (-1);              //invalid data ptr
   if (Row <=0 || Row >m_x) return (-2);       //invalid row arg
   if (Column <=0 || Column >m_y) return (-3); //invalid column arg

   if (HGetElement(data, Row, Column) < 0) return (-5);

   return (0);

   }

/*****************************************************************
**  GetRow
**      This function is identical to populaterow, except it
**      returns the value contained in the particular row
*****************************************************************/

int sMatrix::GetRow(void * data, int Row){
```

```
   unsigned int tempptr = 0;
   int x;
   void * tempdata = malloc (m_datatypesize);

   if (!tempdata) return (-8);     //internal error

   if (data == NULL){
      free(tempdata);
      return (-1);               //invalid data ptr
      }

   if (Row <=0 || Row >m_x){
      free(tempdata);
      return (-2);               //invalid row arg
      }

   tempptr = (unsigned int)data;

   for ( x=1 ; x<=m_y ; x++ ){

      if (HGetElement(tempdata, Row, x) < 0){
         free(tempdata);
         return (-5);
         }

      if (!memcpy ((void *)tempptr, tempdata, m_datatypesize)){
         free(tempdata);
         return (-8);  //There is an internal error return failure
         }

      tempptr += m_datatypesize;  //incerment the pointer

      }

   free(tempdata);

   return (0);

   }

/*****************************************************************
**  GetColumn
**      This function is identical to populatecolumn, except it
**      returns the value contained in the particular column
*****************************************************************/

int sMatrix::GetColumn(void * data, int Column){

   unsigned int tempptr = 0;
   int y;
   void * tempdata = malloc (m_datatypesize);

   if (!tempdata) return (-8);     //internal error

   if (data == NULL){
      free(tempdata);
      return (-1);               //invalid data ptr
      }

   if (Column <=0 || Column >m_y){
      free(tempdata);
      return (-3);               //invalid Column arg
      }

   tempptr = (unsigned int)data;

   for ( y=1 ; y<=m_x ; y++ ){

      if (HGetElement(tempdata, y, Column) < 0){
         free(tempdata);
```

```
            return (-5);
            }

        if (!memcpy ((void *)tempptr, tempdata, m_datatypesize)){
            free(tempdata);
            return (-8);  //There is an internal error return failure
            }

        tempptr += m_datatypesize;  //incerment the pointer

        }

    free(tempdata);

    return (0);

    }

int sMatrix::GetColumn(sMatrix *dest,int Column)
{
    int y;
    double tempDouble;
    if (dest == NULL)
                        {
        return (-1);            //invalid data ptr
        }
            if(((dest->m_x)!=m_x)||((dest->m_y)!=1))
            {
                    return -1;
            }
    if (Column <=0 || Column >m_y)
                        {
        return (-1);            //invalid Column arg
        }
    for ( y=1 ; y<=m_x ; y++ ){

        if (HGetElement(&tempDouble, y, Column) < 0)
                                    {
            return (-1);
            }

        if ((dest->HSetElement(&tempDouble, y,1))<0)
                                    {
             return (-1);
            }

         }

    return (0);
}

/*****************************************************************
**  GetArea (void * version)
**      This function is identical to populatearea, except it
**      returns the value contained in the particular area
*****************************************************************/

int sMatrix::GetArea(void * data, int Row, int Column, int Rows,
                int Columns){

    unsigned int tempptr = 0;
    int x,y;
    void * tempdata;

    /*  First verify everything is inplace to work on the matrix */

    if (data == NULL) return (-1);              //invalid data ptr
    if (Row <=0 || Row >m_x) return (-2);       //invalid row arg
    if (Column <=0 || Column >m_y) return (-3); //invalid column arg
    if (Rows <=0) return (-6);
```

```
   if (Columns <=0) return (-7);

   tempdata = malloc (m_datatypesize);

   if (!tempdata) return (-8);     //internal error not allocated

   tempptr = (unsigned int)data;   //assign the pointer, or more accurately
                                   //the value of the pointer

   for (x=Row ; x<=m_x && x<=Row+(Rows-1) ; x++){
      for (y=Column ; y<=m_y && y<=Column+(Columns-1) ; y++){

         if (HGetElement(tempdata, x, y) <0){
            free(tempdata);
            return (-5);
            }

         if (!memcpy ((void *)tempptr, tempdata, m_datatypesize)){
            free(tempdata);
            return (-8);  //There is an internal error return failure
            }

         tempptr += m_datatypesize;  //incerment the pointer

         }
      }

   return (0);

   }

/*****************************************************************
**  GetArea (sMatrix * version)
**      This function is identical to populatearea, except it
**      returns the value contained in the particular area
*****************************************************************/

int sMatrix::GetArea(sMatrix * dest, int Row, int Column){

   int dest_x, dest_y, dest_type;
   int x,y;
   void * tempdata;

   if (dest == NULL) return (-1);            //invalid data ptr
   if (Row <=0 || Row >m_x) return (-2);       //invalid row arg
   if (Column <=0 || Column >m_y) return (-3); //invalid column arg
   if (dest->GetSize(&dest_x, &dest_y) < 0)
      return (-10);   //dest is not ready

   // now check that the data types of the two matricies match

   if (dest->GetType(&dest_type) < 0)
      return (-10);

   if (dest_type != m_datatype)
      return (-11);   //type mismatch

   /*  All the necessary checks have been performed, let's move on */

   tempdata = malloc(m_datatypesize);  //allocate the temporary storage

   for (x=Row ; x<=m_x && x<=dest_x; x++){
      for (y=Column ; y<=m_y && y<=dest_y ; y++){

         /* Get the source element */
         if (HGetElement(tempdata, x, y) < 0){
            free (tempdata);
            return (-5);
            }
```

```
            /* If there are no errors, write it into this matrix */
            if (dest->Populate(tempdata, (x+1)-Row, (y+1)-Column) < 0){
               free (tempdata);
               return (-5);
               }
            }
         }

   free (tempdata);

   return (0);
   }

/*****************************************************************
** Functions for relating to the data type
*****************************************************************/

/*****************************************************************
**  SetType
**      This function allows you to set the type of data to be
**      stored in the class, in case you did not do it at the
**      time of construction.  If the type is already set an
**      error will be returned and nothing in the class will
**      be changed.
*****************************************************************/

int sMatrix::SetType(int Datatype){

   if (m_datatype >=0) return (-12); //type already set

   if (Datatype >=0 && Datatype < mMAX){
      m_datatype = Datatype;

      switch (m_datatype){

         case mCHAR:
            m_datatypesize = sizeof(char);
            break;

         case mINT:              //all of these are the same length
         case mLONG:
         case mFLOAT:
            m_datatypesize = sizeof(long);
            break;

         case mDOUBLE:
            m_datatypesize = sizeof(double);
            break;
         } //end switch
      }//endif datatype >0 && < max

   else {
      m_datatype = -1;
      m_datatypesize = 0;
      return (-13);       //invalid type
      }

   return(0);
   }

/*****************************************************************
**  GetType
**      This function allows you to get the type of data stored
**      in the class, in case you do not know it.  If the type
**      has not been set an error will be returned.
*****************************************************************/

int sMatrix::GetType(int * Datatype){

   if (m_datatype < 0) return (-14);
```

```
   *Datatype = m_datatype;

   return (0);

   }

/*****************************************************************
**  Functions relating to size of the matrix
*****************************************************************/

/*****************************************************************
**  SetSize(int, int)
**      SetSize allows you to set the size of a matrix, if
**      you did not specify a size at the time of construction/
**      creation.  It will return a -15 if the size of the
**      matrix has already been set.  If the type has not been
**      set a -16 will be return.
**      THE TYPE MUST BE SET BEFORE CALLING SET SIZE.
*****************************************************************/

int sMatrix::SetSize(int Rows, int Columns){

   if (Rows <=0) return (-2);      //invalid row argument
   if (Columns <=0) return (-3);   //inavlid column argument

   if (m_x > 0) return (-15);      //matrix is already set
   if (m_datatype < 0) return (-16);   //type is not set.  Must be set first

   switch (m_datatype){
      case mCHAR:
         m_data = (void *)new char[Rows * Columns];
         if (!m_data) return (-8);   //error allocation failed
         break;
      case mINT:
         m_data = (void *)new int[Rows * Columns];
         if (!m_data) return (-8);   //error allocation failed
         break;
      case mLONG:
         m_data = (void *)new long[Rows * Columns];
         if (!m_data) return (-8);   //error allocation failed
         break;
      case mFLOAT:
         m_data = (void *)new float[Rows * Columns];
         if (!m_data) return (-8);   //error allocation failed
         break;
      case mDOUBLE:
         m_data = (void *)new double[Rows * Columns];
         if (!m_data) return (-8);   //error allocation failed
         break;
      }  //end switch

            m_x = Rows;
            m_y = Columns;

   return (0);

   }


/*****************************************************************
**  GetSize (int *, int*)
**      GetSize enable you to query a sMatrix class about it's
**      size.  The arguments are Rows and Columns respectively.
**      If the size is not yet set, a -17 will be returned
*****************************************************************/

int sMatrix::GetSize(int * Rows, int * Columns){

   if (m_x <= 0 || m_y <= 0) return (-17);     //error size not set
```

```
   if (!Rows) return (-2);     //rows is null, invalid
   if (!Columns) return (-3);  //columns is null, invalid

   *Rows = m_x;
   *Columns = m_y;

   return (0);

   }

/*****************************************************************
**  CompareSize (int , int, int*)
**      This function allows you to compare the size of 2
**      matricies.  See Note 2 for more information of the
**      operation of this function
*****************************************************************/

int sMatrix::CompareSize(int Rows, int Columns, int * ErrorVal){

   int retval;

   if (m_x <=0 || m_y <=0){
      *ErrorVal = -1;
      return (0);
      }

   *ErrorVal = 0;  //Set this incase the user forget to set it before
                   //calling this function.  We don't want them to think
                   //there was an error when indeed there isn't.

   if (m_x == Rows) {
      if (m_y == Columns) retval = 0;
      if (m_y <  Columns) retval = -1;
      if (m_y >  Columns) retval = 1;
      }

   else if (m_x < Rows){
      if (m_y == Columns) retval = -2;
      if (m_y <  Columns) retval = -3;
      if (m_y >  Columns) retval = 4;
      }

   else if (m_x > Rows){
      if (m_y == Columns) retval = 2;
      if (m_y <  Columns) retval = 5;
      if (m_y >  Columns) retval = 3;
      }

   return (retval);
   }

/*****************************************************************
**  CompareSize (sMatrix *, int*)
**      This function allows you to compare the size of 2
**      matricies.  See Note 2 for more information of the
**      operation of this function.  A errorval of -20 means
**      the source matrix was not available
*****************************************************************/

int sMatrix::CompareSize(sMatrix * source, int * ErrorVal){

   int Rows, Columns, retval;

   if (!source){
      *ErrorVal = -1;
      return (0);
      }

   if (m_x <=0 || m_y <=0){
      *ErrorVal = -1;
```

```
    return (0);
    }

  if (source->GetSize(&Rows, &Columns) < 0){
    *ErrorVal = -20;
    return (0);
    }

  *ErrorVal = 0;  //Set this incase the user forget to set it before
                  //calling this function.  We don't want them to think
                  //there was an error when indeed there isn't.

  if (m_x == Rows) {
    if (m_y == Columns) retval = 0;
    if (m_y <  Columns) retval = -1;
    if (m_y >  Columns) retval = 1;
    }

  else if (m_x < Rows){
    if (m_y == Columns) retval = -2;
    if (m_y <  Columns) retval = -3;
    if (m_y >  Columns) retval = 4;
    }

  else if (m_x > Rows){
    if (m_y == Columns) retval = 2;
    if (m_y <  Columns) retval = 5;
    if (m_y >  Columns) retval = 3;
    }

  return (retval);
  }

/*****************************************************************
**  Math Operations
*****************************************************************/

/*****************************************************************
** overloaded operators for simple math
*****************************************************************/

/*****************************************************************
**  operator +
**      This operator provides the ability to add to equally
**      sized matricies with no extra code.
*****************************************************************/

sMatrix& sMatrix::operator+(sMatrix & source) {

  sMatrix *result=new sMatrix;
  int x, y;
            int type;

            void * d1, * d2;

  if (source.GetSize(&x, &y)<0) return (*this);
  if (x != m_x || y != m_y) return (*this);

            source.GetType(&type);
            result->SetType(type);
            result->SetSize(x, y);

  d1 = malloc(m_datatypesize);/*******Are Not Freed******************/
  d2 = malloc(m_datatypesize);

  for (x=1;x<=m_x;x++)
    for (y=1;y<=m_y;y++){

        HGetElement(d1, x, y);
        source.GetElement(d2, x, y);
```

```
switch (m_datatype){
   case mCHAR:
      char *cd1, *cd2, cd3;
      cd1 = (char *)d1;
      cd2 = (char *)d2;
                                                cd3 = *cd1 + *cd2;
                                                result->Populate((void *)&cd3, x, y);
                                                break;
   case mINT:
      int *id1, *id2, id3;
      id1 = (int  *)d1;
      id2 = (int  *)d2;
                              id3 = *id1 + *id2;
                              result->Populate((void *)&id3, x, y);
                              break;
 case mLONG:
                                                long *ld1, *ld2, ld3;
      ld1 = (long *)d1;
      ld2 = (long *)d2;
                              ld3 = *ld1 + *ld2;
                              result->Populate((void *)&ld3, x, y);
                              break;
   case mFLOAT:
      float *fd1, *fd2, fd3;
      fd1 = (float*)d1;
      fd2 = (float*)d2;
                              fd3 = *fd1 + *fd2;
                              result->Populate((void *)&fd3, x, y);
                              break;
   case mDOUBLE:
      double *dd1, *dd2, dd3;
      dd1 = (double*)d1;
      dd2 = (double*)d2;
                              dd3 = *dd1 + *dd2;
                              result->Populate((void *)&dd3, x, y);
      break;
   }//end switch

                  /*switch (m_datatype){
   case mCHAR:
      char *cd1, *cd2, cd3;
      cd1 = (char *)d1;
      cd2 = (char *)d2;
                              break;
   case mINT:
      int *id1, *id2, id3;
      id1 = (int  *)d1;
      id2 = (int  *)d2;
                              break;
   case mLONG:
                              long *ld1, *ld2, ld3;
      ld1 = (long *)d1;
      ld2 = (long *)d2;
                              break;
   case mFLOAT:
      float *fd1, *fd2, fd3;
      fd1 = (float*)d1;
      fd2 = (float*)d2;
                              break;
   case mDOUBLE:
      double *dd1, *dd2, dd3;
      dd1 = (double*)d1;
      dd2 = (double*)d2;
      break;
   }//end switch*/

}//end for y & x

     free(d1);
```

```
            free(d2);
   return (*result);

   }
/*****************************************************************
**  operator +=
**      This operator provides the ability to add to equally
**      sized matricies with no extra code.
*****************************************************************/

void sMatrix::operator+=(sMatrix & source) {

   int x, y;
            void * d1, * d2;

   if (source.GetSize(&x, &y)<0) return;
   if (x != m_x || y != m_y) return;

   d1 = malloc(m_datatypesize);
   d2 = malloc(m_datatypesize);

   for (x=1;x<=m_x;x++)
      for (y=1;y<=m_y;y++)
                        {

         HGetElement(d1, x, y);
         source.GetElement(d2, x, y);

         switch (m_datatype){
            case mCHAR:
               char *cd1, *cd2, cd3;
               cd1 = (char *)d1;
               cd2 = (char *)d2;
                                    cd3 = *cd1 + *cd2;
                                    Populate((void *)&cd3, x, y);
                                    break;
            case mINT:
               int *id1, *id2, id3;
               id1 = (int  *)d1;
               id2 = (int  *)d2;
                                    id3 = *id1 + *id2;
                                    Populate((void *)&id3, x, y);
                                    break;
            case mLONG:
                                    long *ld1, *ld2, ld3;
               ld1 = (long *)d1;
               ld2 = (long *)d2;
                                    ld3 = *ld1 + *ld2;
                                    Populate((void *)&ld3, x, y);
                                    break;
            case mFLOAT:
               float *fd1, *fd2, fd3;
               fd1 = (float*)d1;
               fd2 = (float*)d2;
                                    fd3 = *fd1 + *fd2;
                                    Populate((void *)&fd3, x, y);
                                    break;
            case mDOUBLE:
               double *dd1, *dd2, dd3;
               dd1 = (double*)d1;
               dd2 = (double*)d2;
                                    dd3 = *dd1 + *dd2;
                                    Populate((void *)&dd3, x, y);
               break;
            }//end switch
                        }

            free(d1);
            free(d2);
   return ;
```

```
  }

/*****************************************************************
**  operator -
**      This operator provides the ability to subtract to equally
**      sized matricies with no extra code.
*****************************************************************/

sMatrix& sMatrix::operator-(sMatrix & source) {

   sMatrix *result=new sMatrix;
   int x, y;
            int type;

   void * d1, * d2;

   if (source.GetSize(&x, &y)<0) return (*this);
   if (x != m_x || y != m_y) return (*this);

             source.GetType(&type);
             result->SetType(type);
             result->SetSize(x, y);

   d1 = malloc(m_datatypesize);/*Are Not freed*/
   d2 = malloc(m_datatypesize);

   for (x=1;x<=m_x;x++)
      for (y=1;y<=m_y;y++){

         HGetElement(d1, x, y);
         source.GetElement(d2, x, y);

         switch (m_datatype){
            case mCHAR:
               char *cd1, *cd2, cd3;
               cd1 = (char *)d1;
               cd2 = (char *)d2;
                                    cd3 = *cd1 - *cd2;
                                    result->Populate((void *)&cd3, x, y);
                                    break;
            case mINT:
               int *id1, *id2, id3;
               id1 = (int  *)d1;
               id2 = (int  *)d2;
                                    id3 = *id1 - *id2;
                                    result->Populate((void *)&id3, x, y);
                                    break;
            case mLONG:
                                    long *ld1, *ld2, ld3;
               ld1 = (long *)d1;
               ld2 = (long *)d2;
                                    ld3 = *ld1 - *ld2;
                                    result->Populate((void *)&ld3, x, y);
                                    break;
            case mFLOAT:
               float *fd1, *fd2, fd3;
               fd1 = (float*)d1;
               fd2 = (float*)d2;
                                    fd3 = *fd1 - *fd2;
                                    result->Populate((void *)&fd3, x, y);
                                    break;
            case mDOUBLE:
               double *dd1, *dd2, dd3;
               dd1 = (double*)d1;
               dd2 = (double*)d2;
                                    dd3 = *dd1 - *dd2;
                                    result->Populate((void *)&dd3, x, y);
               break;
            }//end switch
```

```
        }//end for y & x

            free(d1);
            free(d2);
   return (*result);

   }

/*****************************************************************
**  operator *
**      This operator provides the ability to multiply
**      appropriately sized matricies with no extra code.
*****************************************************************/

sMatrix& sMatrix::operator*(sMatrix & source) {

   sMatrix *result=new sMatrix;
   int x, y;
            int type;

            int i, j, k;

   void * d1, * d2;

   if (source.GetSize(&x, &y)<0) return (*this);
   if (m_y != x) return (*this);

            source.GetType(&type);
            result->SetType(type);
            result->SetSize(m_x, y);

   d1 = malloc(m_datatypesize);//Are not freed
   d2 = malloc(m_datatypesize);

   for (i=1;i<=m_x;i++)
                        for (j=1;j<=y;j++){

        switch (m_datatype){
           case mCHAR:
                        char csum;
                        csum = 0;
                        char *cd1, *cd2;
                        for (k=1;k<=m_y;k++){
                        HGetElement(d1, i, k);
                        source.GetElement(d2, k, j);
                        cd1 = (char *)d1;
                        cd2 = (char *)d2;
                        csum += (*cd1 * *cd2);
                        }
                        result->Populate((void *)&csum, i, j);
                        break;
           case mINT:
              int isum;
                        isum = 0;
                        int *id1, *id2;
                        for (k=1;k<=m_y;k++){
                        HGetElement(d1, i, k);
                        source.GetElement(d2, k, j);
                        id1 = (int *)d1;
                        id2 = (int *)d2;
                        isum += (*id1 * *id2);
                        }
                        result->Populate((void *)&isum, i, j);
                        break;
           case mLONG:
                        long lsum;
                        lsum = 0;
                        long *ld1, *ld2;
                        for (k=1;k<=m_y;k++){
```

```
                        HGetElement(d1, i, k);
                        source.GetElement(d2, k, j);
                        ld1 = (long *)d1;
                        ld2 = (long *)d2;
                        lsum += (*ld1 * *ld2);
                        }
                        result->Populate((void *)&lsum, i, j);
                        break;
            case mFLOAT:
               float fsum;
                        fsum = 0;
                        float *fd1, *fd2;
                        for (k=1;k<=m_y;k++){
                        HGetElement(d1, i, k);
                        source.GetElement(d2, k, j);
                        fd1 = (float *)d1;
                        fd2 = (float *)d2;
                        fsum += (*fd1 * *fd2);
                        }
                        result->Populate((void *)&fsum, i, j);
                        break;
            case mDOUBLE:
               double dsum;
                        dsum = 0;
                        double *dd1, *dd2;
                        for (k=1;k<=m_y;k++){
                        HGetElement(d1, i, k);
                        source.GetElement(d2, k, j);
                        dd1 = (double *)d1;
                        dd2 = (double *)d2;
                        dsum += (*dd1 * *dd2);
                        }
                        result->Populate((void *)&dsum, i, j);
               break;
            }//end switch

                                    }//end for y & x
             free(d1);
             free(d2);
   return (*result);

   }

sMatrix& sMatrix::operator* (double& xx){
   if (m_y == 0) return (*this);

   sMatrix *result=new sMatrix;
   void * d1;
   int x = 0, y = 0, type = 0;

             type = m_datatype;
             result->SetType(type);
             result->SetSize(m_x, m_y);

   d1 = malloc(m_datatypesize);/******Was Not Freed****************/

   for (x=1;x<=m_x;x++){
      for(y=1;y<=m_y;y++){

         HGetElement(d1, x, y);
         switch (m_datatype){
            case mCHAR:
               char *cd1, cdr;
               cd1 = (char *)d1;
               cdr = (char)*cd1 * (char)xx;
               result->Populate((void *)&cdr, x, y);
               break;
            case mINT:
               int *id1, idr;
               id1 = (int  *)d1;
```

```
            idr = (int)*id1 * (int)xx;
            result->Populate((void *)&idr, x, y);
                                                    break;
         case mLONG:
                                                    long *ld1, ldr;
            ld1 = (long *)d1;
            ldr = (long)*ld1 * (long)xx;
            result->Populate((void *)&ldr, x, y);
                                                    break;
         case mFLOAT:
            float *fd1, fdr;
            fd1 = (float*)d1;
            fdr = (float)*fd1 * (float)xx;
            result->Populate((void *)&fdr, x, y);
                                                    break;
         case mDOUBLE:
            double *dd1, ddr;
            dd1 = (double*)d1;
            ddr = (double)*dd1 * (double)xx;
            result->Populate((void *)&ddr, x, y);
            break;
          }//end switch
       }
    }
          free(d1);
  return(*result);
  }
sMatrix& sMatrix::operator* (float& xx){
  if (m_y == 0) return (*this);

  sMatrix *result=new sMatrix;
  void * d1;
  int x = 0, y = 0, type = 0;

           type = m_datatype;
           result->SetType(type);
           result->SetSize(m_x, m_y);

  d1 = malloc(m_datatypesize);

  for (x=1;x<=m_x;x++){
    for(y=1;y<=m_y;y++){

      HGetElement(d1, x, y);
      switch (m_datatype){
         case mCHAR:
            char *cd1, cdr;
            cd1 = (char *)d1;
            cdr = (char)*cd1 * (char)xx;
            result->Populate((void *)&cdr, x, y);
            break;
         case mINT:
            int *id1, idr;
            id1 = (int  *)d1;
            idr = (int)*id1 * (int)xx;
            result->Populate((void *)&idr, x, y);
                                                    break;
         case mLONG:
                                                    long *ld1, ldr;
            ld1 = (long *)d1;
            ldr = (long)*ld1 * (long)xx;
            result->Populate((void *)&ldr, x, y);
                                                    break;
         case mFLOAT:
            float *fd1, fdr;
            fd1 = (float*)d1;
            fdr = (float)*fd1 * (float)xx;
            result->Populate((void *)&fdr, x, y);
                                                    break;
         case mDOUBLE:
```

```
                double *dd1, ddr;
                dd1 = (double*)d1;
                ddr = (double)*dd1 * (double)xx;
                result->Populate((void *)&ddr, x, y);
                break;
              }//end switch
            }
          }

    return(*result);
    }
sMatrix& sMatrix::operator* (int& xx){
    if (m_y == 0) return (*this);

    sMatrix *result=new sMatrix;
    void * d1;
    int x = 0, y = 0, type = 0;

                type = m_datatype;
                result->SetType(type);
                result->SetSize(m_x, m_y);

    d1 = malloc(m_datatypesize);

    for (x=1;x<=m_x;x++){
       for(y=1;y<=m_y;y++){

          HGetElement(d1, x, y);
          switch (m_datatype){
             case mCHAR:
                char *cd1, cdr;
                cd1 = (char *)d1;
                cdr = (char)*cd1 * (char)xx;
                result->Populate((void *)&cdr, x, y);
                break;
             case mINT:
                int *id1, idr;
                id1 = (int  *)d1;
                idr = (int)*id1 * (int)xx;
                result->Populate((void *)&idr, x, y);
                                                        break;
             case mLONG:
                                                        long *ld1, ldr;
                ld1 = (long *)d1;
                ldr = (long)*ld1 * (long)xx;
                result->Populate((void *)&ldr, x, y);
                                                        break;
             case mFLOAT:
                float *fd1, fdr;
                fd1 = (float*)d1;
                fdr = (float)*fd1 * (float)xx;
                result->Populate((void *)&fdr, x, y);
                                                        break;
             case mDOUBLE:
                double *dd1, ddr;
                dd1 = (double*)d1;
                ddr = (double)*dd1 * (double)xx;
                result->Populate((void *)&ddr, x, y);
                break;
             }//end switch
          }
       }

    return(*result);
    }
sMatrix& sMatrix::operator* (long& xx){
    if (m_y == 0) return (*this);

    sMatrix *result=new sMatrix;
    void * d1;
```

```
   int x = 0, y = 0, type = 0;

               type = m_datatype;
               result->SetType(type);
               result->SetSize(m_x, m_y);

   d1 = malloc(m_datatypesize);

   for (x=1;x<=m_x;x++){
      for(y=1;y<=m_y;y++){

         HGetElement(d1, x, y);
         switch (m_datatype){
            case mCHAR:
               char *cd1, cdr;
               cd1 = (char *)d1;
               cdr = (char)*cd1 * (char)xx;
               result->Populate((void *)&cdr, x, y);
               break;
            case mINT:
               int *id1, idr;
               id1 = (int  *)d1;
               idr = (int)*id1 * (int)xx;
               result->Populate((void *)&idr, x, y);
                                                        break;
            case mLONG:
                                                        long *ld1, ldr;
               ld1 = (long *)d1;
               ldr = (long)*ld1 * (long)xx;
               result->Populate((void *)&ldr, x, y);
                                                        break;
            case mFLOAT:
               float *fd1, fdr;
               fd1 = (float*)d1;
               fdr = (float)*fd1 * (float)xx;
               result->Populate((void *)&fdr, x, y);
                                                        break;
            case mDOUBLE:
               double *dd1, ddr;
               dd1 = (double*)d1;
               ddr = (double)*dd1 * (double)xx;
               result->Populate((void *)&ddr, x, y);
               break;
            }//end switch
         }
      }

   return(*result);
   }
sMatrix& sMatrix::operator* (char& xx){
   if (m_y == 0) return (*this);

   sMatrix *result=new sMatrix;
   void * d1;
   int x = 0, y = 0, type = 0;

               type = m_datatype;
               result->SetType(type);
               result->SetSize(m_x, m_y);

   d1 = malloc(m_datatypesize);

   for (x=1;x<=m_x;x++){
      for(y=1;y<=m_y;y++){

         HGetElement(d1, x, y);
         switch (m_datatype){
            case mCHAR:
               char *cd1, cdr;
               cd1 = (char *)d1;
```

```
            cdr = (char)*cd1 * (char)xx;
            result->Populate((void *)&cdr, x, y);
            break;
         case mINT:
            int *id1, idr;
            id1 = (int  *)d1;
            idr = (int)*id1 * (int)xx;
            result->Populate((void *)&idr, x, y);
                                                        break;
         case mLONG:
                                                        long *ld1, ldr;
            ld1 = (long *)d1;
            ldr = (long)*ld1 * (long)xx;
            result->Populate((void *)&ldr, x, y);
                                                        break;
         case mFLOAT:
            float *fd1, fdr;
            fd1 = (float*)d1;
            fdr = (float)*fd1 * (float)xx;
            result->Populate((void *)&fdr, x, y);
                                                        break;
         case mDOUBLE:
            double *dd1, ddr;
            dd1 = (double*)d1;
            ddr = (double)*dd1 * (double)xx;
            result->Populate((void *)&ddr, x, y);
            break;
         }//end switch
      }
   }

   return(*result);
}

sMatrix& sMatrix::operator =(sMatrix &source)
{
            int x,y;
            int i, j,type;
            void *tempholder;
            if(this!=&source)
            {

                        if (source.GetSize(&x, &y)<0) return (*this);
                        source.GetType(&type);
                        delete [] m_data;
                        m_datatype=-1;
                        m_datatypesize = 0;
                        m_x = 0;
                        m_y = 0;
                        if (SetType(type))
                                    return (*this); //failure operating on dest
                        if (SetSize(x, y))
                                    return (*this); //error operating on dest.
                        tempholder = malloc(sizeof(double));//Type is assumed double always
                        for (i=1;i<=m_x;i++)
                                    for (j=1;j<=m_y;j++)
                                    {
                                                source.HGetElement(tempholder, i, j);
                                                Populate(tempholder, i, j);
      }
                        free(tempholder);
            }
            return(*this);
}
```

```
/*****************************************************************
** non operator based math function
*****************************************************************/


/*****************************************************************
** int Transpose(sMatrix&)
**  This function does just what it's name implies, transposing
**  the current matrix (that is the one that this function is
**  called from, storing the result in the matrix provided as
**  the argument to this function.
**
**  The Smatrix provided to this function must be completely
**  empty and not set in anyway.
*****************************************************************/

int sMatrix::Transpose(sMatrix& dest){

   int i,j;

   void *tempholder;

   if (m_x == 0 || m_y == 0) return (-4); //not set
   if (dest.SetType(m_datatype))
      return (-25); //failure operating on dest
   if (dest.SetSize(m_y, m_x))
      return (-26); //error operating on dest.

   tempholder = malloc(m_datatypesize);

   for (i=1;i<=m_x;i++)
      for (j=1;j<=m_y;j++){

         HGetElement(tempholder, i, j);
         dest.Populate(tempholder, j, i);

         }

   free (tempholder);
   return (0);

   }

/*****************************************************************
**  int SVD(sMatrix& u, sMatrix& w, sMatrix& vT);
**      This SVD algorithm used here differs from the one found
**      in Numerical Recipies in C in a number of key ways.
**
**      First, in the book version of the algorythm, the original
**      version, the Matrix A (which is the matrix be decomposed
**      is destroyed and is instead used to hold U.
**
**      Secondly, the algorythm presented in NRinC states that
**      you must square the matrix before hand, if it is not
**      already.  This functionality has been moved into the
**      function to make it easier to work with.
**
**      Thirdly, the NRinC version provides an untransposed V.
**      This version transposes V before returning it.
**
**      Finally, the book version returns W as a vector of the
**      diagonal Matrix W.  This version returns W as a full
**      matrix, with the appropriate zero padding added.
**
**      This function will only proceed if the current Matrix A
**      if of type double.  Use the function itof of this
**      class to convert integer matricies to floating point
**      matricies.  To reverse use ftoi.
**
**      u, w and vT must be provided empty, with nothing set.
```

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

/*int sMatrix::SVD(sMatrix& u, sMatrix& ws, sMatrix& vT){

    int flag, i, its, j, jj, k, l, nm;
    double c, f, h, s, x, y, z;
    double anorm=0.0, g=0.0, scale=0.0;
    double *rv1;
    int type;
    //extra variables not in NRinC
    int m, n;
    double **a, **v, *w;

    //ensure other matricies are blank

    if (!u.GetType(&type)) return (-1);
    if (!ws.GetType(&type)) return (-2);
    if (!vT.GetType(&type)) return(-3);

    //ensure that I am double

    if (m_datatype != mDOUBLE) return (-4);

    //now perform a row check.  If I am not the correct number of rows
    // (rows must great than columns).  If not I will create a temporary
    // clone of myself and zero stuff it.

    m = m_x;
    n = m_y;
    if (m_x < m_y) m = m_y;

    sMatrix sqMatrix(mDOUBLE, m, n);  //double type and m_x sizes

    c = 0;
    sqMatrix.PopulateArea((void*)&c, 1, 1, m, n);
    sqMatrix.PopulateArea(this, 1, 1); //copy this matrix into the new squareone.

    //allocate memory
    rv1 = new double[n+1];
    a = (double **)malloc((unsigned)(m+1)*sizeof(double*));
    v = (double **)malloc((unsigned)(n+1)*sizeof(double*));
    w = (double *)malloc((unsigned)(n+1) * sizeof(double));

            for (i=0;i<=m+1;i++){
                    a[i]=(double *) malloc((unsigned)(n+1)*sizeof(double));
            }
            for (i=0;i<=n+1;i++){
                    v[i]=(double *) malloc((unsigned)(n+1)*sizeof(double));
            }

    //copy contents of this matrix into **a
            double tempd;

            for (i=1;i<=m;i++){
                    for (j=1;j<=n;j++){
                    sqMatrix.GetElement((void *)&tempd, i, j);
                    a[i][j]=tempd;
                    }
            }

    //back to the book

    for (i=1;i<=n;i++){
       l = i + 1;
       rv1[i]=scale*g;
       g = s = scale = 0.0;
       if (i <= m){
          for (k=i;k<=m;k++) scale +=fabs(a[k][i]);
          if (scale){
             for (k=i;k<=m;k++){
```

```
            a[k][i] /= scale;
            s += a[k][i] * a[k][i];
            }
         f = a[i][i];
         g = -SIGN(sqrt(s), f);
         h = f*g-s;
         a[i][i]=f-g;
         if (i!=n){
            for (j=l;j<=n;j++){
               for (s=0.0,k=i;k<=m;k++) s += a[k][i]*a[k][j];
               f = s/h;
               for (k=i;k<=m;k++) a[k][j] += f*a[k][i];
               }
            }
            for (k=i;k<=m;k++) a[k][i] *= scale;
         }
      }
   w[i]=scale*g;
   g=s=scale=0.0;
   if (i <= m && i !=n){
      for (k=l;k<=n;k++) scale += fabs(a[i][k]);
      if (scale){
         for (k=l;k<=n;k++){
            a[i][k] /= scale;
            s += a[i][k] * a[i][k];
            }
         f = a[i][l];
         g = -SIGN(sqrt(s), f);
         h = f*g-s;
         a[i][l]=f-g;
         for (k=l;k<=n;k++) rv1[k]=a[i][k]/h;
         if (i != m ){
            for (j=l;j<=m;j++){
               for (s=0.0,k=l;k<=n;k++) s += a[j][k]*a[i][k];
               for (k=l;k<=n;k++) a[j][k] += s*rv1[k];
               }
            }
         for (k=l;k<=n;k++) a[i][k] *= scale;
         }
      }
   anorm=MAX(anorm, (fabs(w[i])+fabs(rv1[i])));
   }
//accumulation of right-hand transformations
for (i=n;i>=1;i--){
   if (i < n){
      if (g){
         for (j=l;j<=n;j++)  //double division to avoid possible underflow
            v[j][i] = (a[i][j]/a[i][l])/g;
         for (j=l;j<=n;j++){
            for (s=0.0, k=l;k<=n;k++) s+= a[i][k]*v[k][j];
            for (k=l;k<=n;k++) v[j][k] += s*v[k][i];
            }
         }
      for (j=l;j<=n;j++) v[i][j]=v[j][i]=0.0;
      }
   v[i][i]=1.0;
   g=rv1[i];
   l=i;
   }
//accumulation of left-hand transformations

for (i=n;i>=1;i--){
   l=i+1;
   g=w[i];
   if (i < n)
      for (j = l; j <= n; j++) a[i][j]=0.0;
   if (g) {
      g = 1.0/g;
      if (i != n){
         for (j=l;j<=n;j++) {
```

```
            for (s=0.0, k=l;k<=m;k++) s+= a[k][i]*a[k][j];
            f = (s/a[i][i])*g;
            for (k=i;k<=m;k++) a[k][j] += f*a[k][i];
            }
         }
      for (j=i;j<=m;j++) a[j][i] *= g;
      }
   else {
      for (j=i;j<=m;j++) a[j][i] = 0.0;
      }
   ++a[i][i];
   }
//diagonalization of bidiagonal form.
for (k=n;k>=1;k--){              //loop over singular values
   for (its=1;its<=30;its++){  //loop over allowed iterations
      flag = 1;
      for (l=k;l>=1;l--){     //test for splitting:
         nm = l -1;          //note that rv1[1] is always zero
         if ((double)(fabs(rv1[l])+anorm) == anorm){
            flag = 0;
            break;
            }
         if ((double)(fabs(w[nm])+anorm) == anorm) break;
         }
      if (flag){
         c = 0.0;           //cancellation of rv1[i], if l>1
         s = 1.0;
         for (i=l;i<=k;i++){
            f = s *rv1[i];
            rv1[i] = c *rv1[i];
            if ((double)(fabs(f)+anorm) == anorm) break;
            g = w[i];
            h = PYTHAG(f,g);
            w[i] = h;
            h = 1.0/h;
            c = g * h;
            s = (-f*h);
            for (j=1;j<=m;j++){
               y = a[j][nm];
               z = a[j][i];
               a[j][nm] = y *c + z * s;
               a[j][i] = z * c - y * s;
               }
            }
         }
      z = w[k];
      if (l==k){              //convergence
         if (z < 0.0){       //singular value is made nonnegative
            w[k] = -z;
            for (j=1;j<=n;j++) v[j][k]=(-v[j][k]);
            }
         break;
         }
      if (its == 30){
         free (a);
         free (v);
         free (w);
         delete[] rv1;
         return(-99); //no convergence
         }
      x = w[l];
      nm = k-1;
      y = w[nm];
      g = rv1[nm];
      h = rv1[k];
      f = ((y-z)*(y+z)+(g-h)*(g+h))/(2.0*h*y);
      g = PYTHAG(f, 1.0);
      f = ((x-z)*(x+z)+h*((y/(f+SIGN(g, f)))-h))/x;
      //next qr transformation
      c = s = 1.0;
```

```
        for (j=l;j<=nm;j++){
            i = j + 1;
            g = rv1[i];
            y = w[i];
            h = s * g;
            g = c * g;
            z = PYTHAG(f, h);
            rv1[j] = z;
            c = f/z;
            s = h/z;
            f = x * c + g * s;
            g = g * c - x * s;
            h = y * s;
            y = y * c;
            for (jj = 1;jj<=n;jj++){
                x = v[jj][j];
                z = v[jj][i];
                v[jj][j] = x * c + z * s;
                v[jj][i] = z * c - x * s;
                }
            z = PYTHAG(f,h);
            w[j] = z; //rotation can be arbitrary is z= 0
            if (z){
                z = 1.0/z;
                c = f * z;
                s = h * z;
                }
            f = (c * g) + (s * y);
            x = (c * y) - (s * g);
            for (jj=1;jj<=m;jj++){
                y = a[jj][j];
                z = a[jj][i];
                a[jj][j] = y * c + z * s;
                a[jj][i] = z * c - y * s;
                }
            }
        rv1[l] = 0.0;
        rv1[k] = f;
        w[k] = x;
        }
    }

//copy from temporary matricies and vectors to final matricies

//first initialize the emtpy matricies

if (u.SetType(mDOUBLE)){
    //free (a);
    //free (v);
    //free (w);
    delete[] rv1;
    return (-50);
    }
if (ws.SetType(mDOUBLE)){
    //free (a);
    //free (v);
    //free (w);
    delete[] rv1;
    return (-50);
    }
if (vT.SetType(mDOUBLE)){
    //free (a);
    //free (v);
    //free (w);
    delete[] rv1;
    return (-50);
    }

            tempd = 0.0;
if (u.SetSize(m, n)){
```

```
    //free (a);
    //free (v);
    //free (w);
    delete[] rv1;
    return (-50);
    }
        u.PopulateArea((void*)&tempd, 1, 1, m, n);
  if (ws.SetSize(n, n)){

    //free (a);
    //free (v);
    //free (w);
    delete[] rv1;
    return (-50);
    }
        ws.PopulateArea((void*)&tempd, 1, 1, n, n);

  if (vT.SetSize(n, n)){
    //free (a);
    //free (v);
    //free (w);
    delete[] rv1;
    return (-50);
    }
        vT.PopulateArea((void*)&tempd, 1, 1, n, n);
  //okay everything should be inited..
  //now copy data

  double t;
  for (i = 1; i <= m; i++){
    for (j = 1; j <= n; j++){
      t = a[i][j];
      u.Populate((void*)&t, i, j);
      }
    }

  //u should be ready now, it's time to copy w.

  for (i = 1; i <= n; i++){
    t = w[i];
    ws.Populate((void *)&t, i, i);
    }

  //ws should be copied over now.
  //next copy and transpose V (transpose takes place during copy).
  //we can do this safely becuase v is guaranteed to be n x n

  for (i = 1; i <= n; i++){
    for (j = 1; j <= n; j++){
      t = v[i][j];
      vT.Populate((void *)&t, j, i); //note the transpose
      }
    }

  //clean up
  //free (a);
  //free (v);
  //free (w);
  delete[] rv1;
  return (0);
  }

/*****************************************************************
**  int Psuedoinv(sMatrix &)
**     This function calculates the psuedoinverse of this
**     matrix.  It does so using the SVD function.
**     This function is non-destrcutive.  That is the matrix
**     remains intact after the operation.
**
**     This matrix must be double in order for this to work.
```

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

/*int sMatrix::Psuedoinv(sMatrix & retmat){

   sMatrix u, w, vT, uT, v;
   int n, m, i;
   double t, s;
   sMatrix temphold;

   //ensure that I am double
   if (m_datatype != mDOUBLE) return (-4);

   if (SVD(u, w, vT)) return (-9); //svd failed

   if (w.GetSize(&m, &n)) return (-8); //internal error

   //compute s+ (inverse s)
   for (i = 1; i <= n ; i++){
      w.GetElement((void *)&t, i, i);
      if (t == 0.0) s = 0.0;
      else{
         s = 1.0/t;
         }
      w.Populate((void *)&s, i, i);
      }

   //u and v need to be transposed..

   if (u.Transpose(uT)) return (-8); //internal error
   if (vT.Transpose(v)) return (-8); //internal error

   //now multiply all together. (a+ = u * s+ * vT)

   temphold = w * uT;
   retmat = v * temphold;

   return (0);

   }

/*****************************************************************
**  int Chol(sMatrix &);
**
**  The notes for the implementation details of this function
**  can be found in the header file (Matrix.h)
**
**  Additional notes:  This must be double values as well.
**  A negative error means error on entry, a positive error
**  means that the error occured during calculation.
*****************************************************************/

/*int sMatrix::Chol(sMatrix& R){

   int temp;
   if (m_x != m_y) return (-1);  //Matrix MUST be nxn

   if (!R.GetType(&temp)) return (-2);  //dest matrix must not be init'ed.

   if (m_datatype != mDOUBLE) return (-3); //must be double

   int p, j, k, retval = 0;
   double atmp1, rtmp1, stmp1, ttmp1, utmp1, vtmp1;

   atmp1 = rtmp1 = stmp1 = ttmp1 = utmp1 = vtmp1 = 0.0;

   R.SetType(mDOUBLE);
   R.SetSize(m_x, m_y);
   double m = 0.0;
   R.PopulateArea((void*)&m, 1, 1, m_x, m_y);
```

```
   for (p = 1; p <= m_x; p++){

      //this code is for the diagnolas ONLY!
      if (p>1){
         HGetElement((void *)&atmp1, p, p);
         stmp1 = 0.0;
         for (k = 1; k <p; k++){
            R.GetElement((void *)&ttmp1, k, p);
            vtmp1 = ttmp1 * ttmp1;
            stmp1 += vtmp1;
            }
         rtmp1 = sqrt((atmp1 - stmp1));
         R.Populate((void *)&rtmp1, p, p);
         }
      else {  //p == 1
         HGetElement((void *)&atmp1, p, p);
         rtmp1 = sqrt(atmp1);
         R.Populate((void*)&rtmp1, p, p);
         }
      //check to make sure code won't blow up.
      if (rtmp1 <= .0001){  //<----------arbitrary can be changed!
         retval = 1;
         break;
         }

      //end diagonal only code

      //this code processes anything that isn't on the diagonal.
      for (j = 1; j <= m_x; j++){
         stmp1 = 0.0;
         if (j > p){
            HGetElement((void *)&atmp1, p, j);
            if (p >1) for (k = 1; k < p; k++){
               R.GetElement((void*)&ttmp1, k, p);
               R.GetElement((void*)&vtmp1, k, j);
               stmp1 += (ttmp1 * vtmp1);
               }
            R.GetElement((void*)&utmp1, p, p);
            rtmp1 = ((atmp1 - stmp1)/utmp1);
            R.Populate((void*)&rtmp1, p,j);
            }//end if j>p
         }//end j loop
      } //end p loop
   return (retval);
   }*/
int sMatrix::PrintMatrix(int startx, int starty, int endx, int endy)
{
               int i,j;
               double tempDouble;
               if((startx<1)||(starty<1))
                              return -1;
               if((startx>m_x)||(starty>m_y))
                              return -1;
               if((startx>endx)||(starty>endy))
                              return -1;

                for(i=startx;i<=endx;i++)
                              {
                                             for(j=starty;j<=endy;j++)
                                                            {
                                                                           cout << "Element "<< i <<"  "<<j<< "  ";
                                                                           GetElement(&tempDouble,i,j);
                                                                           cout << tempDouble << endl;
                                                            }
                              }
               return 0;
}

int sMatrix::BiDiagonalize(sMatrix *Bptr, sMatrix *Wptr, sMatrix *rv1, double *aNormPtr)
{
```

```
int i,j,k,l,result,m,n;
        double scale,g,s,c,h,f,tempDouble,tempDouble2;
        if((Bptr==NULL)||(Wptr==NULL)||(rv1==NULL)||(aNormPtr==NULL))
                return(-1);
if((Bptr->GetSize(&m,&n))<0)
                return(-1);
if((m!=m_x)||(n!=m_y))
                 return(-1);
if(((Wptr->m_x)!=m_y)||((Wptr->m_y)!=1))
                return(-1);
if(((rv1->m_x)!=m_y)||((rv1->m_y)!=1))
                return(-1);
        Bptr->PopulateArea(this, 1, 1); //copy current matrix into B
c = 0;
Wptr->PopulateArea((void*)&c, 1, 1, n, 1);
        rv1->PopulateArea((void*)&c, 1, 1, n, 1);
        g=0;
        scale=0;
        (*aNormPtr)=0;
        for(i=1;i<=n;i++) //Repeat for all rows and columns
        {
        l=i+1;//Operations aplied to elements following the diagonal one
        tempDouble=scale*g;
   result=rv1->HSetElement(&tempDouble,i,1); //Superdiagonal for row operations is -segmma*scale
        //Recall that scaling is applied to the components of the vector V
        g=0;
        s=0;
        scale=0;
        if(i<=m) //Notice that column operations are applied to the first min(m,n)
        {
        for(k=i;k<=m;k++)
        {
                result=Bptr->HGetElement(&tempDouble,k,i);
                scale=scale+fabs(tempDouble);//Get l1 norm of the ith column to determine ui
        }
        if(scale!=0)
        {
                for(k=i;k<=m;k++)
                {
                result=Bptr->HGetElement(&tempDouble,k,i);
                tempDouble=tempDouble/scale; //scale components of the ith column
                result=Bptr->HSetElement(&tempDouble,k,i);
                s=s+(tempDouble*tempDouble); //l2 norm of the current column
                }
                result=Bptr->HGetElement(&f,i,i); //This is the first element (v1)
                g=-1*GetSign(sqrt(s),f); //g is -1*segmma
                h=f*g-s; // h is -Pi where Pi is the scale factor used in Householder transformations
                //See the book of Stewart
                tempDouble=f-g;
                result=Bptr->HSetElement(&tempDouble,i,i); // first component of u differes
                //from a[i,i] by segmma; v1=zetta1+segmma
                for(j=l;j<=n;j++)
                {
                        s=0;
                        for(k=i;k<=m;k++)
                        {
                        result=Bptr->HGetElement(&tempDouble,k,i);
                        result=Bptr->HGetElement(&tempDouble2,k,j);
                        s=s+tempDouble*tempDouble2; //This is Sum(Vi*Alphai); see Stewart
                        }
                        f=s/h; //divide by -Pi
                        for(k=i;k<=m;k++)
                        {
                        result=Bptr->HGetElement(&tempDouble,k,i);
                        result=Bptr->HGetElement(&tempDouble2,k,j);
                        tempDouble2=tempDouble2+f*tempDouble; //Apply ith column transformaton
                        // to the jth column
                        result=Bptr->HSetElement(&tempDouble2,k,j);
                        }
                        }
```

```
		for(k=i;k<=m;k++)
		{
		result=Bptr->HGetElement(&tempDouble,k,i);
		tempDouble=tempDouble*scale;  //Remove the scaling from the transformation vector
			result=Bptr->HSetElement(&tempDouble,k,i);
		}
		} //end if scale~=0
}// end if i<=m
tempDouble=scale*g; //This is the ith diagonal element (-segmma)
result=Wptr->HSetElement(&tempDouble,i,1);
//Recall that scaling was applied before and it is removed here before saving
//to w(i,1)
g=0;
s=0;
scale=0; //Previous lines are initializations
if((i<=m)&&(i!=n))
		//Repeat for the proper number of rows transformations
{
		for(k=l;k<=n;k++)
		{
			result=Bptr->HGetElement(&tempDouble,i,k);
			scale=scale+fabs(tempDouble);//The l2 norm of the ith row starting from (i+1)
		}
		if(scale!=0)
		{
			for(k=l;k<=n;k++)
			{
				result=Bptr->HGetElement(&tempDouble,i,k);
				tempDouble=tempDouble/scale; //scale components of the ith row
				result=Bptr->HSetElement(&tempDouble,i,k);
				//starting from component (i+1)
				s=s+tempDouble*tempDouble;//l2 norm of scaled row
			}
			result=Bptr->HGetElement(&f,i,l);//This is v1 (first component of the
			//transformation vector
			g=-1*GetSign(sqrt(s),f);//This is -segmma
			h=f*g-s; //This is -Pi
			tempDouble=f-g; //First component of transformation vector is B(i,l)+segmma
			result=Bptr->HSetElement(&tempDouble,i,l);
			for(k=l;k<=n;k++)
			{
				result=Bptr->HGetElement(&tempDouble,i,k);
				//rv1 is used here as temprary
				tempDouble=tempDouble/h;
				//storage; however rv1(i,1) is not changed
				result=rv1->HSetElement(&tempDouble,k,1);
				// it is set at the start of the
				// main loop
			}
			for(j=l;j<=m;j++)//Apply ith row transformation to all following rows
			{ //starting from column(i+1)
				s=0;
				for(k=l;k<=n;k++)
				{
					result=Bptr->HGetElement(&tempDouble,i,k);
					result=Bptr->HGetElement(&tempDouble2,j,k);
					s=s+tempDouble*tempDouble2;
					//This is Sum(Vi*Alphai); see Stewart
				}
				for(k=l;k<=n;k++)
				{
					result=Bptr->HGetElement(&tempDouble,j,k);
					result=rv1->HGetElement(&tempDouble2,k,1);
					tempDouble=tempDouble+s*tempDouble2;
					 //Apply ith row transformation
					// to the jth row
					result=Bptr->HSetElement(&tempDouble,j,k);
				}
			}
			for(k=l;k<=n;k++)
```

```
                                        {
                                                result=Bptr->HGetElement(&tempDouble,i,k);
                                                tempDouble=tempDouble*scale;
                                                        //Remove the scaling from the transformation vector
                                                result=Bptr->HSetElement(&tempDouble,i,k);
                                        }
                                }// end if scale~=0
                        }// end if((i<=m)&&(i~=n)
                        result=Wptr->HGetElement(&tempDouble,i,1);
                        result=rv1->HGetElement(&tempDouble2,i,1);
                        (*aNormPtr)=GetMax((*aNormPtr),(fabs(tempDouble)+fabs(tempDouble2)));
                }// end main loop ;for(i=1;i<=n;i++)
 return 0;
}
int sMatrix::AccmlteLeft(sMatrix *UPtr, sMatrix *WPtr)
{
 int m,n,l,i,j,k,mnmin;
 double c,g,f,s,tempDouble,tempDouble2;
 if((UPtr==NULL)||(WPtr==NULL))
        return (-1);
 if(((UPtr->m_x)!=m_x)||((UPtr->m_y)!=m_y))
        return(-1);
 UPtr->PopulateArea(this, 1, 1); //copy current matrix into U
 m=UPtr->m_x;
 n=UPtr->m_y;
 if(((WPtr->m_x)!=n)||((WPtr->m_y)!=1))
        return (-1);
 if(m<n)
        mnmin=m;
 else
        mnmin=n; //Number of Column transformations is equal to mnmin
 for (i=mnmin; i>=1;i--) //repeat for all transformations starting with the smallest one (1by1)
 {
        l=i+1; //Apply the ith transformation to columns starting with column (i+1)
        WPtr->HGetElement(&g,i,1); //g is -segmma
        for(j=l;j<=n;j++) // fill first row to the right of the element U(i,i) by zeros
        {
                c=0;
                UPtr->Populate(&c,i,j);
        }
        if(g!=0)
        {
                g=1/g;
                for(j=l;j<=n;j++)
                {
                        s=0;
                        for(k=l;k<=m;k++)
                        {
                                UPtr->HGetElement(&tempDouble,k,i);
                                UPtr->HGetElement(&tempDouble2,k,j);
                                s=s+tempDouble*tempDouble2; // This is Sum(ViAlphai)
                        }
                        UPtr->HGetElement(&tempDouble,i,i);
                        f=(s/tempDouble)*g;// This is -1*Sum(ViAlphai)/Pi
                        for(k=i;k<=m;k++) //Apply ith transformation to the jth column
                        {
                                UPtr->HGetElement(&tempDouble,k,j);
                                UPtr->HGetElement(&tempDouble2,k,i);
                                tempDouble=tempDouble+f*tempDouble2;
                                UPtr->HSetElement(&tempDouble,k,j);
                        }
                }
                for(j=i;j<=m;j++)
                {
                        UPtr->HGetElement(&tempDouble,j,i);
                        tempDouble=tempDouble*g;
                        UPtr->HSetElement(&tempDouble,j,i);// All elements U(j,i) are scaled by -1/segmma
                }
        }
        else
```

```
		{
			for(j=i;j<=m;j++)
				UPtr->HSetElement(&c,j,i);// segmma is zero so zero the corresponding vector
		}
		UPtr->HGetElement(&tempDouble,i,i);
		tempDouble=tempDouble+1;
	UPtr->HSetElement(&tempDouble,i,i);// increment the first element in the ith row by 1
 }
return 0;
}

int sMatrix::AccmlteRight(sMatrix *VPtr, sMatrix *rv1)
{
 int m,n,l,i,j,k,result;
 double c,g,s,tempDouble,tempDouble2;
 if((VPtr==NULL)||(rv1==NULL))
		return (-1);
 if(((VPtr->m_x)!=m_y)||((VPtr->m_y)!=m_y))
		return(-1);
 m=m_x;
 n=m_y;
 if(((rv1->m_x)!=n)||((rv1->m_y)!=1))
		return (-1);
 g=0;
 l=0;
 for(i=n;i>=1;i--)//Repeat for all transformations; Notice that the first one is the identity
 {
		if(i<n)
		{
			if(g!=0)
			{
				for(j=l;j<=n;j++) //Notice that l=i+1 effectively
				{
					result=GetElement(&tempDouble,i,j);
				   result=GetElement(&tempDouble2,i,l);
					tempDouble=(tempDouble/tempDouble2)/g;//This is equivalent to Vi/(-Pi)
					//Recall that g=-segmma and B(i,1)=v1 and Pi=v1*(-segmma)
					VPtr->HSetElement(&tempDouble,j,i);
				}
				for(j=l;j<=n;j++)
				{
					s=0;
					for(k=l;k<=n;k++)
					{
						result=GetElement(&tempDouble,i,k);
						VPtr->GetElement(&tempDouble2,k,j);
						s=s+tempDouble*tempDouble2;//This is Sum(Vi*Alphai)
					}
					for(k=l;k<=n;k++)
					{
						result=VPtr->HGetElement(&tempDouble,k,j);
						result=VPtr->GetElement(&tempDouble2,k,i);
						tempDouble=tempDouble+s*tempDouble2; //Update jth column of the
current V
						//See Stewart for more details
						//This last expression can be read as
						//V(k,j)=V(k,j)-(V(k,i)/Pi)*Sum(Vi*Alphai)
						VPtr->HSetElement(&tempDouble,k,j);
					}
				}
				//Everey iteration of thsi loop affects only the block (l:n,l:n) of V
			}
			for(j=l;j<=n;j++)
			{
				c=0;
				VPtr->HSetElement(&c,i,j);//This loop introduces zeros in the off diagonal parts of V
				VPtr->HSetElement(&c,j,i);// to make the upper right corner of V looks like the identity
			}
		}
		c=1;
```

```
            VPtr->HSetElement(&c,i,i);// Put 1 in the location (i,i) (remember the identity)
            rv1->HGetElement(&g,i,1); //This is -segmma of the corresponding transformation
            l=i;
 }
 return 0;
}

int sMatrix::SVDA(sMatrix *WPtr, sMatrix *rv1, sMatrix *UPtr,sMatrix *VPtr,int m, int n,double aNorm)
{
int i,j,k,l,nm,index,its,jj,result;
double c,s,f,g,h,x,y,z,tempDouble,tempDouble2;
if((WPtr==NULL)||(rv1==NULL)||(UPtr==NULL)||(VPtr==NULL))
            return -1;
if(((WPtr->m_x)!=n)||((WPtr->m_y)!=1)||((rv1->m_x)!=n)||((rv1->m_y)!=1))
            return -1;
if(((UPtr->m_x)!=m)||((UPtr->m_y)!=n)||((VPtr->m_x)!=n)||((VPtr->m_y)!=n))
            return -1;
for(k=n;k>=1;k--)//Repeat for all singular values
            {
                        for(its=1;its<=30;its++) //Maximum number of iteration for each SV is 30
                        {
                                    for(l=k;l>=1;l--)//test diagonals and subdiagonals for splitting and convergence
                                    {
                                                nm=l-1;
                                                rv1->GetElement(&tempDouble,l,1);
                                                if((fabs(tempDouble)+aNorm)==aNorm)//negligible off diagonal
                                                {
                                                            index=2; // Index=2 means that we have to check convergence part
                                                            break;
                                                }
                                                if(nm>0)
                                                {
                                                            WPtr->GetElement(&tempDouble,nm,1);
                                                            if((fabs(tempDouble)+aNorm)==aNorm)// negligible diagonal
                                                            {
                                                                        index=1;
                                                                        break;
                                                            }
                                                }
                                    }
                                    if(index==1) //Diagonal is negligible; neighbouring off diagonal is shifted
                                    {
                                                c=0;
                                                s=1;
                                                for(i=l;i<=k;i++)
                                                {
                                                            rv1->GetElement(&tempDouble,i,1);
                                                            f=s*tempDouble;//This is the new shifted element (see LinPack guide)
                                                            tempDouble=tempDouble*c; //Update the superdiagonal
                                                            rv1->HSetElement(&tempDouble,i,1);
                                                            if((fabs(f)+aNorm)==aNorm)//Is new shifted element negligible
                                                            {
                                                                        index=2; // Got to test for convergence
                                                                        break;
                                                            }
                                                            WPtr->GetElement(&g,i,1);//Diagonal Element before rotation
                                                            h=GetPythag(f,g);//sqrt(zetta^2+etta^2)
                                                            WPtr->HSetElement(&h,i,1);//New rotated diagonal element
                                                            h=1.0/h;//1/sqrt(zetta^2+etta^2)
                                                            c=g*h;//zetta/sqrt(zetta^2+etta^2)
                                                            s=-1*f*h;// -etta/sqrt(zetta^2+etta^2)
                                                            for(j=1;j<=m;j++)//This loop applies the rotation to shift the nondiagonal
element
                                                            {
                                                                        UPtr->HGetElement(&tempDouble,j,nm);
                                                                        UPtr->HGetElement(&tempDouble2,j,i);

            result=Rotate(&tempDouble,&tempDouble2,tempDouble,tempDouble,c,s);
                                                                        UPtr->HSetElement(&tempDouble,j,nm);//Notice that Rotation is
applied to U'
```

```
                                        UPtr->HSetElement(&tempDouble2,j,i);
                                }
                        }//end for(i=1;i<=k;i++)
                }//end if(Index==1)
                if(index==2)
                {
                        WPtr->GetElement(&z,k,1); //kth diagonal element
                        if(l==k)//Is the obtained SV the kth one (convergence)
                        {
                                if(z<0)//Is the obtained SV negative
                                {
                                        z=-1*z;
                                        WPtr->HSetElement(&z,k,1); //change sign of the kth diagonal
element
                                        for(j=1;j<=n;j++)
                                        {
                                                VPtr->GetElement(&tempDouble,j,k);
                                                tempDouble=tempDouble*-1;//Change sign of the kth
right eigenvector
                                                VPtr->HSetElement(&tempDouble,j,k);
                                        }
                                }
                                break; //Go to next value of k to get the next SV
                        }
                }
                //This Part is reached if no superdiagonal or diagonal is negligible or the
                //negligible diagonal is not the kth one (no convergence yet)
                if(its==30)
                {
                        //cout << "NO Convergence in 30 iterations" << endl;
                        exit(-1);
                }
                //This block calculates the necessary shift for the QR iteration
                WPtr->GetElement(&x,l,1);//Get the lth diagonal element
                nm=k-1;//This is the lowest index for a diagonal
                WPtr->GetElement(&y,nm,1);//lower left corner diagonal of the matrix
                rv1->GetElement(&g,nm,1);
                rv1->GetElement(&h,k,1);
                f=((y-z)*(y+z)+(g-h)*(g+h))/(2.0*h*y);
                g=GetPythag(f,1.0);
                f=((x-z)*(x+z)+h*((y/(f+GetSign(g,f)))-h))/x;
                //Next QR transformation
                c=1.0;
                s=1.0;
                for(j=l;j<=nm;j++)//Repeat for the lower left submatrix starting from l to (k-1)
                {
                //The QR iteration applies a right transformation followed by a left transformation
                        i=j+1;//Rotation is applied to neighbouring rows
                        rv1->GetElement(&g,i,1);
                        WPtr->GetElement(&y,i,1);
                        h=s*g;
                        g=c*g;
                        z=GetPythag(f,h);//sqrt(Zetta^2+etta^2)
                        rv1->HSetElement(&z,j,1); //Set transformed superdiagonal element
                        c=f/z; //C ans s are the rotation parameters
                        s=h/z;
                        //The next four lines prepars for the next rotation
                        f=(x*c)+(g*s);
                        g=(-1*x*s)+(g*c);
                        h=y*s;
                        y=y*c;
                        for(jj=1;jj<=n;jj++)//Apply rotation to accumulated right transformations
                        {
                                VPtr->HGetElement(&tempDouble,jj,j);
                                VPtr->HGetElement(&tempDouble2,jj,i);
                                result=Rotate(&tempDouble,&tempDouble2,tempDouble,tempDouble2,c,s);
                                VPtr->HSetElement(&tempDouble,jj,j);//Notice that Rotation is applied to U'
                                VPtr->HSetElement(&tempDouble2,jj,i);
                        }
                        z=GetPythag(f,h);
```

```
                                        WPtr->HSetElement(&z,j,1); //set Rotated diagonal
                                        if(z!=0)
                                        {
                                                z=1/z;
                                                c=f*z;
                                                s=h*z;
                                        }
                                        f=(c*g)+(s*y); //f and x are set for the next Right rotation
                                        x=(-1*s*g)+(c*y);
                                        for(jj=1;jj<=m;jj++)
                                        {
                                                UPtr->HGetElement(&tempDouble,jj,j);
                                                UPtr->HGetElement(&tempDouble2,jj,i);
                                                result=Rotate(&tempDouble,&tempDouble2,tempDouble,tempDouble2,c,s);
                                                UPtr->HSetElement(&tempDouble,jj,j);//Notice that Rotation is applied to U'
                                                UPtr->HSetElement(&tempDouble2,jj,i);
                                        }
                                }
                                tempDouble=0;
                                rv1->HSetElement(&tempDouble,l,1);
                                rv1->HSetElement(&f,k,1);
                                WPtr->HSetElement(&x,k,1);
                        }//end for its=1:30
                }//end for k=n:1
                return 0;
}

int sMatrix::PsuedoInverse(sMatrix *BInverse)
{
                int m,n,i,result;
                double tempDouble/*, temptest*/;
                m=m_x;
                n=m_y;
                sMatrix *C=new sMatrix(mDOUBLE,m,n);
                sMatrix *WPtr=new sMatrix(mDOUBLE,n,1);
                sMatrix *rv1= new sMatrix(mDOUBLE,n,1);
                sMatrix *U=new sMatrix(mDOUBLE,m,n);
                sMatrix *UColumn= new sMatrix(mDOUBLE,m,1);
                sMatrix *V=new sMatrix(mDOUBLE,n,n);
                sMatrix *VColumn=new sMatrix(mDOUBLE,n,1);
                double *aNormPtr=new double;
                if(BInverse==NULL)
                        return -1;
                if(((BInverse->m_x)!=m_y)||((BInverse->m_y)!=m_x))
                        return -1;

                result=BiDiagonalize(C,WPtr,rv1,aNormPtr);
                if (result<0)
                        return -1;
   result=C->AccmlteLeft(U,WPtr);
                if (result<0)
                        return -1;
   result=C->AccmlteRight(V,rv1);
                if (result<0)
                        return -1;
                result=SVDA(WPtr, rv1, U,V,m, n,(*aNormPtr));
                if (result<0)
                        return -1;
                tempDouble=0;
                delete C;
                C=NULL;
                delete rv1;
                rv1=NULL;
                delete aNormPtr;
                aNormPtr=NULL;
                BInverse->PopulateArea(&tempDouble, 1, 1, n, m); //Fill psuedoinverse with zeros
                //WPtr->PrintMatrix(1,1,n,1);
                //cin >> temptest;
                for(i=1;i<=n;i++)
                {
```

```
                        WPtr->GetElement(&tempDouble,i,1);//Get ith SV
                        //cout <<tempDouble;
                        if(tempDouble!=0)
                        {
                                tempDouble=1.0/tempDouble;
                                result=U->GetColumn(UColumn,i);
                                //cout << result;
                                if (result<0)
                                        return -1;
                                result=V->GetColumn(VColumn,i);
                                //cout << result;
                        //      VColumn->PrintMatrix(1,1,n,1);
                                //cout << result;
                        //      cin >> temptest;
                                if (result<0)
                                        return -1;
        sMatrix *UColumnT=new sMatrix;
        result=UColumn->Transpose((*UColumnT));
                        //      cout << result;
                        //      cin>>temptest;
                                if (result<0)
                                        return -1;
                                sMatrix *D;
                                (D)=&((*VColumn)*(*UColumnT));
                                sMatrix *E;
                                (E)=&((*D)*tempDouble);
                                (*BInverse)+=(*E);
                                delete D;
                                delete E;
                                //(*BInverse)=(*BInverse)+(*VColumn)*(*UColumnT)*tempDouble;
                                //(*BInverse)+=(*VColumn)*(*UColumnT)*tempDouble;
                                delete UColumnT;
                        }
                }
                delete U;
                delete V;
                delete UColumn;
                delete VColumn;
                delete WPtr;
   return 0;
}


//This function solves a sustem of equations using pseudo inverse
//The solution is allocated memory in the calling function
sMatrix& sMatrix::SolveEquationsSystem(sMatrix *b)
{
        int iColumnsA,iRowsA;
        int iResult;
        GetSize(&iRowsA,&iColumnsA);
        sMatrix *x;
        sMatrix *Ain=new sMatrix(mDOUBLE,iColumnsA,iRowsA);
        if(Ain==NULL)
                exit(-1);
        iResult=PsuedoInverse(Ain);
        if(iResult<0)
        {
                delete Ain;
                Ain=NULL;
                exit(-1);
        }
        x=&((*Ain)*(*b));
        delete Ain;
        return(*x);
}
```

```
/*****************************************************************
**  Helper functions.. these are only called by the class
*****************************************************************/

int sMatrix::HGetElement(void * data, int Row , int Column){

   unsigned int tempptr =
      (unsigned int)m_data;   //setup a temporary pointer to work with
            unsigned int tempctr = 0;

   if (Row > 1)
      tempctr=(m_datatypesize * (m_y*(Row-1))); //position for row

   if (Column > 1)
      tempctr +=(m_datatypesize * (Column-1));  //positon for column


            tempptr = (unsigned int)m_data + tempctr;

   /*  We have now moved the pointer to the location of the element  **
   **  we want to set.  Now just set the data and return            */

   if (!memcpy (data, (void *)tempptr, m_datatypesize)){
      return (-1);  //There is an error return failure
      }

   return(0);
   }

int sMatrix::HSetElement(void * data, int Row, int Column){

   unsigned int tempptr =
      (unsigned int)m_data;   //setup a temporary pointer to work with

            unsigned int tempctr = 0;

   if (Row > 1)
      tempctr=(m_datatypesize * (m_y*(Row-1))); //position for row

   if (Column > 1)
      tempctr +=(m_datatypesize * (Column-1));  //positon for column


            tempptr = (unsigned int)m_data + tempctr;


   /*  We have now moved the pointer to the location of the element  **
   **  we want to set.  Now just set the data and return            */

   if (!memcpy ((void *)tempptr, data, m_datatypesize)){
      return (-1);  //There is an error return failure
      }

   return (0);
   }

double sMatrix::GetSign(double a, double b)
{
            if(b>=0)
                        return(fabs(a));
            else
                        return(-1*fabs(a));
}

double sMatrix::GetMax(double a, double b)
{
            if(a>b)
                        return(a);
            else
                        return(b);
```

```
}

//This function finds the length of a two component vector and avoids overflow or underflow
double sMatrix::GetPythag(double a, double b)
{
	double at,bt,ratio;
	at=fabs(a);
	bt=fabs(b);
	if(at>bt)
	{
		ratio=bt/at;
		return(at*sqrt(1+ratio*ratio));
	}
	else
	{
		ratio=at/bt;
		return(bt*sqrt(1+ratio*ratio));
	}
}

int sMatrix::Rotate(double *aOut, double *bOut, double aIn, double bIn, double c, double s)
{
	if((aOut==NULL)||(bOut==NULL))
		return -1;
	(*aOut)=(aIn*c)+(bIn*s);
	(*bOut)=(-1*aIn*s)+(bIn*c);
	return 0;
}

//This function returns a pointer to the data

void * sMatrix::GetPointer()
{
	return(m_data);
}
//Get the L2 norm of a vector
double  sMatrix::GetL2Norm()
{
	int i;
	double dSum, dValue;
	if((m_x<=0)||(m_y!=1))// Vector should have only one column and a positive number of rows
		return 0.0;
	dSum=0;
	for(i=1;i<=m_x;i++)
	{
		GetElement(&dValue,i,1);
		dSum=dSum+(dValue*dValue);
	}
	return(sqrt(dSum));
}

//This function updates the current matrix using Broyden's update
//It recieves the new vector, the old vector and the step
//It returns zero if all OK otherwise it returns -1
int sMatrix::BroydenUpdate(sMatrix* RNew, sMatrix* ROld, sMatrix* Step)
{

	int iNoRowsRNew, iNoRowsROld, iNoRowsStep, iNoColsRNew,iNoColsROld, iNoColsStep;
	double dNorm,dNormInverse;
	if((RNew==NULL)||(ROld==NULL)||(Step==NULL))//Pointers properly set
		return -1;//return an error flag
	//The following lines check that sizes are proper
	RNew->GetSize(&iNoRowsRNew,&iNoColsRNew);//Get size of new response
	ROld->GetSize(&iNoRowsROld,&iNoColsROld);//Get size of old response
	Step->GetSize(&iNoRowsStep,&iNoColsStep);//Get size of step
	if((iNoColsRNew!=1)||(iNoColsROld!=1)||(iNoColsStep!=1))//Are they column vectors
		return -1;//return an error flag
	if((iNoRowsRNew!=iNoRowsROld)||(iNoRowsRNew!=m_x)||(iNoRowsStep!=m_y))//Check number of rows
		return -1;//return an error flag
	dNorm=Step->GetL2Norm();//Get sqrt(hTh)
```

```
	dNorm=dNorm*dNorm;//This is hTh
	sMatrix* pDF=&((*RNew)-(*ROld));//Get Rnew-Rold
	sMatrix* pBh=&((*this)*(*Step));//Get Bh
	sMatrix* pNumerator=&((*pDF)-(*pBh));//This is Rnew-Rold-Bh
	delete pDF;//free memory allocated to pDF
	delete pBh;//free memory allocated to Bh
	dNormInverse=1.0/dNorm;
	sMatrix* pScaledNumerator=&((*pNumerator)*dNormInverse);//this is Rnew-Rold-Bh/hTh
	delete pNumerator;
	sMatrix *pStepTranspose=new sMatrix;//This  is hT
	if(pStepTranspose==NULL)//Did memory allocation failed
	{
		delete pScaledNumerator;//delete allocated memory
		return -1;//return an error flag
	}
	Step->Transpose(*pStepTranspose);//Get hT
	sMatrix *pIncrement=&((*pScaledNumerator)*(*pStepTranspose));//this is (Rnew-Rold-Bh/hTh)hT
	delete pScaledNumerator;//free memory
	delete pStepTranspose;//free memory
	(*this)+=(*pIncrement);//Add increment
	delete pIncrement;//free memory
	return 0;//return success flag
}
```

APPENDIX C

Momentum.h

```
//
// Momentum.h: an interface to the Momentum class
//
//////////////////////////////////////////////////////////////////////

//#if !defined(AFX_MOMENTUM_H__24C09EA5_4CE9_11D4_9EB5_006094EB072F__INCLUDED_)
//#define AFX_MOMENTUM_H__24C09EA5_4CE9_11D4_9EB5_006094EB072F__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "..\SMX_SOURCE\Simulator.h"
#include "Simulator.h"

/// define some erros flags

#define CANNOT_RUN_ADS                  1
#define HPADS_WND_NOTFOUND      2
#define PRJ_NOTFOUND                    3
#define DSN_NOTFOUND            4
#define LAYOUTWND_NOT_FOUND     5
#define CANNOT_OPEN_INPUTFILE   6
#define CANNOT_CREATE_OUTPUTFILE 7

class Momentum : public Simulator
{
public:
        Momentum( CString InputFileName, CString OutputFileName, int HelpFlag);
        virtual ~Momentum();
        void ReadInput();
        void SetAllParameters(int NofPoints, int NofParameters, double *Parameters);
        void SetNofPoints(int NofPoints);
        void UncheckDlgButton( CWnd *ParentWnd, int ButtonID);
        void PressBotton(CWnd *ParentWnd, int ButtonID);
        void FillEditBox(CWnd* ParentWnd,CString EditBoxStr, int EditBoxID);
        CWnd * FindWindowExt( const char * CaptionName,CWnd *StartWnd);
        CWnd * FindChildWindow(CWnd *ParentWnd,const char * ClassName, const char * WndCaption);
        void test(void);
        void SetMultiplePointOption(BOOL bMultiplePoints);
        int WriteInputFiles();
        int Simulate();
        int GetResult(sMatrix &);

private:

        void CallHelp();
        void GetMomDataFileName(CString &MomDataFileName);
        int DesignNotFound();
        void WriteOutputFile();
        void SaveResults( CWnd *LayoutWindow);
        void WaitForSimulation(void);
        void SetLayoutWndCaption(char *, CString &LayoutWndCaption);
        void OpenDesignFile(CWnd * LayOutWindow);
        void SetSimulationWindow(void);
        void SetIniLayoutNumber(void);
        int CannotOpenDesign(void);
        void PressSubMenuItem(CWnd *WndPtr, int *SubMenuIndex,int NoSubMenues,int ItemIndex);
        // select a submenu of the main window and activate an item
        int  FindWindow(char *WindClass, char *WindTitle,
                        CWnd  *& WndPtr,  double DurationInSec);
        // returns 0 if succedd an non zero if he fails to find it after DurationInSec
        void OpenAdsPrj(CWnd *AdsWnd);  // open the hpads project
        void  SetParametersWindow();  // set the values of the design parameters
```

```
        void RunAds(CWnd*& hpads_Window);  // run hpads and prepare the macro file
        int  WriteAdsMacroFile(void);  // write the hpads macro file
        void CloseAds(CWnd *WndPtr);  // close hpads

        CString m_sOutputFileName;
        CString m_sInputFileName;
        CString m_sLayoutCaption;
        CString m_sProjectName;  // contains the full path of the project
        CString m_sDesignName;  // contains the design name

        int m_iHelpFlag;
        int m_iCurrentPoint;
        int m_iNofPoints;
        int m_iLayoutNumber;
        int  m_iErrorFlag;                  // not 0 if an error happens
        double * m_pAllParameters;

};

//#endif // !defined(AFX_MOMENTUM_H__24C09EA5_4CE9_11D4_9EB5_006094EB072F__INCLUDED_)
```

Momentum.cpp

```
#include "stdafx.h"
#include "Momentum.h"
#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

Momentum::Momentum(CString InputFileName="input.dat",
                   CString OutputFileName="output.dat", int HelpFlag=0)
{
        m_iErrorFlag=0;
        m_iCurrentPoint=1;
        m_pAllParameters=NULL;
        m_sInputFileName=InputFileName;
        m_sOutputFileName=OutputFileName;
        m_iHelpFlag=HelpFlag;

}

//////////////////////////////////////////////////////////////////////
// Construction
//////////////////////////////////////////////////////////////////////

int Momentum::GetResult(sMatrix &)
{

        return 1;
}
//////////////////////////////////////////////////////////////////////
// Destruction
//////////////////////////////////////////////////////////////////////

Momentum::~Momentum()
{
        if(m_pAllParameters!=NULL)
                delete [] m_pAllParameters;
}

//////////////////////////////////////////////////////////////////////
// this function calls HpAds Momentum
// to simulate the structure at multiple
// points in different frequency bands
//////////////////////////////////////////////////////////////////////

int Momentum::Simulate()
{

// check for help option

        if(m_iHelpFlag)
        {
                CallHelp();
                return 0;
        }

        ReadInput(); // read the input file
        CWnd * hpadsWindow;
        if(m_iErrorFlag) return 0; // return if an error happens
        RunAds(hpadsWindow); // run hpads
        WriteAdsMacroFile(); // write the macro file which is used to open the project
        OpenAdsPrj(hpadsWindow);  // open the project by playing the Macro file back
        Sleep(1000);                                    // wait till the project is open
```

```
	for (int i=1; i<=m_iNofPoints; i++)
	{
		SetParametersWindow(); // set the design parameters windows
		Sleep(2000);				// wait till the window finishes it job
		SetSimulationWindow(); // set frequencies,launch the simulator
		Sleep(1000);
	}
	CloseAds(hpadsWindow); // close hpads
	return 1;
}

//////////////////////////////////////////////////////////////////////
// this function is inherited from the
// simulator class and does nothing here
//////////////////////////////////////////////////////////////////////

int Momentum::WriteInputFiles()
{
	return 1;
}

//////////////////////////////////////////////////////////////////////
// I did't use this option
// it can be used in future development though
//////////////////////////////////////////////////////////////////////

void Momentum::SetMultiplePointOption(BOOL bMultiplePoints)
{
}

//////////////////////////////////////////////////////////////////////
// write the macro file of the current momentum project
// this macro file only opens the momentum projet
// other ways to open the project are very difficult to
// program
// a typical macro file takes the form
// /*BEGIN*/
// de_set_window_by_sequence(0, 0);
// de_open_project("C:\\awad\\Research\\Hantenna_prj");
// de_close_all();
// de_open_window(2);
// de_set_window_by_sequence(2, 0);
// de_open_design("Hantenna.dsn", FALSE);
// /*END*/
//////////////////////////////////////////////////////////////////////

int Momentum::WriteAdsMacroFile()
{
	CString McroFileName;
	McroFileName = m_sProjectName+"\\"+"macro.dem";
	ofstream Macro(McroFileName);
	//// insert '/' after each '/' carachter for the
	/// macro file syntax
	CString temp_string=m_sProjectName;
	int i=0,j=0;
	while(TRUE)
		{
			j=temp_string.Find("\\",i);
			if(j==-1) break;
			temp_string.Insert(j,"\\");
			i=j+2;
		}

	Macro << "de_open_project(\"" + temp_string + "\");\n";
	Macro << "de_close_all();\n";
	Macro << "de_open_window(2);\n";
	Macro << "de_set_window_by_sequence(2, 0);\n";
	Macro << "de_open_design(\"" + m_sDesignName + "\", FALSE);";
	return 1;
}
```

```
//////////////////////////////////////////////////////////////////////
// run hpads and play the macro
// file for the current project
//////////////////////////////////////////////////////////////////////

void Momentum::RunAds(CWnd*& hpads_Window)

{

	// check if the design exists
	if(DesignNotFound())
	{
		m_iErrorFlag=DSN_NOTFOUND;
		return;
	}
	char hpads[]="hpads.exe"; // hpads exe file name
	int hpads_run_error=1;
	hpads_run_error=WinExec(hpads,SW_SHOW); //SW_SHOW//SW_MINIMIZE//SW_HIDE
	Sleep(2000);
	if(hpads_run_error<=31)			// WinExec return <=31 if an error happens
						// see the help for WinExec
	{
		m_iErrorFlag=CANNOT_RUN_ADS;
		return;
	}
	//// look for the hpads main window
	hpads_Window=NULL;
	char WindTitle[]="Advanced Design System (Main)";
	char WindClass[]="API_Main_75";
	if(FindWindow(WindClass, WindTitle,hpads_Window,120)) // if the window not found
	// in 120 Sec report an error
	{
		m_iErrorFlag=HPADS_WND_NOTFOUND;
		return;
	}
m_iErrorFlag;
	return;
}

//////////////////////////////////////////////////////////////////////
// close hpads
//////////////////////////////////////////////////////////////////////

void Momentum::CloseAds(CWnd *WndPtr)
{
	int FileSubMenu=0;
	int ExitIndex=16;
	int CloseDesignIndex=12;
	// close all designs
	PressSubMenuItem(WndPtr,&FileSubMenu,1,CloseDesignIndex);
	// close the save window Message Box
	CWnd *SaveWnd=NULL;
	FindWindow("#32770", "Save Changes", SaveWnd, 60); // find the Save window
	if(SaveWnd) SaveWnd->PostMessage(WM_COMMAND,0x00000007);
	// exit from ads
	PressSubMenuItem(WndPtr,&FileSubMenu,1,ExitIndex);
	// close the confirmation window
	Sleep(5000);
	CWnd *Conf_Wnd=NULL;
	FindWindow("#32770", "Confirmation", Conf_Wnd, 60); // find the exit
	// confirmation window
	if(Conf_Wnd)
	Conf_Wnd->SendMessage(WM_COMMAND,0x00000007);
	return ;
}
```

```
//////////////////////////////////////////////////////////////////////
// this function sets the values of the
// design parameters in
//////////////////////////////////////////////////////////////////////

void Momentum::SetParametersWindow()
{
	if(m_iErrorFlag) return; // return if an error happens
	// find the design layout window and open the design file
	CWnd * LayOutWindow;
CString LayoutWndCaption;
	SetLayoutWndCaption(NULL,LayoutWndCaption);  // set the current layout caption
	FindWindow("API_Layout_75", (char*)(LPCTSTR) LayoutWndCaption,
			LayOutWindow, 30);
	/// activate the parameters item in the optimization menu
	int SubMenuIndex[2]= {10,5}; // the indexes of the Momentum and Optimization submenues
	int ParametersItem=0;	// the index of the Parameters Item
	PressSubMenuItem(LayOutWindow, SubMenuIndex, 2,ParametersItem);
	/// find the parameters window
	CWnd *ParametersWnd=NULL;
	CString tempstr;
	tempstr.Format("%d",m_iLayoutNumber);
	CString WndTitle="Optimization Parameters:"+
				tempstr; // add the layout number to window title
	FindWindow("#32770",(char*)LPCTSTR(WndTitle),
			ParametersWnd,60); // find the layout window
	/// set the values of the design parameters
	CWnd* StarValueEditBox=NULL;
	CWnd *ParamList=NULL;
	CString ParamValue; // a string to keep the current parameter value
	ParamList=ParametersWnd->GetDlgItem( 0x00000006 ); // get the Parameters window pointer
	SetPoint(m_iNoParameters, (m_pAllParameters+(m_iCurrentPoint-1)*
			m_iNoParameters));
	for(int i=0;i<m_iNoParameters;i++)
	{
	ParamValue.Format("%f",m_pParameters[m_iNoParameters-i-1]);
		FillEditBox(ParametersWnd, ParamValue, 0x00000010);
		ParametersWnd->PostMessage( WM_COMMAND, 0x0000001D);  // press the update button
		ParamList->SendMessage(WM_SETFOCUS,NULL,NULL);  // set focus on the Parameters window
	ParamList->PostMessage(WM_KEYDOWN,VK_UP,0x000009); // Press the up arrow key
	}
	PressBotton(ParametersWnd, 0x0000001F);
	// set the current layout number
	m_iLayoutNumber+=m_iNoParameters+1;
	// close the MomOPTpipe window
	CWnd *MomOPTpipeWnd=NULL;
	FindWindow("API_eestatus_75", NULL, MomOPTpipeWnd, 5); // find the layout window
	int WindoSubMenu=3;
	if(MomOPTpipeWnd)
		PressSubMenuItem(MomOPTpipeWnd,&WindoSubMenu,1,1); // close  this window
	// close the last layout window
	int FileSubMenu=0;
	PressSubMenuItem(LayOutWindow,&FileSubMenu,1,22); // close  this last layout window
	return;
}

//////////////////////////////////////////////////////////////////////
// this tested this class during the development
//////////////////////////////////////////////////////////////////////

void Momentum::test()
{
	if(m_iHelpFlag)
	{
		CallHelp();
		return;
	}
	ReadInput();
	// Set the design information
	//CString PathName="C:\\awad\\Research\\MSL_prj";
```

```
		//CString DesignName="line.dsn";

//CString PathName="C:\\awad\\Research\\Double_folded_stub_filter_prj";
//CString DesignName="dfstub.dsn";
		//CString PathName="C:\\awad\\Research\\CPW_resonator_prj";
		//CString DesignName="resonator.dsn";
		//CString PathName="C:\\awad\\Research\\Microstrip_resonator_prj";
		//CString DesignName="resonator.dsn";
		//CString PathName="C:\\awad\\Research\\SingleStub_prj";
		//CString DesignName="filter.dsn";

//		m_sProjectName= PathName;
//  m_sDesignName = DesignName;
		///// Set the number of parameters
		//m_iNoParameters=3;
		//m_iNofPoints=1;
		//double Parameters[9]={90.0, 90.0, 5.0, 80.0, 80.0,6.0, 70.0, 70.0, 5.5 };
		//SetPoint(m_iNoParameters, Parameters);

		//SetAllParameters(m_iNofPoints,m_iNoParameters,Parameters);
		//SetNoPorts(2);
		///// Set the frequency bands
		//int NoFrequencyBands=2;
		//int NoFrequencies=4;
		//sMatrix *pFrequencyBands=new sMatrix(mDOUBLE,NoFrequencyBands,3);
		//double Start=5,Stop=10,Step=5;
		//pFrequencyBands->Populate(&Start,1,1);
		//pFrequencyBands->Populate(&Stop,1,2);
		//pFrequencyBands->Populate(&Step,1,3);
		//Start=15;Stop=20;Step=5;
		//pFrequencyBands->Populate(&Start,2,1);
		//pFrequencyBands->Populate(&Stop,2,2);
//pFrequencyBands->Populate(&Step,2,3);
		//SetFrequencies(NoFrequencyBands,NoFrequencies, pFrequencyBands);
		Simulate();
		//delete [] pFrequencyBands;
}

//////////////////////////////////////////////////////////////////////
// find an opend window
// keep looking for this window for
// DurationinSec time then exit with value
// 0 if the window is found or 1 otherwise
//////////////////////////////////////////////////////////////////////

int Momentum::FindWindow(char *WindClass, char *WindTitle,
                                 CWnd *&WndPtr,   double DurationInSec)
{
		double Max_Count=0;
		WndPtr=NULL;
		while(!WndPtr && Max_Count<DurationInSec)
		{
                                WndPtr=WndPtr->FindWindow(WindClass,WindTitle);
                                Sleep(100);  // wait 0.01 sec untill the window appears
                                Max_Count+=0.1;
		}
		if(Max_Count>=DurationInSec)  // if the window doesn't appear in 2 minuits return an error
                        return 1;
		else
                        return 0;
}

//////////////////////////////////////////////////////////////////////
// open the ads project
//////////////////////////////////////////////////////////////////////

void Momentum::OpenAdsPrj(CWnd *AdsWnd)
{
		if(m_iErrorFlag)	return;   // return if an error happens
		int OptionSubMenu=2;
```

```
	int PlayMacroIndex=6;
	PressSubMenuItem(AdsWnd,&OptionSubMenu,1,PlayMacroIndex); // activate the PlayMacro item
	// find the Run Macro window
	CWnd * RunMacroWnd=NULL;
	FindWindow("#32770", "Run Macro", RunMacroWnd, 60);
	// write the Macro file name and path in the edit box of the
	// Run Macro window
	CString MacroFileName;
	MacroFileName=m_sProjectName+"\\macro.dem"; // set up the Macro file name
	Sleep(1000);
	FillEditBox(RunMacroWnd, MacroFileName, 0x00000480); // write the Macro file name in the edit box
	PressBotton(RunMacroWnd, 0x00000001);// press the open button
	SetIniLayoutNumber();		// set the initial layout number for later use
	return ;
}

//////////////////////////////////////////////////////////////////////
// this function activates an item in
// a sub menu
//////////////////////////////////////////////////////////////////////

void Momentum::PressSubMenuItem(CWnd *WndPtr, int *SubMenuIndex,
					int NoSubMenues,int ItemIndex)
{
	CMenu *MenuPtr[10]={NULL,NULL,NULL,NULL,NULL,
				NULL,NULL,NULL,NULL,NULL};
	WndPtr->SendMessage(WM_ACTIVATE,(WPARAM)WA_ACTIVE,NULL);
	MenuPtr[0]=WndPtr->GetMenu();
	for(int i=1; i<=NoSubMenues; i++)
	MenuPtr[i]=MenuPtr[i-1]->GetSubMenu(SubMenuIndex[i-1]);
	UINT ItemID=MenuPtr[i-1]->GetMenuItemID(ItemIndex);
	WndPtr->SendMessage(WM_SETFOCUS,NULL,NULL); // set focus on the menu item
	WndPtr->PostMessage( WM_COMMAND, ItemID);
	Sleep(1000);
}

//////////////////////////////////////////////////////////////////////
// return 1 if the program cannot open
// the layout design otherwise return 0
//////////////////////////////////////////////////////////////////////

int Momentum::CannotOpenDesign()
{

	CWnd *ErrorWnd;
	char WndTitle[6][8]={"Error:1","Error:2","Error:3",
				"Error:4","Error:5","Error:6"};// possible Error window titles
	for(int i=0;i<6;i++)
	{
		if(!FindWindow("#32770", WndTitle[i], ErrorWnd, 1))
		{
			PressBotton(ErrorWnd,0x00000007);
			return 1;
		}
	}
	return 0;
}

//////////////////////////////////////////////////////////////////////
// set the layout number which is used to
// open the parameter and the simulator
// windows
//////////////////////////////////////////////////////////////////////

void Momentum::SetIniLayoutNumber()
{
	CWnd * LayoutWnd;
	if(FindWindow("API_Layout_75", NULL, LayoutWnd, 60)) // find the layout window
	{
		m_iErrorFlag=LAYOUTWND_NOT_FOUND;
```

```
                return;
        }
        CString LayoutWndTitle;
        LayoutWnd->GetWindowText(LayoutWndTitle);
        int length=LayoutWndTitle.GetLength();
        int i=LayoutWndTitle.Find(':'); // the layout number is imediately after the
        CString tempstr=LayoutWndTitle.Right(length-(i+1));// a string to keep this number
        m_iLayoutNumber= atoi( (LPCTSTR) tempstr);  // convert the layout number from a string to integer
}

//////////////////////////////////////////////////////////////////
// this function fills in the frequencies
// and launch the simulator
//////////////////////////////////////////////////////////////////

void Momentum::SetSimulationWindow()
{
        if(m_iErrorFlag)        return;   // return if an error happens
        // find the current layout window
        CWnd * LayoutWindow;
        CString LayoutWndCaption;
        SetLayoutWndCaption("_work",LayoutWndCaption);  // set the current layout caption
        FindWindow("API_Layout_75", (char *) (LPCSTR)LayoutWndCaption,
                        LayoutWindow, 60);
        /// activate the Simulate Circuit in the Simulation sub menu
        int SubMenuIndex[2]= {10,4}; // the index of Simulation submenue
        int SimCircuitItem=0;       // the index of the Simulate Circuit Item
        PressSubMenuItem(LayoutWindow, SubMenuIndex, 2,
                                SimCircuitItem);
        /// find the Simulation Control window
        CWnd *SimulationWnd=NULL;
        CString tempstr;
        tempstr.Format("%d",m_iLayoutNumber);
        CString WndTitle="Simulation Control:"+ tempstr; // add the layout number to the window title
        FindWindow("#32770",(char*)LPCTSTR(WndTitle),
                        SimulationWnd,60); // find the layout window
        ///////Set the Sweep type list box to Linear
        CWnd *SweepComboBox;
        SweepComboBox=SimulationWnd->GetDlgItem( 0x0000000E ); // get the Sweep window
        SweepComboBox->SendMessage(WM_SETFOCUS,NULL,NULL);  // set focus on the Parameters  window
        SweepComboBox->PostMessage(WM_KEYDOWN,VK_DOWN,0x00000002); // Press the down arrow key
        SweepComboBox->PostMessage(WM_KEYDOWN,VK_DOWN,0x00000002); // Press the down key again
                                                                        // to select the Linear Sweep

        /// cut off previous frequencies
        while(SimulationWnd->GetDlgItem(0x0000000A)->IsWindowEnabled( ))
        PressBotton(SimulationWnd,0x0000000A);
        ////// Fill in the frequency bands
        double Start,Stop,Step;
        CString Value;
        for(int i=1;i<=m_iNoFrequencyBands;i++)
        {
                m_pFrequencyBands->GetElement(&Start,i,1);//Get band start
                m_pFrequencyBands->GetElement(&Stop,i,2);//Get band stop
                m_pFrequencyBands->GetElement(&Step,i,3);//Get band step
                // Fill in the Start edit box
                Value.Format("%f",Start); // convert double to a string
                FillEditBox(SimulationWnd, Value, 0x00000010);
                // Fill in the Stop edit box
                Value.Format("%f",Stop); // convert double to a string
                FillEditBox(SimulationWnd, Value, 0x00000012);
                // Fill in the Step edit box
                Value.Format("%f",Step); // convert double to a string
                FillEditBox(SimulationWnd, Value, 0x00000014);
                // press the Add to the frequency Plan List button
                PressBotton(SimulationWnd,0x00000016);
        }
        // uncheck the data display option
        UncheckDlgButton(SimulationWnd,0x00000025);
        // uncheck  the reuse file from previous simulation wnd
```

```
		UncheckDlgButton(SimulationWnd,0x0000001E);
		// write the data set file name
		CString DataSetFileName="simdata";
		FillEditBox(SimulationWnd, DataSetFileName,0x00000021);
		//SimulationWnd->PostMessage( WM_COMMAND, 0x0000002D); // keep this comment
		CString DataFileName;
		DataFileName=m_sProjectName+"\\data\\simdata.ds";
		DeleteFile( (LPCTSTR) DataFileName);
		PressBotton(SimulationWnd, 0x0000002C); // press the Simulate button
		WaitForSimulation(); // wait till the simulator is done
		// delete the status window
		CWnd *StatusWnd;
		FindWindow("API_eestatus_75",NULL,StatusWnd,60); // find the layout window
		int StatusSubMenuIndex= 3; // the index of Simulation submenue
		if(StatusWnd) PressSubMenuItem(StatusWnd, &StatusSubMenuIndex, 1,1);
		// save the current results in the file output.dat
		SaveResults(LayoutWindow);
		// reopen the design file in case of multiple points simulation
		m_iCurrentPoint++;
		if(m_iCurrentPoint<=m_iNofPoints)
			OpenDesignFile(LayoutWindow);
}

//////////////////////////////////////////////////////////////////////
// this function open the design
// file which is used to generate
// new structure
//////////////////////////////////////////////////////////////////////

void Momentum::OpenDesignFile(CWnd *LayoutWindow)
{
		// press the Open item in the layout window
		int FileSubMenu=0;
		PressSubMenuItem(LayoutWindow,&FileSubMenu,1,1);
		// find the Open window caption
		CWnd * OpenFileWindow;
		CString tempstr;
		tempstr.Format("%d",m_iLayoutNumber);
		CString OpenWndCaption="Open Design:"+tempstr;
		FindWindow("#32770", (char*)(LPCTSTR)OpenWndCaption,OpenFileWindow , 10);
		OpenFileWindow->SendMessage(WM_ACTIVATE,(WPARAM)WA_ACTIVE,NULL);
		// set the design file name in the Open window
		CWnd* FilterEditBox=NULL;
		FilterEditBox=OpenFileWindow->GetDlgItem( 0x00000005 ); // get a pointer to filter edit box pointer
		CString FileName=m_sProjectName+"\\networks\\"+m_sDesignName;
		CWnd *TempWnd;
		FillEditBox(OpenFileWindow, FileName, 0x00000005);// write the design file name in the filter edit box
		// press the Filter button
		PressBotton(OpenFileWindow, 0x0000000C);
		FilterEditBox->SendMessage(WM_KILLFOCUS,NULL,NULL); // kill focus on the filter edit box
		// set focous to the list box
		TempWnd=OpenFileWindow->GetDlgItem(0x00000008); // a pointer to the list box
		TempWnd->SendMessage(WM_SETFOCUS,NULL,NULL);
		// press the OK button
		OpenFileWindow->SendMessage( WM_COMMAND, 0x0000000B);
		return;
}

//////////////////////////////////////////////////////////////////////
// set the layout window caption in order
// to find this windows when it appears
//////////////////////////////////////////////////////////////////////

void Momentum::SetLayoutWndCaption(char *needed, CString &LayoutWndCaption)
{
		CString tempstr1,tempstr2, tempstr3;
		int length=m_sProjectName.GetLength();
		int PrjNameIndex=m_sProjectName.ReverseFind('\\');// find a 0 based index of the project name in the project full path
		tempstr1=m_sProjectName.Right(length-PrjNameIndex-1); // this string contains the  project name
		tempstr2=m_sDesignName.Left(m_sDesignName.Find('.')); // this string contains the contains the design file name without
```

```
//an extintion
	tempstr3.Format("%d",m_iLayoutNumber);		// convert the layout number to a string
	LayoutWndCaption="[ "+ tempstr1+" ] "+ tempstr2+ needed +
						" (Layout):"+tempstr3;
}

//////////////////////////////////////////////////////////////////
// for for the current simulation
// to finish
//////////////////////////////////////////////////////////////////

void Momentum::WaitForSimulation()
{

	CString DataFileName;
	DataFileName=m_sProjectName+"\\data\\simdata.ds";

	FILE * DataFile=NULL;

	while(!DataFile)
		DataFile=fopen( (const char*)(LPCTSTR)DataFileName, "r" );
	fclose(DataFile);
}

//////////////////////////////////////////////////////////////////
// extract and save the simulated results in
// the ascii file output.dat
//////////////////////////////////////////////////////////////////

void Momentum::SaveResults(CWnd *LayoutWindow)
{
	CString DataFileName;
	DataFileName=m_sProjectName+"\\data\\output.s2p"; // set up the Macro file name
	DeleteFile( (LPCTSTR) DataFileName);
	/// activate the Visualization in the Post Processing  sub menu
	int SubMenuIndex[2]= {10,6}; // the index of Post Processing submenue
	int VisualItem=1;	// the index of the Visualization Item
	PressSubMenuItem(LayoutWindow, SubMenuIndex, 2, VisualItem);
	// Find the viualization window
	CWnd * VisualWnd;
	CString WindowCaption, tempstr;
	tempstr=m_sDesignName.Left(m_sDesignName.Find('.'));
	WindowCaption="HP MOMENTUM (" + tempstr +  "_work)";
	//WindowCaption="HP MOMENTUM (" + tempstr +  ")"; // temporarly statement
	FindWindow(NULL, (char*)(LPCTSTR)WindowCaption,VisualWnd, 60);
	// Press the Export S Matrix in the File submenu
	int FileMenuIndex= 0;    // the index of File submenue
	int ExportSMatrixItem=2; // the index of the Export S Matrix
	Sleep(10000);
	PressSubMenuItem(VisualWnd, &FileMenuIndex, 1, ExportSMatrixItem);
	// write the data file name in the Export Matrix Dialog
	CWnd * ExportWnd, *FlNameEditBox, *OutputTypeWnd,*TouchDStoneWnd ,*OKWnd;
	ExportWnd=FindWindowExt("??????8???",VisualWnd);
	// choose touchstone option
	OutputTypeWnd=ExportWnd->GetWindow( GW_CHILD)->GetNextWindow(GW_HWNDNEXT)
					->GetWindow( GW_CHILD)->GetWindow( GW_CHILD)
					->GetWindow( GW_HWNDLAST)->GetNextWindow(GW_HWNDPREV)
					->GetWindow( GW_CHILD)->GetWindow( GW_CHILD)->GetWindow(
GW_CHILD)
					->GetNextWindow(GW_HWNDNEXT);
	OutputTypeWnd->SendMessage(WM_ACTIVATE,(WPARAM)WA_ACTIVE,NULL);
	TouchDStoneWnd=OutputTypeWnd->GetDlgItem(0x00000065);
	TouchDStoneWnd->SendMessage(WM_SETFOCUS,NULL,NULL);
	OutputTypeWnd->CheckRadioButton( 0x00000064, 0x00000065,0x00000065 );
	/// write the output file name in the file edit box
	FlNameEditBox=ExportWnd->GetWindow( GW_CHILD)->GetNextWindow(GW_HWNDNEXT)
					->GetWindow( GW_CHILD)->GetWindow( GW_CHILD)->GetWindow(
	GW_HWNDLAST)   ;
	DataFileName=m_sProjectName+"\\data\\simdata.s2p"; // set up the Macro file name
	FillEditBox(FlNameEditBox, DataFileName, 0x00000000); // write the output file name in the edit box
```

```
	DeleteFile( (LPCTSTR) DataFileName);
	// press the OK button
	OKWnd=ExportWnd->GetWindow( GW_CHILD)->GetNextWindow(GW_HWNDNEXT)
		->GetWindow( GW_CHILD)->GetNextWindow(GW_HWNDNEXT)->GetWindow( GW_CHILD);
	OKWnd->SendMessage(WM_SETFOCUS,NULL,NULL);
	OKWnd->SendMessage(WM_COMMAND, 0x00000BB9);
	////// close the Yes/No dialog	box
	PressSubMenuItem(VisualWnd, &FileMenuIndex, 1, 5);
	CWnd *YesNoDialogBox;
	YesNoDialogBox=FindWindowExt("??????????",VisualWnd);
	YesNoDialogBox->SendMessage(WM_COMMAND, 0x00000001);
	//if(m_iCurrentPoint==1)
	WriteOutputFile();
}

//////////////////////////////////////////////////////////////////////
// find a child window with
// a specific class name and captions
//////////////////////////////////////////////////////////////////////

CWnd * Momentum::FindChildWindow(CWnd *ParentWnd, const char *ClassName,
                                                                const char *WndCaption)
{
	HWND WndHandel;
	WndHandel=FindWindowEx((HWND) ParentWnd,NULL, ClassName,WndCaption);
	return (CWnd *) WndHandel;
}

//////////////////////////////////////////////////////////////////////
// find not a top level window with a specific
// class name and caption name
// note the the ordinary FindWindow function
// searches only for a top level windows not
// a child windows
//////////////////////////////////////////////////////////////////////

CWnd * Momentum::FindWindowExt(const char * CaptionName, CWnd *StartWnd)
{
	CWnd *oldwnd, *newwnd;
	CString  wndcaption;
	oldwnd=StartWnd;
	do
	{
		newwnd=oldwnd->GetNextWindow(GW_HWNDNEXT);
		if (newwnd==NULL) break;
		newwnd->GetWindowText(wndcaption);
		oldwnd=newwnd;
	}
	while ((wndcaption !=CaptionName));
	if (newwnd!=NULL) return newwnd;
	oldwnd=StartWnd;
	do
	{
		newwnd=oldwnd->GetNextWindow(GW_HWNDPREV);
		if (newwnd==NULL) break;
		newwnd->GetWindowText(wndcaption);
		oldwnd=newwnd;
	}

	while ((wndcaption !=CaptionName) );
	return newwnd;
}

//////////////////////////////////////////////////////////////////////
// write in a specified edit
// box
//////////////////////////////////////////////////////////////////////
void Momentum::FillEditBox(CWnd *ParentWnd, CString EditBoxStr, int EditBoxID)
{
	CWnd* EditBoxWnd=NULL;
```

```
	ParentWnd->SendMessage(WM_ACTIVATE,(WPARAM)WA_ACTIVE,NULL); // actvate the parent window
	while(!ParentWnd->IsWindowEnabled( ));
	EditBoxWnd=ParentWnd->GetDlgItem( EditBoxID ); // get a pointer to the edit box
	EditBoxWnd->SendMessage(WM_SETFOCUS,NULL,NULL); // set focus on the edit box
	EditBoxWnd->SendMessage(WM_SETTEXT,0,(LPARAM) // write in the filter edit box
				(LPCTSTR)EditBoxStr);
	EditBoxWnd->SendMessage(WM_KILLFOCUS,NULL,NULL); // kill focus on the edit box
}

//////////////////////////////////////////////////////////////////////
// press a button in adialog box
//////////////////////////////////////////////////////////////////////

void Momentum::PressBotton(CWnd *ParentWnd, int ButtonID)
{
	CWnd* ButtonWnd=NULL;
	ParentWnd->SendMessage(WM_ACTIVATE,(WPARAM)WA_ACTIVE,NULL); // actvate the parent window
	while(!ParentWnd->IsWindowEnabled( ));
	ButtonWnd=ParentWnd->GetDlgItem(ButtonID ); // get a pointer to the
	ButtonWnd->SendMessage(WM_SETFOCUS,NULL,NULL); // set focus on the edit box
	ParentWnd->SendMessage(WM_COMMAND,ButtonID);    // press the button
}

//////////////////////////////////////////////////////////////////////
// this function unchechs
// a dialog button
//////////////////////////////////////////////////////////////////////

void Momentum::UncheckDlgButton(CWnd *ParentWnd, int ButtonID)
{
	CWnd *DialogButtonWnd;
	DialogButtonWnd=ParentWnd->GetDlgItem(ButtonID);
	DialogButtonWnd->SendMessage(BM_SETCHECK,(WPARAM) BST_UNCHECKED,0);
	DialogButtonWnd->SendMessage(WM_SETFOCUS,NULL,NULL);
	ParentWnd->SendMessage(WM_COMMAND,ButtonID);
}

//////////////////////////////////////////////////////////////////////
// extract the S-Parameters from momentum
// output file simdata.s2p and write them
// in an ascii output file in the format
// rs11 is11 rs12 is12 rs21 is21 rs22 is22
// *   *   *    *   *   *   *   *
//////////////////////////////////////////////////////////////////////

void Momentum::WriteOutputFile()
{
	CString MomDataFileName; // to hold the file name generated by Momentum
	GetMomDataFileName(MomDataFileName);
	if(m_iCurrentPoint==1)
		DeleteFile((LPCTSTR) m_sOutputFileName);   // delete the output file in writing data to it
	ifstream MomDataFile(MomDataFileName);
	ofstream OutputFile(m_sOutputFileName,ios::out|ios::app);
	if(OutputFile.bad())
	{
		m_iErrorFlag=CANNOT_CREATE_OUTPUTFILE;
		return;
	}
	// fined the "!END EXCITATION_MAPS" string in the MomDataFile
	char strbuff[1000];
	CString tempstr;
	MomDataFile.getline(strbuff,1000);
	tempstr=(const char *) strbuff;
	while (tempstr!="!END EXCITATION_MAPS")
	{
		MomDataFile.getline(strbuff,1000);
		tempstr=(const char *) strbuff;
	}
	// write S parameters in the output file
	double Value;
```

```
	int NofSParameters,i,j;
	NofSParameters=m_iNoPorts*m_iNoPorts*2; // number of real and imaginary S parameters
	for(i=0;i<m_iNoFrequencies;i++)
	{
		MomDataFile >> Value ; // just ignore the frequency
		// start writing S parameters
		for (j=0; j<NofSParameters;j++)
		{
			MomDataFile >> Value;
			OutputFile  << Value;
			OutputFile  << " ";
		}
		OutputFile  << "\n";
	}
	// close all files
	MomDataFile.close();
	OutputFile.close();
}

//////////////////////////////////////////////////////////////////////
// check if the nominal file not found
//////////////////////////////////////////////////////////////////////

int Momentum::DesignNotFound()
{
	CString DesignFileName;
	DesignFileName=m_sProjectName+"\\networks\\"+m_sDesignName;
	FILE * DesignFile;
	DesignFile=fopen( (const char*)(LPCTSTR)DesignFileName, "r" );
	if(!DesignFile)
		return 1;
	else
		return 0;
}

//////////////////////////////////////////////////////////////////////
// set the number of points
//////////////////////////////////////////////////////////////////////

void Momentum::SetNofPoints(int NofPoints)
{
	m_iNofPoints=NofPoints;
}

//////////////////////////////////////////////////////////////////////
// set the parameters at all points
//////////////////////////////////////////////////////////////////////

void Momentum::SetAllParameters(int NofPoints, int NofParameters, double *Parameters)
{
	m_iNofPoints=NofPoints;
	m_iNoParameters=NofParameters;
	m_pAllParameters= new double[NofPoints*NofParameters];
	for (int i=0;i<(NofPoints*NofParameters);i++)
	m_pAllParameters[i]=Parameters[i];
	return;
}

//////////////////////////////////////////////////////////////////////
// set up the complete  Momentum
// data file name
//////////////////////////////////////////////////////////////////////

void Momentum::GetMomDataFileName(CString &MomDataFileName)
{
	CString PortNumber;
	PortNumber.Format("%i",m_iNoPorts);
	MomDataFileName=m_sProjectName+"\\data\\simdata.s"+PortNumber+"p";
}
```

```
//////////////////////////////////////////////////////////////////////
// read the input file which contains
// all the information about the
// project
//////////////////////////////////////////////////////////////////////
void Momentum::ReadInput()
{
	CString FullInputFileName; // the complete input file name
	char strbuff[1000];
	CString tempstr;
	// set up the complete input file name
	//FullInputFileName=m_sProjectName+"\\data\\"+ InputFileName;
	ifstream InputFile(m_sInputFileName,ios::nocreate |ios::in);
	// check if it cannot open the input file
	if(InputFile.fail())
	{
		m_iErrorFlag=CANNOT_OPEN_INPUTFILE;
		return;
	}
	// get the path name
	InputFile.eatwhite();
	InputFile.getline(strbuff,1000);
	tempstr=(const char *) strbuff;
	m_sProjectName=tempstr;
	// write the design file name
	InputFile.eatwhite();
	InputFile.getline(strbuff,1000);
	tempstr=(const char *) strbuff;
	m_sDesignName=tempstr;
	// get the number of points
	InputFile >> m_iNoParameters;
	// get the number of points
	InputFile >> m_iNofPoints;
	// read the paramtrs values
	double *Parameters= new double[m_iNofPoints*m_iNoParameters];
	for( int i=0; i<(m_iNofPoints*m_iNoParameters);i++)
		InputFile >> Parameters[i];
	SetAllParameters(m_iNofPoints,m_iNoParameters,Parameters);
	delete [] Parameters;
	// read number of ports
	InputFile >> m_iNoPorts;
	// read the number of the frequency bands
	int NoFrequencyBands;
	InputFile >> NoFrequencyBands;
	// read the frequency bands and set the number of frequencies
	int NoFrequencies=0;
	sMatrix FrequencyBands=sMatrix(mDOUBLE,NoFrequencyBands,3);
	double Start, Stop, Step;
	for (i=1;i<=NoFrequencyBands;i++)
	{
		InputFile >> Start;
		InputFile >> Stop;
		InputFile >> Step;
		FrequencyBands.Populate(&Start,i,1);
		FrequencyBands.Populate(&Stop,i,2);
		FrequencyBands.Populate(&Step,i,3);
		NoFrequencies+= (((Stop-Start)/Step)+1.01);
	}
	// set all frequencies
	SetFrequencies(NoFrequencyBands,NoFrequencies, &FrequencyBands);
}

//////////////////////////////////////////////////////////////////////
// call the help function
//////////////////////////////////////////////////////////////////////
void Momentum::CallHelp()
{
	MessageBox(NULL,"  MomentumDriver.exe  [-h] [input file name] [output file name]  ",
			"Momentum Driver Help",MB_OK); // display a help message box
}
```