Programming for Engineering and Science - Assignment 2

Rafael Pacheco
Seyed Mohammad Reza Attar Seyedi
Arturo Acosta

June 5, 2017

1 ASSIGNMENT 2

CONTENTS

1.1 INTRODUCTION

In the present assignment, a finite element code has been developed in order to solve the following problem for a 2D and 3D domain.

$$\begin{cases} \Delta u = 0 & in\Omega \\ \nabla u \cdot n = -1 & in\Gamma_{in} = 0 \times (0,1) \\ \nabla u \cdot n = 1 & in\Gamma_{out} = 1 \times (0,1) \\ \nabla u \cdot n = 0 & in\partial\Omega(\Gamma_{in} = 0 \cup \Gamma_{out}) \\ u(0,0) = 0 \end{cases} \tag{1.1}$$

This code should able to recreate the cases for 2D and 3D meshes given in class. Although with the correct data input, it can analyse similar problems.

1.2 METHODOLOGY

The "GUI" is a terminal windows where the user is able to select several things:

- Analysis type : 3D or 2D .
- Type of element: Triangle, Quadrilateral, Tetrahedra.
- Order of the element: Linear or Quad.
- Select a mesh: 1 - 5.
- Load mesh ".dat" files.
- Introduce the Boundary Conditions.

Some of them are shown after others. It is important to recall that the boundary conditions are introduced by the user, using some mathematical geometries (planes). So if it is needed the problem is able to perform similar analysis for the same domain with different Boundary Conditions.

1.2.1 INPUT FILES

The input files required are those given in class. The loads both files you have chosen and then starts reading and storing the data to perform the analysis. If the two files are not com-

patible, errors will occur. Again, recall that with different meshes from the ones given in class, the code is able to perform the analysis for a similar domain.

Basically the code checks whether or not the elements are inside the Boundary Conditions, so it is not fixed to the domain given in class. As long as the mesh provided matches with the Boundary Conditions introduced with the user, the code should be able to solve the problem.

1.2.2 MESH

The meshes given are refined from 1 to 5. They different types of elements and degrees. The elements types of elements implemented are:

- Triangles:

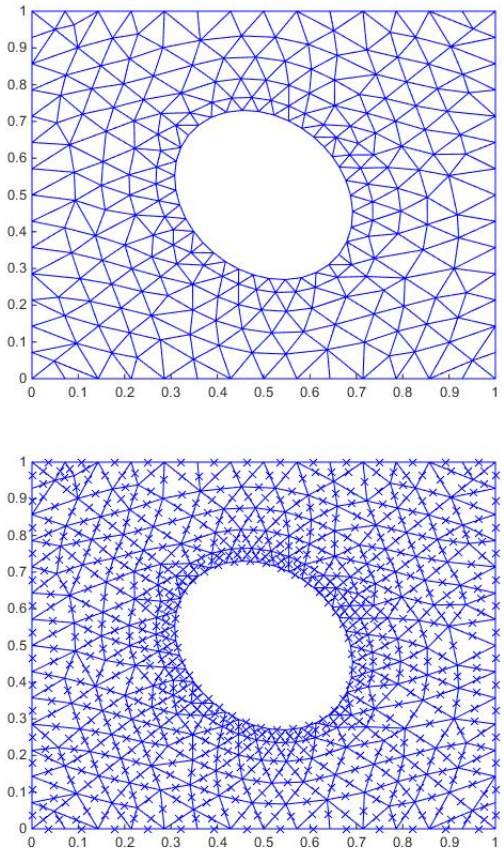

Figure 1.1: Linear triangles and Quad Triangles.

- Quadrilaterals:

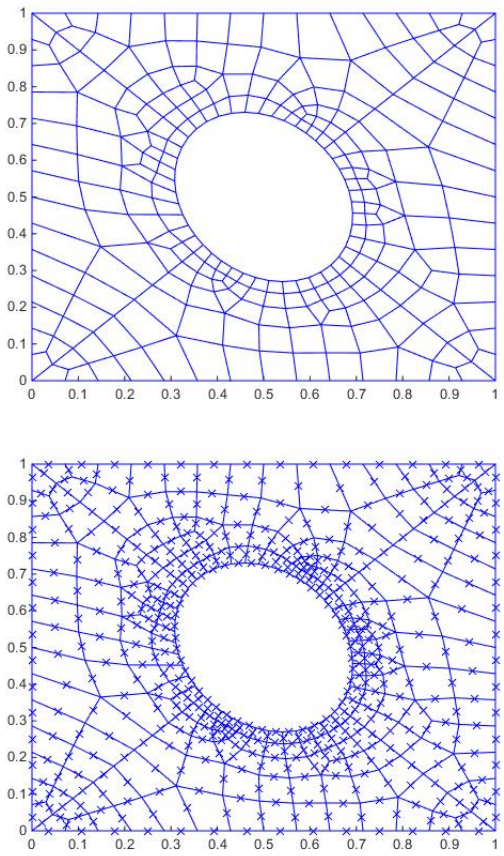

Figure 1.2: Linear triangles and Quad Quadrilaterals.

- Tetrahedra:

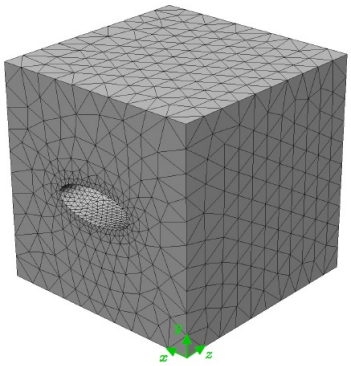

Figure 1.3: Linear Tetrahedra.

1.3 Boundary Conditions

As said previously boundary conditions are introduced by the user in the following manner:

- First definition of the boundary Equation:

$$\begin{array}{l} ax+by+cz=d \qquad\qquad ,[x_1,x_2,y_1,y_2,z_1,z_2] \\ \text{BC:} \\ \text{Neumann} \quad (q\cdot n=q_n) \\ \text{Dirichlet} \quad (u=u_D) \end{array} \tag{1.2}$$

- Then add more conditions until the user decides to stop.

1.4 Structure of the code

1 User selection of the case (Dimension, type, order)

2 User inputs the two files (coordinates and connectivity matrices) for the analysis.

3 User defines the boundary conditions.

4 The code starts:

- Creates the Global mesh.
- Creates local meshes for the boundary conditions.
- Creates the local axis for the boundary conditions.
- Creates Boundary elements to solve the r.h.s . This includes Neumann and Dirichlet reduction of the full system $-K(:,i)u(i)$.
- Creates Stiffness elements.
- Assembles Boundary elements on the r.h.s vector and stiffness elements on l.h.s matrix.
- Reduces the stiffness matrix to one without the Dirichlet terms.
- Uses *LU* decomposition to solve the system.

- Passes the nodal displacements to the postprocess, in order to write the postprocess file (vtk).

5 User should chose the output format of the results.

1.5 Design of the code

The code has several files:

1 GUI.cpp : File containing the core of the program and the prepost-processing.

2 Input.cpp / Input.h : Class created in order to handle the reading of the plain text files given in the class. Depending on the user analysis parameters, it stores the information differently(mainly the connectivity matrix, which varies on the element).

3 Mesh.cpp / Mesh.h : Class created in order to define the computational domain. This is the core for the analysis and computation. The whole program relies on this class. This class is as well a morphological class since it has a children class named "Boundary".

The main idea behind this, is that the mesh class, has the information of the domain, however, there is a subdivision of the domain for the boundaries, where the element type is no longer the same (e.g. the Neumann boundary conditions use a reduced dimension element type respect the stiffness).

4 Element.cpp / Element.h : Class created to handle the computation inside each element. There are different elements defined by the type of element and order and two more elements not requested in order to integrate the r.h.s term of 2D problems. These are 1D linear and Quad elements.

5 Matrix.cpp/Matrix.h : Class created in order to operate inside a matrix space. Some useful operators and special functions are defined.

6 vector.cpp/Matrix.h : Class created in order to operate inside a vector space. Again some useful operators and special functions are defined. This two last classes are linked with each other, being able to operate, at some certain degree, together.

It would be good to recall that Compiler Directives and enumeration were used to make the code more powerful.

1.5.1 PREPOST-PROCESSING

Gui.cpp is the file in which the user can introduce different settings for the analysis. It is the responsible to create the **mesh** object which will be the base for the analysis.

Some compiler entries are written in order to make it easier for the developer to skip the inputs. This is defined by the _DEBUG pragma. When the code is compiled on Release this flag is neglected and therefore the user is able to interact with the code again.

1.5.2 MESH

The mesh class is the trunk of the code. Its principal parameters are:

1. Matrices: X,T Xm (middle point), K (Global stiffness), F(global r.hs.), KU (dirichlet vector).

2. enum: dimension, type, order

3. element: elem (created to define the type of element of the mesh).

4. member functions (all public): Give_...(any of the matrices described above) and Set_...(some of the matrices above). Also one functoin to Create the Boundary Conditions (user input) and other to obtain the localaxis of a given element.

5 Class Boundary (children). It has inherit all the previous options and virtual overwritten the one to create boundary conditions.

If GUI.cpp created a object mesh to run the analysis, mesh acts as a manager to identify the Boundary Conditions and subdivide the domain for these and once it has collected and rearranged all the information (preconditioner). It starts looping through the elements and nodes to creating an element named **elem** which will perform the elemental assembly to later retrieve this information to mesh, which will assemble it with the global.

Observe that since boundary class is a children, it can perform the same routine as mesh but for its own subspace X_{BC} and T_{BC} being able to ask for boundary elements which are useful to obtain the r.h.s. terms without interfering with the main domain. So inside mesh, a boundary element **elem** will be again created to obtain the elemental r.h.s. nodal forces vector for Dirichlet and Neumann.

It is useful to do it this way, because the code is more independent and also with a large

number of B.C. , it is quicker to manage them this way and avoid to store many information in an object when this information can be realised.

1.5.3 ELEMENT

Element is the space where the operations are undertaken:

1 Matrices: N, $\frac{dN}{d\xi_i}$, [zgp,ω], nodesCoord, P (element real coordinates), J (jacobian) , invJ, det J, B_e, K_e, F_e .

2 member functions. Basically void functiosn to create the different matrices from above.

Element is useful to obtain the basic stiffness and nodal forces information and return it back to the main trunk (mesh). In the future some recycle design could be added to avoid creating and destroying this element, and optimise the computational speed then.

1.5.4 MATRIX AND VECTOR

Something probably different from the approach of this assignment, has been to create from zero the computational space. The difference between a matrix and a vector element reside on the fact that matrices are somehow organised vectors.

Huge part of the code, has been into developing this spaces in order to make the code lighter, optimal and unique. Many different operators were defined to be able to operate between matrices, vectors and matrices&vectors . Inverse operator, determinant operator and trace operator were created to avoid having to create instances or members in the main classes.

1.5.5 POSTPROCESSING

Again inside GUI.cpp once the calculations are done, the user is able to chose between .vtk and .m postprocess files.

The simplest way to create this module is to give a template to the software and it will modify according to some special keyword the parts of the output template. This was done by creating a function inside Input.h (which in here is an output operation). To read the template and store the matrices or vectors as the template requires.

1.6 RESULTS

Unluckily the 3D results are giving some errors which could not be debugged. On the other hand the vtk template is not working correctly, only leaving to the Matlab template.

Now for mesh cases 1, some results are shown. There is some error on the calculation of the r.h.s. due to the 1D linear and quad element. It is believed that it may come because of:

$$J = \frac{dNi}{d\xi_I} \cdot NodeCoord \cdot P(x_k) \tag{1.3}$$

Using the configuration of the element ($NodeCoord$) and the basic corner nodes ($P(x_k)$). The Jacobian can be obtained without having to create many different conditions. This is useful when using degeneration of elements, such as a serendipity quadrilateral of 8 nodes. Changing the $NodeCoord$ matrix, it will give back the new element.

And it is believed that the problem is coming as a result of this simplification.

On the other hand, the velocity now of the analysis is around 6 times the one it was on Matlab. Which is obvious because C++ is a compiled language and not a interpreter language. One beautiful upgrade would be to use OpenMP, for the stiffness assembling and calculus module. Which does not require any sort of order of calculation, this will make the code even much more quicker.

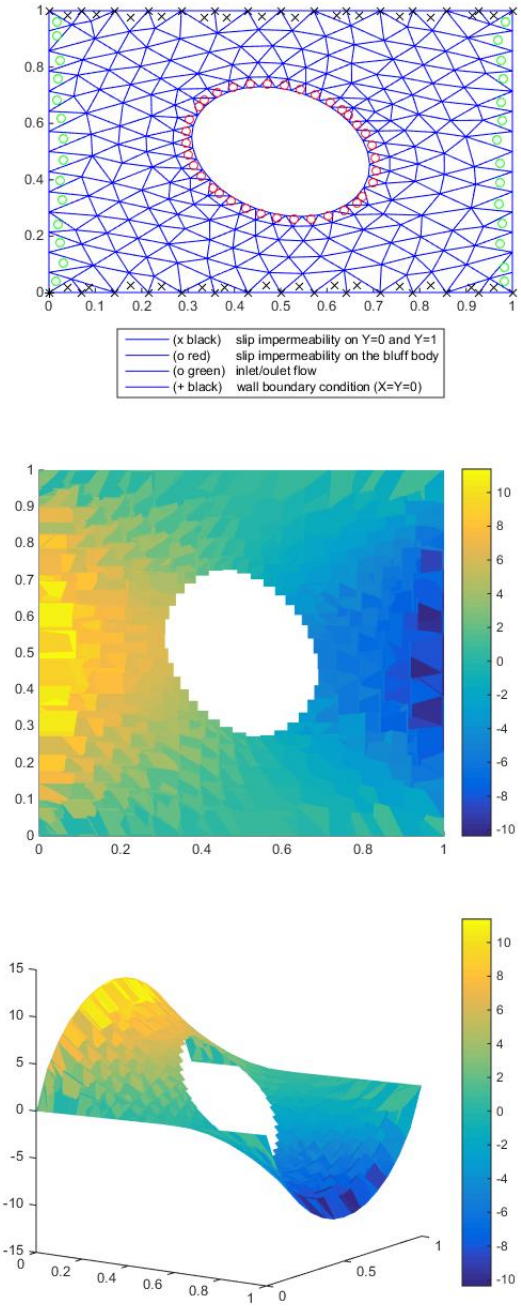

Figure 1.4: 2D and 3D Linear Quadrilaterals.

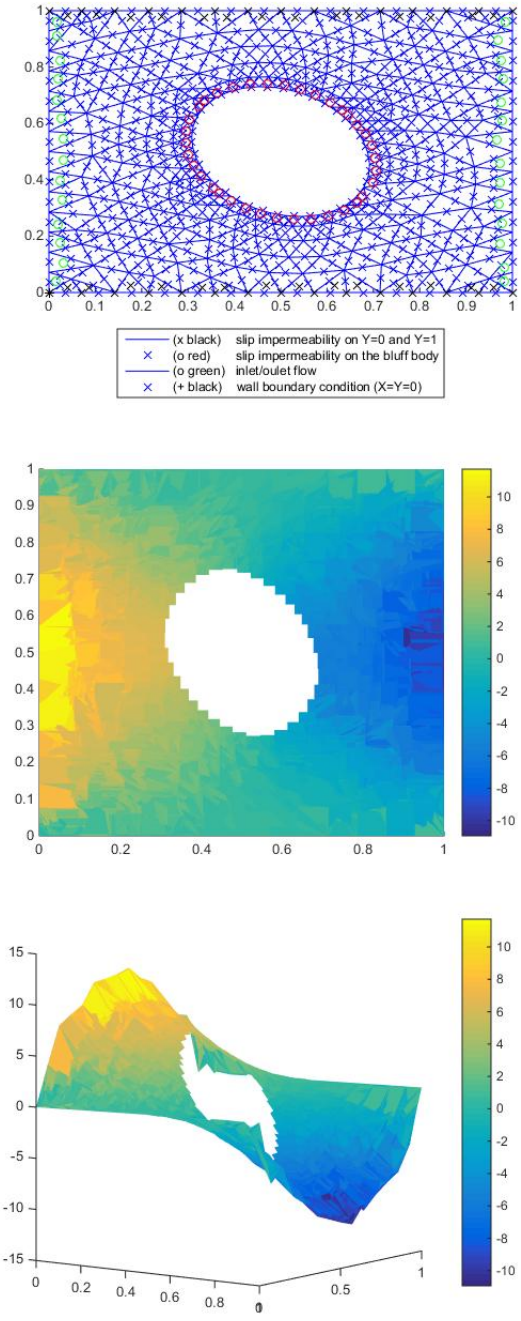

Figure 1.5: 2D and 3D Linear Quadrilaterals.

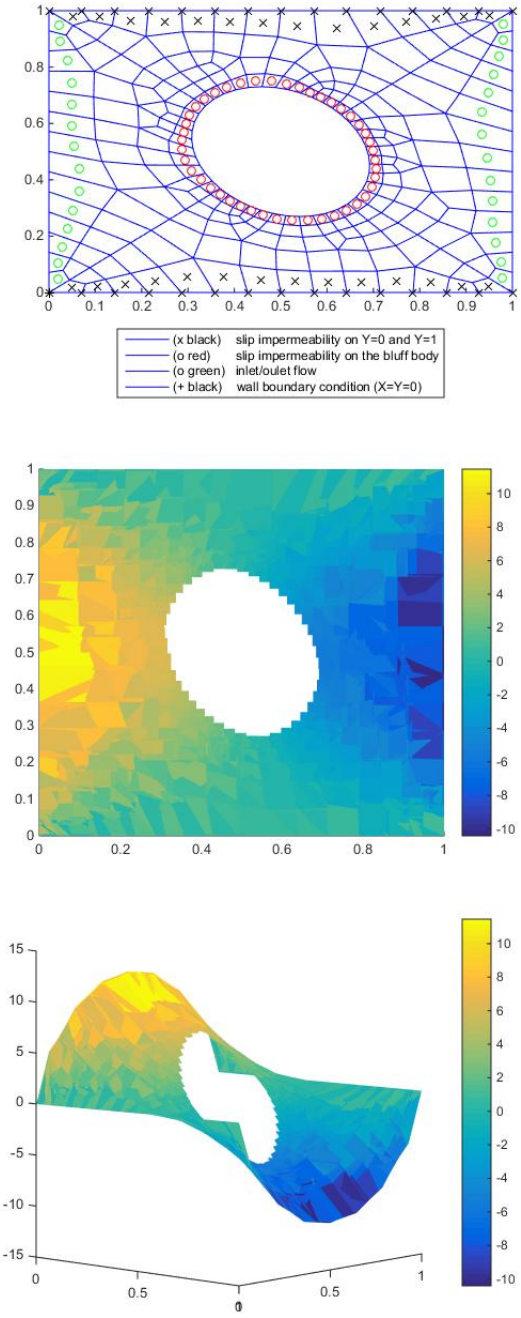

Figure 1.6: 2D and 3D Linear Quadrilaterals.

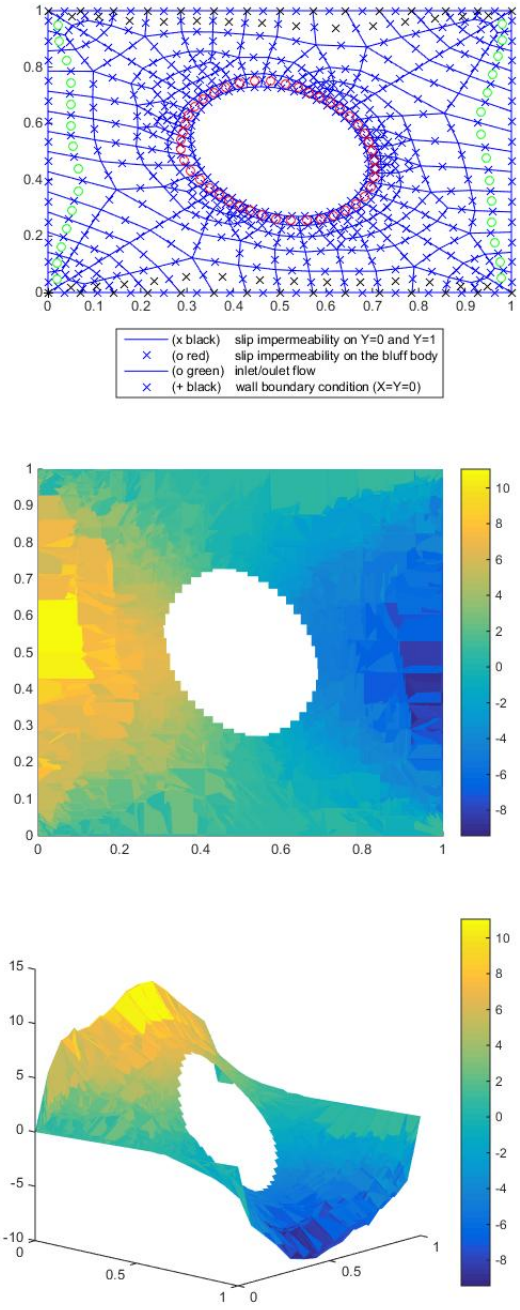

Figure 1.7: 2D and 3D Linear Quadrilaterals.

2 APPENDIX

```
1  #include "stdafx.h"
2  #include "vector.h"
3  #include "Matrix.h"
4  #include <iostream>
5  #include <math.h>
6  #include <fstream>
7  #include "Input.h"
8  #include "Mesh.h"
9
10 using namespace std;
11
12 int main()
13 {
14         enum Dimension{ Dim2 =  2 , Dim3 = 3 } dim;
15         enum Type{ Triangle = 3, Quadrilateral = 4, Tetrahedra } type;
16         enum Order{ Lin = 1, Quad = 2 } order;
17         int  opt(-1), mesh(0);
18         vector qn(3); qn(1) = 0; qn(2) = 0; qn(3) = 0;
19         string sdir="",sdir2="";
20         Input Dir1;
21 #ifndef _DEBUG
22         cout << "3D enabled (1) or 2D enabled (0): " << endl;
23         cin >> opt;
24         if (opt == 1)
25                 dim = Dim3;
26         else
27                 dim = Dim2;
28
29         cout << "Type: quadrilateral (1) or triangle (0): " << endl;
30         cin >> opt;
31         if (opt == 1)
32                 type = Quadrilateral;
33         else
34                 type = Triangle;
35
36         cout << "Order: Quad (1) or Lin (0): " << endl;
37         cin >> opt;
38         if (opt == 1)
39                 order = Quad;
40         else
41                 order = Lin;
```

```
42          cout << "Mesh: number from 1-5: " << endl;
43          cin >> mesh;
44          cout << "input the current components\n   Ux : ";
45          cin >> qn(1);
46          cout << "   Uy : ";
47          cin >> qn(2);
48          cout << "   Uz : ";
49          cin >> qn(3);
50          cout << "\nThe current is qn=[" << qn(1) << "," << qn(2) << ","
                << qn(3) << "]" << endl;
51
52  #else
53
54          dim = Dim2;
55          type = Quadrilateral;
56      mesh = 1;
57          qn(1) = 1; qn(2) = 0; qn(3) = 0;
58          if (true){
59                  if (dim == Dim2){
60                          sdir = "C:\\Users\\Rafael\\Dropbox\\Work\\UPC-
                                PES-HW1\\Mesh\\PES_2D_quad_lin\\Mesh1\\
                                Element_2D_quad_lin.dat";
61                          sdir2 = "C:\\Users\\Rafael\\Dropbox\\Work\\UPC-
                                PES-HW1\\Mesh\\PES_2D_quad_lin\\Mesh1\\
                                Node_2D_quad_lin.dat";
62                          order = Lin;
63                  }
64                  else{
65                          sdir = "C:\\Users\\Rafael\\Dropbox\\Work\\UPC-
                                PES-HW1\\Mesh\\PES_3D_tetra_lin\\
                                Element_3D_tetra_lin.dat";
66                          sdir2 = "C:\\Users\\Rafael\\Dropbox\\Work\\UPC-
                                PES-HW1\\Mesh\\PES_3D_tetra_lin\\
                                Node_3D_tetra_lin.dat";
67                          order = Lin;
68                  }
69          }
70          else{
71                  sdir = "C:\\Users\\Rafael\\Dropbox\\Work\\UPC-PES-HW1\\
                        Mesh\\PES_2D_quad_quad\\Mesh1\\Element_2D_quad_quad.
                        dat";
72                  sdir2 = "C:\\Users\\Rafael\\Dropbox\\Work\\UPC-PES-HW1\\
                        Mesh\\PES_2D_quad_quad\\Mesh1\\Node_2D_quad_quad.dat
                        ";
```

```
73                  order = Quad;
74          }
75
76  #endif
77
78          if (dim != Dim3 && dim != Dim2 || type != Quadrilateral && type
                != Triangle || order != Quad && order != Lin || mesh >5 &&
                mesh < 1)
79                  cout << "Error defining the parameters" << endl;
80          else
81          {
82                  cout << "\nAnalysis selected: " << endl;
83
84                  if (dim == Dim3 )
85                  cout << "3D" << endl;
86                  else
87                  cout << "2D" << endl;
88
89                  if (type == Quadrilateral)
90                          cout << "Quadrilateral" << endl;
91                  else
92                          cout << "Triangle" << endl;
93
94                  if (order == Quad)
95                          cout << "Order : Quad" << endl;
96                  else
97                          cout << "Order : Lin" << endl;
98
99                          cout << "Mesh : " << mesh <<endl;
100
101 #ifndef _DEBUG
102                 cout << "\nType the connectivity file you want to open (
                        e.g., C:/Users/.../Element_2D_quad_lin.dat) \n\n
                        Note:  1)the \"/\" character instead of \"/\"   \
103                         \n  Note : 2) If the file is not coherent with
                                the analysis type selected, errors will
                                occur\n" << endl;
104                 cin >> sdir;
105 #endif
106                 Matrix T(Dir1.Read(Dir1.setpath(sdir), type*order));
107                 //T.plot("Connectivity");
108 #ifndef _DEBUG
109                 cout << "\nType the connectivity file you want to open (
                        e.g., C:/Users/.../Element_2D_quad_lin.dat) \n\n
```

```
                    Note:  1)the \"/\" character instead of \"/\"   \
110                                                                 \n  Note : 2) If
                                                                        the file is
                                                                        not
                                                                    coherent
                                                                    with the
                                                                    analysis
                                                                    type
                                                                    selected,
                                                                    errors will
                                                                    occur\n" <<
                                                                    endl;
111                 cin >> sdir2;
112 #endif
113
114                 Matrix X(Dir1.Read2(Dir1.setpath(sdir2), dim));
115
116                 //X.plot("Position");
117                 Mesh mesh(T, X,dim,type,order);
118         }
119
120         cin.get();
121         return 0;
122 }
123
124 // C:\Users\Rafael\Dropbox\Work\UPC-PES-HW1\Mesh\PES_2D_quad_lin\Mesh1\
      Element_2D_quad_lin.dat
125 // C:\Users\Rafael\Dropbox\Work\UPC-PES-HW1\Mesh\PES_2D_quad_lin\Mesh1\
      Node_2D_quad_lin.dat
126
127 // C:\Users\Rafael\Dropbox\Work\UPC-PES-HW1\Mesh\PES_2D_quad_quad\Mesh1\
      Element_2D_quad_quad.dat
128 // C:\Users\Rafael\Dropbox\Work\UPC-PES-HW1\Mesh\PES_2D_quad_quad\Mesh1\
      Node_2D_quad_quad.dat
```

```
1 #include <stdlib.h>
2 #include <algorithm>
3 #include <iostream>
4 #include <cmath>
5 #include <memory>
6 #include <string>
7 #include <fstream>
8
9 using namespace std;
```

```
10
11 #define Verify(s)                                                    \
12         if (!(s))                                                    \
13                 {                                                    \
14                 cerr << "Error: " << "No more memory available\n";   \
15                 exit(1);                                             \
16                 }                                                    \
17
18 #define Error(ToStringStream)                                        \
19   {                                                                  \
20     cerr << "Error: " << ToStringStream << "\n";                     \
21     cin.get();                                                       \
22         exit(1);                                                     \
23   }
24
25
26 class Input
27 {
28
29 public:
30         friend class Matrix;
31         friend class vector;
32         Matrix M;
33         Matrix Read(string s , int n);
34         Matrix Read2(string s, int n);
35         string setpath(string s);
36         void ReplaceStringInPlace(std::string& subject, const std::
               string& search,
37                 const std::string& replace) {
38                 size_t pos = 0;
39                 while ((pos = subject.find(search, pos)) != std::string
                       ::npos) {
40                         subject.replace(pos, search.length(), replace);
41                         pos += replace.length();
42                 }
43         }
```

```
44
45 };
```

```
1  #include "stdafx.h"
2  #include "Matrix.h"
3  #include "vector.h"
4  #include <stdlib.h>
5  #include <algorithm>
6  #include <iostream>
7  #include <fstream>
8  #include <cmath>
9  #include <memory>
10 #include <cstring>
11 #include <string.h>
12 #include "Input.h"
13
14
15 using namespace std;
16
17 Matrix Input::Read(string s, int n){
18         ifstream myReadFile;
19         myReadFile.open(s);
20         char output[100];
21         Matrix M(20000, n);
22         if (myReadFile.is_open()) {
23                 int i(1);
24                 int j(1);
25                 while (!myReadFile.eof()) {
26
27                         myReadFile >> output;
28                         M(i, j)= atof( output);
29                         if (j  == n)
30                         {
31                                 j = 0;
32                                 i++;
33                         }
34                         j++;
35
36
37                 }
38                 M.redimension(i - 1, n);
39                 myReadFile.close();
40                 return M;
41
```

```
42          }
43  }
44  Matrix Input::Read2(string s, int n){
45          if (n == 3)
46          return  Read(s, n);
47          else{
48                  ifstream myReadFile;
49                  myReadFile.open(s);
50                  char output[100];
51                  Matrix M(20000, n);
52                  if (myReadFile.is_open()) {
53                          int i(1);
54                          int j(1);
55                          while (!myReadFile.eof()) {
56
57                                  myReadFile >> output;
58                                  if (j != 1)
59                                          M(i, j - 1) = atof(output);
60                                  if (j == n + 1)
61                                  {
62                                          j = 0;
63                                          i++;
64                                  }
65                                  j++;
66
67
68                          }
69                          M.redimension(i - 1, n);
70                          myReadFile.close();
71                          return M;
72
73                  }
74          }
75  }
76
77  string Input::setpath(string s)
78  {
79          ReplaceStringInPlace(s, "\\", "\\\\");
80          return s;
81  }
```

```
1  #include <stdlib.h>
2  #include <algorithm>
3  #include <iostream>
```

```
4  #include <cmath>
5  #include <memory>
6  #include <string>
7  #include <fstream>
8  using namespace std;
9
10 #define Verify(s)                                                    \
11         if (!(s))
                                                                       \
12                         {
                                                                       \
13                 cerr << "Error: " << "No more memory available\n";
                        \
14                 exit(1);
                                                                       \
15                         }
                                                                       \
16
17 #define Error(ToStringStream)                                       \
18   {                                                                 \
19     cerr << "Error: " << ToStringStream << "\n";                    \
20     cin.get();
                                                \
21         exit(1);                                                          \
22   }
23 class Element;
24 class Mesh
25 {
26 private:
27
28         Matrix *Cond;
29         vector* N, *D;
30 protected:
31         enum Dimension{ Dim2 = 2, Dim3 = 3 } dimension;
32         enum Type { Linear = 2, Triangle = 3, Quadrilateral = 4,
              Tetahedra = 5 } type;
33         enum Order{ Lin = 1, Quad = 2 } order;
34         Matrix* T;
35         Matrix* X;
36         Matrix* Xm;
37         Matrix* K,*F,*KU;
38         int dim;
```

```
39          Element* elem;
40          friend class Matrix;
41          friend class vector;
42          friend class Element;
43
44  public:
45          Mesh();
46          Mesh(int dimref, int typeref, int orderref);
47          Mesh(Matrix T0, Matrix X0 ,int dimref, int typeref, int orderref
                );
48          ~Mesh();
49          void createXm();
50          Matrix* GiveT(){ return T; };
51          Matrix* GiveX(){ return X; };
52          Matrix* GiveXm(){ return Xm; };
53          Matrix* GiveCond(){ return Cond; };
54          vector* GiveN(){ return N; };
55          vector* GiveD(){ return D; };
56          void SetDim(Dimension dimref){ dimension = dimref; }
57          void SetType(Type typeref){ type = typeref; }
58          void SetOrder(Order orderref){ order = orderref; }
59          void CreateCond();
60          int Givetype(){}
61          Matrix Create_element(int i);
62          vector Localaxis(Matrix X, vector nodes);
63
64
65  };
66
67  class Boundary : public Mesh{
68  protected:
69          Matrix* BC;
70          vector* qn;
71
72  public:
73          Boundary() : Mesh(){};
74          Matrix createBC( Matrix* T2, Matrix* X2, Matrix* Cond , vector*
                BC);
75          void SetBC(Matrix BC0){ BC = new Matrix(BC0); };
76          void Setqn(vector qn0){ qn = new vector(qn0); };
77          vector* Giveqn(){ return qn; };
78
79  };
```

```
1 #include "stdafx.h"
2 #include "Matrix.h"
3 #include "vector.h"
4 #include <stdlib.h>
5 #include <algorithm>
6 #include <iostream>
7 #include <fstream>
8 #include <cmath>
9 #include <memory>
10 #include <cstring>
11 #include <string.h>
12 #include "Mesh.h"
13 #include "Element.h"
14
15 Mesh::Mesh()
16 {
17
18 }
19
20 Mesh::Mesh(int dimref, int typeref, int orderref)
21 {
22         if (dimref == 2)
23         {
24                 dimension = Dim2;
25         }
26         else if (dimref == 3)
27         {
28                 dimension = Dim3;
29         }
30         switch (typeref){
31         case 2: type = Linear; break;
32         case 3: type = Triangle; break;
33         case 4: type = Quadrilateral; break;
34         case 5: type = Tetahedra; break;
35         }
36         switch (orderref){
37         case 1: order = Lin; break;
38         case 2: order = Quad; break;
39         }
40 }
41 Mesh::Mesh(Matrix T0, Matrix X0, int dimref, int typeref, int orderref)
42 {
43         if (dimref == 2)
44         {
```

```
45              dimension = Dim2;
46          }
47          else if (dimref == 3)
48          {
49              dimension = Dim3;
50          }
51          switch (typeref){
52          case 2: type = Linear; break;
53          case 3: type = Triangle; break;
54          case 4: type = Quadrilateral; break;
55          case 5: type = Tetahedra; break;
56          }
57          switch (orderref){
58          case 1: order = Lin; break;
59          case 2: order = Quad; break;
60          }
61
62          T = new Matrix(T0);
63          X= new Matrix (X0);
64          dim = int (X0.ncols());
65          createXm();
66          Boundary Neumann, Dirichlet;
67          CreateCond();
68          if (dim == 3) { cout << endl << "Neumann" << endl << "k node x y
                qx qy qz" << endl; Neumann.SetDim(dimension); Neumann.
               SetType(Triangle); Neumann.SetOrder(Lin); }
69          else { cout << endl << "Neumann" << endl << "k node x y z qx qy
               qz" << endl; SetDim(dimension); Neumann.SetType(Linear);
               Neumann.SetOrder(order); }
70          Neumann.SetBC(Neumann.createBC(T, X, Cond ,N ) );
71          if (dim == 3){ cout << endl << "Dirichlet" << endl << "k node x
               y ux uy yz" << endl; Dirichlet.SetDim(dimension); Dirichlet.
               SetType(Triangle); Dirichlet.SetOrder(Lin); }
72          else { cout << endl << "Dirichlet" << endl << "k node x y z ux
               uy yz" << endl; SetDim(dimension); Dirichlet.SetType(Linear)
               ; Dirichlet.SetOrder(order); }
73          Dirichlet.SetBC(Dirichlet.createBC(T, X, Cond, D ) );
74          vector qn_temp(*Dirichlet.Giveqn()); qn_temp = 0;
75          Dirichlet.Setqn(qn_temp);
76
77          Matrix Ktemp(X->nrows(), X->nrows()); Ktemp = 0;
78          Matrix Ftemp(X->nrows(), 1); Ftemp = 0;
79          Matrix KUtemp(X->nrows(), 1); Ftemp = 0;
80          vector qn_Neumann(*Neumann.Giveqn());
```

```
81          vector qn_Dirichlet(*Dirichlet.Giveqn());
82          for (int i(1); i <= Neumann.Giveqn()->L(); i++)
83          {
84
85                  elem = new Element(Neumann.type, Neumann.order,
                        qn_Neumann(i));
86                  Matrix Fe(*elem->GiveF());
87                  for (int ii(1); ii <= T->ncols(); ii++)
88                          Ftemp(T0(i, ii), 1) = Ftemp(T0(i, ii), 1) + Fe(
                            ii, 1);
89          }
90
91          for (int i(1); i <= Dirichlet.Giveqn()->L(); i++)
92          {
93                  for (int ii(1); ii <= T->ncols(); ii++)
94                          KUtemp(T0(i, ii), 1) = KUtemp(T0(i, ii), 1) +
                            qn_Dirichlet(ii);
95          }
96
97
98          for (int i(1); i <= T->nrows(); i++){
99                  elem = new Element(type, order, Create_element(i));
100                 Matrix Ke(*elem->GiveK());
101                 for (int ii(1); ii <= T->ncols(); ii++)
102                         for (int jj(1); jj <= T->ncols(); jj++)
103                         {
104                         Ktemp(T0(i, ii), T0(i, jj)) = Ktemp(T0(i, ii),
                            T0(i, jj)) + Ke(ii, jj);
105                         }
106         }
107 }
108 Mesh::~Mesh()
109 {
110 }
111
112 void Mesh::createXm()
113 {
114
115         Matrix Xm_temp(T->nrows(), X->ncols()); Xm_temp = 0;
116         Matrix T0(*T), X0(*X);
117         int x(0), y(0), z(0);
118         for (int i(1); i <= T->nrows(); i++)
119         {
120                 x = 0; y = 0; z = 0;
```

```
121                 for (int j(1); j <= T->ncols(); j++)
122                         for (int z(1); z <= X->ncols(); z++)
123                         Xm_temp(i, z) += X0(T0(i,j), z) / T->ncols();
124         }
125         Xm = new Matrix(Xm_temp);
126 }
127
128 Matrix Boundary::createBC(Matrix* T2, Matrix* X2, Matrix* Cond2, vector
        *BC)
129 {
130         vector BC0(*BC), qn2(T2->nrows()), q(3), n(3); qn2 = 0;
131         Matrix T0(*T2), X0(*X2), Cond0(*Cond2), Xtemp(X2->nrows(), X2->
              ncols()), Ttemp(T2->nrows(), T2->ncols()), BC_temp(T2->ncols
              () + 3, T2->nrows()); BC_temp = 0;
132         Matrix Tloc(1, T2->ncols()), Xloc(3, T2->ncols()); Xloc = 0;
133         vector nodes(T2->ncols()); nodes = 0;
134         int cont(0);
135         double sum;
136         dim = (X0.ncols());
137         int cont2 = 1, alarm1 = 0;
138         for (int k(1); k <= BC->L(); k++)
139                 for (int i(1); i <= T0.nrows(); i++){
140                 nodes = 0;
141                 for (int j(1); j <= T0.ncols(); j++)
142                 {
143                         cont = 0; sum = 0.0;
144                         for (int ii(1); ii <= X0.ncols(); ii++){
145                                 sum += X0(T0(i, j), ii)*Cond0(BC0(k), ii
                                      );
146                                 if (X0(T0(i, j), ii) >= Cond0(BC0(k), 3
                                      + 2 * ii) & (X0(T0(i, j),ii) <=
                                      Cond0(BC0(k), 4 + 2 * ii)))
147                                         cont++;
148                                 cout << X0(T0(i, j), ii) << "  " <<
                                      Cond0(BC0(k), 3 + 2 * ii) << "  " <<
                                       Cond0(BC0(k), 4 + 2 * ii) << " " <<
                                       cont <<  endl;
149                         }
150                         if (cont == dim & sum == Cond0(BC0(k), 4)){
151                                 for (int ii(1); ii <= X0.ncols(); ii++){
152                                         BC_temp(T0.ncols() + ii, cont2)
                                              = Cond0(BC0(k), 10 + ii);
153                                  Xloc(ii, j) = X0(T0(i, j), ii);
154                                 }
```

```
155                                     cout << endl;
156                                     BC_temp(j, cont2) = T0(i, j);
157                                     nodes(j) = 1;
158                                     alarm1++;
159
160                             }
161
162
163                     }
164                     if (alarm1 >1){
165                             n = Localaxis(Xloc, nodes);
166                             q(1) = Cond0(BC0(k), 11); q(2)=Cond0(BC0(k), 12)
                                    ;q(3)= Cond0(BC0(k), 13);
167                             qn2(cont2) = q(1)*n(1) + q(2)*n(2) + q(3)*n(3);
168                             for (int kk(1); kk <= T0.ncols(); kk++)
169                             Ttemp(cont2,kk)=T0(i,kk);
170                             cont2++;
171
172                     }
173                     alarm1 = 0;
174
175                     }
176             cont2--;
177
178                     Ttemp.redimension(cont2, T0.ncols());
179                     BC_temp.redimension(T2->ncols() + 3, cont2);
180                     qn2.resize(cont2);
181
182                     T = new Matrix(Ttemp);
183                     qn = new vector(qn2);
184                     return BC_temp;
185 }
186 vector Mesh::Localaxis(Matrix X, vector nodes)
187 {
188             Matrix Xm_temp(1, 3); Xm_temp = 0; Matrix ntemp(3, 3); ntemp =
                    0;
189             double norm, sum=0.0;
190             int cont = 1;
191             vector n(3); n = 0;
192             for (int i(1); i <= nodes.L(); i++)
193                     sum += nodes(i);
194
195                     for (int j(1); j <= nodes.L(); j++){
196                             for (int z(1); z <= X.ncols(); z++)
```

```
197                                 Xm_temp(1, z) += nodes(j)*X(z, j) / sum;
198
199                             if (nodes(j) == 0){
200                                 norm = 0;
201                                 for (int z(1); z <= X.ncols(); z++)
202                                 {
203                                     ntemp(2, z) = (X(j, z) - Xm_temp
                                            (1, z));
204                                     norm = pow(ntemp(cont, z), 2) +
                                            norm;
205                                 }
206                                 ntemp(1, 1) = ntemp(1, 1) / norm; ntemp
                                        (1, 2) = ntemp(1, 2) / norm; ntemp
                                        (1, 3) = ntemp(1, 3) / norm;
207                             }
208                             else {
209                                 norm = 0;
210                                 for (int z(1); z <= X.ncols(); z++)
211                                 {
212                                     ntemp(2, z) = (X(j, z) - Xm_temp
                                            (1, z));
213                                     norm = pow(ntemp(cont, z), 2) +
                                            norm;
214                                 }
215                                 ntemp(2, 1) = ntemp(2, 1) / norm; ntemp
                                        (2, 2) = ntemp(2,2 ) / norm; ntemp
                                        (2, 3) = ntemp(2, 3) / norm;
216                             }
217
218                 }
219                 n(1) = ntemp(1, 2)*ntemp(2, 3) - ntemp(1, 3)*ntemp(2, 2)
                        ;
220                 n(2)= ntemp(1, 3)*ntemp(2, 1) - ntemp(1, 1)*ntemp(2, 3);
221                 n(3) = ntemp(1, 1)*ntemp(2, 2) - ntemp(1, 2)*ntemp(2, 1)
                        ;
222                 if (n(3) < 0)
223                     n(1) = -n(1);
224         return n;
225 }
226 void Mesh::CreateCond()
227 {
228         dim = int(X->ncols());
229         enum Coefficients {a=1,b=2,c=3,d=4,x1=5,x2=6,y1=7,y2=8,z1=9,z2
                =10,C=13 } ;
```

```
230         Matrix Condtemp(100, C);
231         vector Ntemp(100), Dtemp(100);
232         int cont(1),Ncont(1),Dcont(1), verify(0);
233         string str="";
234         double temp(0);
235 #ifndef _DEBUG
236                 do {
237                         cout << "More Conditions to be applied? (y=1/n
                            =0)";
238                         cin >> verify;
239                         if (verify == 1)
240                         {
241                                 cout << "\nDtempefine the boundary
                                    equation: \n\n 1. The domain
                                    equation where it is applied (e.g. a
                                    *x+b*y+c*z=d) .\n\n 2. The limits (e
                                    .g. x=[x1 , x2], y=[y1 , y2], z=[z1
                                    , z2] . \n\n 3. Ntempote 0 means
                                    there is no parameters. (Useful for
                                    point/line/area Condtempitions) \n\n
                                      Coefficients: \n   a = ";
242                                 cin >> Condtemp(cont,a);
243                                 cout << "\n   b = ";
244                                 cin >> Condtemp(cont, b);
245                                 cout << "\n   c = ";
246                                 cin >> Condtemp(cont, c);
247                                 cout << "\n   d = ";
248                                 cin >> Condtemp(cont, d);
249                                 cout << "\n  Limits : ";
250                                 cout << "\n   x1 = ";
251                                 cin >> Condtemp(cont, x1);
252                                 cout << "\n   x2 = ";
253                                 cin >> Condtemp(cont, x2);
254                                 if (Condtemp(cont, x2) < Condtemp(cont,
                                    x1)) { temp = Condtemp(cont, x2);
                                    Condtemp(cont, x2) = Condtemp(cont,
                                    x1); Condtemp(cont, x1) = temp; }
255                                 cout << "\n   y1 = ";
256                                 cin >> Condtemp(cont, y1);
257                                 cout << "\n   y2 = ";
258                                 cin >> Condtemp(cont, y2);
259                                 if (Condtemp(cont, y2) < Condtemp(cont,
                                    y1)) { temp = Condtemp(cont, y2);
                                    Condtemp(cont, y2) = Condtemp(cont,
```

```
                y1); Condtemp(cont, y1) = temp; }
260             cout << "\n   z1 = ";
261             cin >> Condtemp(cont, z1);
262             cout << "\n   z2 = ";
263             cin >> Condtemp(cont, z2);
264             if (Condtemp(cont, z2) < Condtemp(cont,
                    z1)) { temp = Condtemp(cont, z2);
                    Condtemp(cont, z2) = Condtemp(cont,
                    z1); Condtemp(cont, z1) = temp; }
265             do{
266                     cout << "\n   Neumann(1) /
                            Dirichlet (0) : ";
267                     cin >> Condtemp(cont, C);
268                     if (Condtemp(cont, C) == 1 ||
                            Condtemp(cont, C) == 0) {
                            break; }
269                     else {
270                             cout << endl << "Error"
                                    << endl;
271                     }
272             } while (true);
273             if (Condtemp(cont, C) == 1){ str = "
                    Neumann"; Ntemp(Ncont) = cont; Ncont
                    ++; cout << "\n      qx = "; cin >>
                    Condtemp(cont, C-2); cout << "\n
                         qy = "; cin >> Condtemp(cont, C
                    -1); cout << "\n      qz = "; cin >>
                     Condtemp(cont, C);}
274             else { str = "Dirichlet"; Dtemp(Dcont)
                    = cont; Dcont++; cout << "\n      ux
                     = "; cin >> Condtemp(cont, C-2);
                    cout << "\n      uy = "; cin >>
                    Condtemp(cont, C-1); cout << "\n
                         uz = "; cin >> Condtemp(cont, C
                    ); }
275             cout << "Your " << str<< " Boundary
                    Conditions is : \n   " << Condtemp(
                    cont, a) << "x" << Condtemp(cont, b)
                     << "y" << Condtemp(cont, c) << "z =
                     " << Condtemp(cont, d) << ", \n
                    for : x[" << Condtemp(cont, x1) << "
                    ," << Condtemp(cont, x2) << "]\n
                             y[" << Condtemp(cont, y1)
                    << "," << Condtemp(cont, y2) << "]\n
```

```
                                                z[" << Condtemp(cont, z1)
                                            << "," << Condtemp(cont, z2) << "]"
                                            << endl;
276                                     cont++;
277                             }
278                             else if (verify == 0)
279                             {
280                                     Condtemp.redimension(cont - 1, C);
281                                     Ntemp.resize(Ncont - 1);
282                                     Dtemp.resize(Dcont - 1);
283                                     break;
284                             }
285                             else
286                                     cout << "Error press retype again"<<endl
                                            ;
287
288                     } while (true);
289 #else
290             Condtemp(1,1)=1 ;Condtemp(1,2)=-1 ;Condtemp(1,3)=0 ;Condtemp
                    (1,4)=0 ;Condtemp(1,5)=0 ;Condtemp(1,6)=0 ;Condtemp(1,7)=0 ;
                    Condtemp(1,8)=1 ;Condtemp(1,9)=0 ;Condtemp(1,10)=0 ;Condtemp
                    (1,11)=-1;Condtemp(1,12)=0 ;Condtemp(1,13)=0 ;
291             Condtemp(2,1)=1 ;Condtemp(2,2)=-1 ;Condtemp(2,3)=0 ;Condtemp
                    (2,4)=0 ;Condtemp(2,5)=1 ;Condtemp(2,6)=1 ;Condtemp(2,7)=0 ;
                    Condtemp(2,8)=1 ;Condtemp(2,9)=0 ;Condtemp(2,10)=0 ;Condtemp
                    (2,11)=1 ;Condtemp(2,12)=0 ;Condtemp(2,13)=0 ;
292             Condtemp(3,1)=1 ;Condtemp(3,2)=-1 ;Condtemp(3,3)=0 ;Condtemp
                    (3,4)=0 ;Condtemp(3,5)=0 ;Condtemp(3,6)=1 ;Condtemp(3,7)=0 ;
                    Condtemp(3,8)=0 ;Condtemp(3,9)=0 ;Condtemp(3,10)=0 ;Condtemp
                    (3,11)=0 ;Condtemp(3,12)=0 ;Condtemp(3,13)=0 ;
293             Condtemp(4,1)=1 ;Condtemp(4,2)=-1 ;Condtemp(4,3)=0 ;Condtemp
                    (4,4)=0 ;Condtemp(4,5)=0 ;Condtemp(4,6)=1 ;Condtemp(4,7)=1 ;
                    Condtemp(4,8)=1 ;Condtemp(4,9)=0 ;Condtemp(4,10)=0 ;Condtemp
                    (4,11)=0 ;Condtemp(4,12)=0 ;Condtemp(4,13)=0 ;
294             Condtemp(5,1)=1 ;Condtemp(5,2)=-1 ;Condtemp(5,3)=0 ;Condtemp
                    (5,4)=0 ;Condtemp(5,5)=0 ;Condtemp(5,6)=0 ;Condtemp(5,7)=0 ;
                    Condtemp(5,8)=0 ;Condtemp(5,9)=0 ;Condtemp(5,10)=0 ;Condtemp
                    (5,11)=0 ;Condtemp(5,12)=0 ;Condtemp(5,13)=0 ;
295             Ntemp(1) = 1; Ntemp(2) = 2; Ntemp(3) = 3; Ntemp(4) = 4; Ntemp.
                    resize(4); Dtemp(1) = 5; Dtemp.resize(1); Condtemp.
                    redimension(5, C);
296 #endif
297
298             cout << endl << endl << "Neumann Conditions:" << endl << "a  b
```

```
            c  d  x1 x2 y1 y2 z1 z2 qx qy qz" << endl;
299         for (int i(1); i <= Ntemp.L(); i++){
300                 for (int j(1); j <= C; j++)
301                         cout << Condtemp(Ntemp(i), j) << "  ";
302                 cout << endl;
303         }
304         cout << endl << endl << "Dirichlet Conditions:" << endl << "a  b
                 c  d  x1 x2 y1 y2 z1 z2 ux uy uz" << endl;
305         for (int i(1); i <= Dtemp.L(); i++)
306                 for (int j(1); j <= C; j++)
307                         cout << Condtemp(Dtemp(i), j) << "  ";
308         cout << endl;
309         N = new vector(Ntemp);
310         D = new vector(Dtemp);
311         Cond = new Matrix(Condtemp);
312 }
313
314 Matrix Mesh::Create_element(int i){
315         Matrix Ttemp(*T), Xtemp(*X), P(Ttemp.ncols(), Xtemp.ncols());
316         for (int j(1); j <= Ttemp.ncols(); j++)
317         {
318                 P(j, 1) = Xtemp(Ttemp(i,j), 1);
319                 P(j, 2) = Xtemp(Ttemp(i,j), 2);
320         }
321         return P;
322 }
```

```
1  // stdafx.h : include file for standard system include files,
2  // or project specific include files that are used frequently, but
3  // are changed infrequently
4  //
5
6  #pragma once
7
8  #include "targetver.h"
9
10 #include <stdio.h>
11 #include <tchar.h>
12
13
14
15 // TODO: reference additional headers your program requires here
```

```
1  // stdafx.cpp : source file that includes just the standard includes
```

```
2 // Exercise_Julio_1.pch will be the pre-compiled header
3 // stdafx.obj will contain the pre-compiled type information
4
5 #include "stdafx.h"
6
7 // TODO: reference any additional headers you need in STDAFX.H
8 // and not in this file

1 #include <stdlib.h>
2 #include <algorithm>
3 #include <iostream>
4 #include <cmath>
5 #include <memory>
6 #include <string>
7 #include <fstream>
8 using namespace std;
9 class Mesh;
10 class Element{
11 protected:
12         enum BC{ None = 0, Neumann = 1, Dirichlet = 2, Both = 3 } bc=
             None;
13         friend class Matrix;
14         friend class vector;
15         friend class Mesh;
16         enum Type { Linear = 2, Triangle = 3, Quadrilateral = 4,
             Tetahedra=5 } type;
17         enum Order{ Lin = 1, Quad = 2 } order;
18         Matrix *J, *invJ, *B, *K, *F;
19         double detJ;
20         int Gauss;
21         Matrix* z, *w;
22         Matrix* N,*Nxi,*Neta,*nodesCoord;
23         Matrix* P;
24
25 public:
26         Element();
27         Element(int typeref, int orderref,double n);
28         Element( int typeref, int orderref, Matrix X);
29         ~Element();
30         void create_J(int i);
31         void create_B(int i);
32         void create_K(int i);
33         void create_F(int i,double qn);
34         void create_N();
```

```
35          void create_Gauss();
36
37          void SetNeumann(){ if (bc != None) bc = Both; else bc = Neumann;
                }
38          void SetDirichlet(){ if (bc != None) bc = Both; else bc =
                Dirichlet; }
39          Matrix* GiveK(){ return K; };
40          Matrix* GiveF(){ return F; };
41
42
43  };
```

```
1  #include "stdafx.h"
2  #include "Element.h"
3  #include "Matrix.h"
4  #include "vector.h"
5
6
7  Element::Element()
8  {
9  }
10 Element::Element(int typeref, int orderref, double qn)
11 {
12          switch (typeref){
13          case 2: type = Linear; break;
14          case 3: type = Triangle; break;
15          case 4: type = Quadrilateral; break;
16          case 5: type = Tetahedra; break;
17          }
18          switch (orderref){
19          case 1: order = Lin; break;
20          case 2: order = Quad; break;
21          }
22          create_Gauss();
23          create_N();
24          for (int i(1); i <= Gauss; i++)
25          {
26                  create_F(i,qn);
27          }
28 }
29
30 Element::Element(int typeref, int orderref, Matrix Pref)
31 {
32          switch (typeref){
```

```
33          case 2: type = Linear; break;
34          case 3: type = Triangle; break;
35          case 4: type = Quadrilateral; break;
36          case 5: type = Tetahedra; break;
37          }
38          switch (orderref){
39          case 1: order = Lin; break;
40          case 2: order = Quad; break;
41          }
42          P = new Matrix(Pref);
43          create_Gauss();
44          create_N();
45          for (int i(1); i <= Gauss; i++)
46          {
47          create_J(i);
48          create_B(i);
49          create_K(i);
50
51
52          }
53
54 }
55 Element::~Element()
56 {
57 }
58
59 void Element::create_J(int i)
60 {
61          Matrix N_temp(*N), P_temp(*P), nodesCoord_temp(*nodesCoord),
                N_dif(2, N_temp.ncols()), Nxi_temp(*Nxi), Neta_temp(*Neta);
62                  for (int j(1); j <=N_temp.ncols(); j++)
63                  {
64                  N_dif(1,j) = Nxi_temp(i, j);
65                  N_dif(2, j) = Neta_temp(i, j);
66                  }
67                  Matrix J_temp(nodesCoord_temp.nrows(), P_temp.ncols());
                        J_temp = 0; Matrix J_temp1(N_dif.nrows(), P_temp.
                        ncols()); J_temp1 = 0;
68                  J_temp.product(nodesCoord_temp, P_temp);
69                  J_temp1.product(N_dif, J_temp);
70                  J_temp = J_temp1;
71                  Matrix J_temp2(J_temp);
72                  J_temp2.i(J_temp1);
73                  J = new Matrix(J_temp1);
```

```
74                  invJ = new Matrix(J_temp2);
75                  detJ = J->det();
76          }
77
78  void Element::create_B(int i)
79  {
80          Matrix N_temp(*N), P_temp(*P), nodesCoord_temp(*nodesCoord),
                N_dif(2, N_temp.ncols()), Nxi_temp(*Nxi), Neta_temp(*Neta);
81          for (int j(1); j <= N_temp.ncols(); j++)
82          {
83                  N_dif(1, j) = Nxi_temp(i, j);
84                  N_dif(2, j) = Neta_temp(i, j);
85          }
86          B = new Matrix(N_dif);
87
88  }
89  void Element::create_K(int i)
90  {
91
92          Matrix Btemp(*B), BJ(J->nrows(), B->ncols()), BJt(B->ncols(), J
                ->nrows()), wtemp(*w);
93          if (i == 1){
94                  Matrix K_init(B->ncols(), B->ncols()); K_init = 0; K =
                        new Matrix(K_init);
95          }
96          BJ = 0; BJ.add_product(*invJ, Btemp);
97          BJt = 0;;
98          double wdetJ (wtemp(1,i)*detJ);
99          for (int j(1); j <= B->ncols(); j++)
100                 for (int k(1); k <= J->nrows(); k++)
101                         BJt(j, k) = BJ(k, j);
102         K->add_product(BJt, BJ, wdetJ);
103 }
104
105 void Element::create_F(int i,double qn)
106 {
107
108         Matrix  wtemp(*w),N_temp(*N);
109         if (i == 1){
110                 Matrix F_init(B->ncols(), B->ncols()); F_init = 0; F =
                        new Matrix(F_init);
111         }
112         double wdetJ(wtemp(1, i)*detJ*qn);
113         F->add_product(N_temp, wdetJ);
```

```
114 }
115
116
117 void Element::create_N()
118 {
119         if (type == Linear)
120         {
121                 if (order == Lin)
122                 {
123                         Matrix nodesCoordtemp(2, 2);
124                         nodesCoordtemp(1, 1) = 1.0; nodesCoordtemp(1, 2)
                                = 0.0;
125                         nodesCoordtemp(2, 1) = 0.0; nodesCoordtemp(2, 2)
                                = 1.0;
126                         Matrix Ntemp(Gauss, 2), ztemp(*z), Nxitemp(Gauss
                            , 2), Netatemp(Gauss, 2);
127                         for (int i(1); i <= Gauss; i++)
128                         {
129                                 Ntemp(Gauss, 1) = 0.5*(1-ztemp(1, i));
130                                 Ntemp(Gauss, 2) = 0.5*(1+ztemp(1, i) );
131
132                                 Nxitemp(Gauss, 1) = -0.5;
133                                 Nxitemp(Gauss, 2) = 0.5;
134
135                                 Netatemp(Gauss, 1) = 0;
136                                 Netatemp(Gauss, 2) = 0;
137                         }
138                         N = new Matrix(Ntemp); Nxi = new Matrix(Nxitemp)
                            ; Neta = new Matrix(Netatemp); nodesCoord =
                            new Matrix(nodesCoordtemp);
139                 }
140                 else if (order == Quad)
141                 {
142                         Matrix nodesCoordtemp(3, 3);
143                         nodesCoordtemp(1, 1) = 1.0; nodesCoordtemp(1, 2)
                                = 0.0; nodesCoordtemp(1, 3) = 0.0;
144                         nodesCoordtemp(2, 1) = 0.0; nodesCoordtemp(2, 2)
                                = 1.0; nodesCoordtemp(2, 3) = 0.0;
145                         nodesCoordtemp(3, 1) = 0.0; nodesCoordtemp(3, 2)
                                = 0.0; nodesCoordtemp(3, 3) = 1.0;
146                         Matrix Ntemp(Gauss, 2), ztemp(*z), Nxitemp(Gauss
                            , 2), Netatemp(Gauss, 2);
147                         for (int i(1); i <= Gauss; i++)
148                         {
```

```
149                                 Ntemp(Gauss, 1) = 0.5*ztemp(1, i)*(1.0 -
                                        ztemp(1, i));
150                                 Ntemp(Gauss, 2) = (1.0 + ztemp(1, i))
                                        *(1.0 - ztemp(1, i));
151                                 Ntemp(Gauss, 3) = 0.5*ztemp(1, i)*(1.0 +
                                        ztemp(1, i));
152
153                                 Nxitemp(Gauss, 1) = -0.5*(2.0*ztemp(1, i
                                        )-1.0);
154                                 Nxitemp(Gauss, 2) = -2.0*ztemp(1, i);
155                                 Nxitemp(Gauss, 3) = 0.5*(2.0*ztemp(1, i)
                                        - 1.0);
156
157                                 Netatemp(Gauss, 1) = 0;
158                                 Netatemp(Gauss, 2) = 0;
159                                 Netatemp(Gauss, 3) = 0;
160                         }
161                         N = new Matrix(Ntemp); Nxi = new Matrix(Nxitemp)
                                ; Neta = new Matrix(Netatemp); nodesCoord =
                                new Matrix(nodesCoordtemp);
162                 }
163         }
164         else if (type == Triangle)
165         {
166                 if (order == Lin)
167                 {
168                         Matrix nodesCoordtemp(3, 3);
169                         nodesCoordtemp(1, 1) = 1.0; nodesCoordtemp(1, 2)
                                = 0.0; nodesCoordtemp(1, 3) = 0.0;
170                         nodesCoordtemp(2, 1) = 0.0; nodesCoordtemp(2, 2)
                                = 1.0; nodesCoordtemp(2, 3) = 0.0;
171                         nodesCoordtemp(3, 1) = 0.0; nodesCoordtemp(3, 2)
                                = 0.0; nodesCoordtemp(3, 3) = 1.0;
172                         Matrix Ntemp(Gauss, 3), ztemp(*z), Nxitemp(Gauss
                                , 3), Netatemp(Gauss, 3);
173                         for (int i(1); i <= Gauss; i++)
174                         {
175                                 Ntemp(Gauss, 1) = -0.5*(ztemp(1, i) +
                                        ztemp(2, i));
176                                 Ntemp(Gauss, 2) = 0.5*(ztemp(1, i) +
                                        1.0);
177                                 Ntemp(Gauss, 3) = 0.5*(1.0 + ztemp(2, i)
                                        );
178
```

```
179                         Nxitemp(Gauss, 1) = -0.5;
180                         Nxitemp(Gauss, 2) = 0.5;
181                         Nxitemp(Gauss, 3) = 0.0;
182
183                         Netatemp(Gauss, 1) = -0.5;
184                         Netatemp(Gauss, 2) = 0.0;
185                         Netatemp(Gauss, 3) = 0.5;
186                     }
187                     N = new Matrix(Ntemp); Nxi = new Matrix(Nxitemp)
                            ; Neta = new Matrix(Netatemp); nodesCoord =
                            new Matrix(nodesCoordtemp);
188                 }
189                 else if (order == Quad)
190                 {
191                     Matrix nodesCoordtemp(3, 6);
192                     nodesCoordtemp(1, 1) = 1.0; nodesCoordtemp(1, 2)
                            = 0.0; nodesCoordtemp(1, 3) = 0.0;
                        nodesCoordtemp(1, 4) = 0.5; nodesCoordtemp
                        (1, 5) = 0.0; nodesCoordtemp(1, 6) = 0.5;
193                     nodesCoordtemp(2, 1) = 0.0; nodesCoordtemp(2, 2)
                            = 1.0; nodesCoordtemp(2, 3) = 0.0;
                        nodesCoordtemp(2, 4) = 0.5; nodesCoordtemp
                        (2, 5) = 0.5; nodesCoordtemp(2, 6) = 0.0;
194                     nodesCoordtemp(3, 1) = 0.0; nodesCoordtemp(3, 2)
                            = 0.0; nodesCoordtemp(3, 3) = 1.0;
                        nodesCoordtemp(3, 4) = 0.0; nodesCoordtemp
                        (3, 5) = 0.5; nodesCoordtemp(3, 6) = 0.5;
195                     Matrix Ntemp(Gauss, 6), ztemp(*z), Nxitemp(Gauss
                            , 6), Netatemp(Gauss, 6);
196                     for (int i(1); i <= Gauss; i++)
197                     {
198                         Ntemp(Gauss, 1) = 0.5*(ztemp(1, i) +
                                ztemp(2, i))*(ztemp(1, i) + ztemp(2,
                                 i) + 1);
199                         Ntemp(Gauss, 2) = 0.5*(ztemp(1, i) + 1)
                                *(ztemp(1, i));
200                         Ntemp(Gauss, 3) = 0.5*(1 + ztemp(2, i))
                                *(ztemp(2, i));
201                         Ntemp(Gauss, 4) = -(ztemp(1, i) + ztemp
                                (2, i))*(ztemp(1, i) + 1);
202                         Ntemp(Gauss, 5) = (ztemp(1, i) + 1)*(
                                ztemp(2, i) + 1);
203                         Ntemp(Gauss, 6) = -(ztemp(1, i) + ztemp
                                (2, i))*(ztemp(2, i) + 1);
```

```
204
205                                 Nxitemp(Gauss, 1) = ztemp(1, i) + ztemp
                                        (2, i) + 0.5;
206                                 Nxitemp(Gauss, 2) = ztemp(1, i) + 0.5;
207                                 Nxitemp(Gauss, 3) = 0;
208                                 Nxitemp(Gauss, 4) = -2 * ztemp(1, i) -
                                        ztemp(2, i) - 1;
209                                 Nxitemp(Gauss, 5) = ztemp(2, i) + 1;
210                                 Nxitemp(Gauss, 6) = -ztemp(2, i) - 1;
211
212                                 Nxitemp(Gauss, 1) = ztemp(1, i) + ztemp
                                        (2, i) + 0.5;
213                                 Nxitemp(Gauss, 2) = 0;
214                                 Nxitemp(Gauss, 3) = ztemp(2, i) + 0.5;
215                                 Nxitemp(Gauss, 4) = ztemp(1, i) + 1;
216                                 Nxitemp(Gauss, 5) = -ztemp(1, i) - 1;
217                                 Nxitemp(Gauss, 6) = -2 * ztemp(2, i) -
                                        ztemp(1, i) - 1;
218                         }
219                         N = new Matrix(Ntemp); Nxi = new Matrix(Nxitemp)
                                ; Neta = new Matrix(Netatemp); nodesCoord =
                                new Matrix(nodesCoordtemp);
220                 }
221         }
222         else if (type == Quadrilateral)
223         {
224                 if (order == Lin)
225                 {
226                         Matrix nodesCoordtemp(4, 4);
227                         nodesCoordtemp(1, 1) = 1.0; nodesCoordtemp(1, 2)
                                = 0.0; nodesCoordtemp(1, 3) = 0.0;
                               nodesCoordtemp(1, 4) = 0.0;
228                         nodesCoordtemp(2, 1) = 0.0; nodesCoordtemp(2, 2)
                                = 1.0; nodesCoordtemp(2, 3) = 0.0;
                               nodesCoordtemp(2, 4) = 0.0;
229                         nodesCoordtemp(3, 1) = 0.0; nodesCoordtemp(3, 2)
                                = 0.0; nodesCoordtemp(3, 3) = 1.0;
                               nodesCoordtemp(3, 4) = 0.0;
230                         nodesCoordtemp(4, 1) = 0.0; nodesCoordtemp(4, 2)
                                = 0.0; nodesCoordtemp(4, 3) = 0.0;
                               nodesCoordtemp(4, 4) = 1.0;
231                         Matrix Ntemp(Gauss, 4), ztemp(*z), Nxitemp(Gauss
                                , 4), Netatemp(Gauss, 4);
232                         for (int i(1); i <= Gauss; i++)
```

```
233                         {
234                             Ntemp(Gauss, 1) = 0.25*((1 - ztemp(1, i)
                                    )*(1 - ztemp(2, i)));
235                             Ntemp(Gauss, 2) = 0.25*((1 + ztemp(1, i)
                                    )*(1 - ztemp(2, i)));
236                             Ntemp(Gauss, 3) = 0.25*((1 + ztemp(1, i)
                                    )*(1 + ztemp(2, i)));
237                             Ntemp(Gauss, 4) = 0.25*((1 - ztemp(1, i)
                                    )*(1 + ztemp(2, i)));
238
239                             Nxitemp(Gauss, 1) = 0.25*(ztemp(2, i) -
                                    1);
240                             Nxitemp(Gauss, 2) = 0.25*(-ztemp(2, i) +
                                     1);
241                             Nxitemp(Gauss, 3) = 0.25*(ztemp(2, i) +
                                    1);
242                             Nxitemp(Gauss, 4) = -0.25*(ztemp(2, i) +
                                     1);
243
244                             Netatemp(Gauss, 1) = 0.25*(ztemp(1, i) -
                                     1);
245                             Netatemp(Gauss, 2) = -0.25*(ztemp(1, i)
                                    + 1);
246                             Netatemp(Gauss, 3) = 0.25*(ztemp(2, i) +
                                     1);
247                             Netatemp(Gauss, 4) = 0.25*(-ztemp(2, i)
                                    + 1);
248                         }
249                         N = new Matrix(Ntemp); Nxi = new Matrix(Nxitemp)
                                ; Neta = new Matrix(Netatemp); nodesCoord =
                                new Matrix(nodesCoordtemp);
250                 }
251                 else if (order == Quad)
252                 {
253                         Matrix nodesCoordtemp(4, 4);
254                         nodesCoordtemp(1, 1) = 1.0; nodesCoordtemp(1, 2)
                                 = 0.0; nodesCoordtemp(1, 3) = 0.0;
                                nodesCoordtemp(1, 4) = 0.0; nodesCoordtemp
                                (1, 5) = 0.5; nodesCoordtemp(1, 6) = 0.0;
                                nodesCoordtemp(1, 7) = 0.0; nodesCoordtemp
                                (1, 8) = 0.5; nodesCoordtemp(1, 9) = 0.25;
255                         nodesCoordtemp(2, 1) = 0.0; nodesCoordtemp(2, 2)
                                 = 1.0; nodesCoordtemp(2, 3) = 0.0;
                                nodesCoordtemp(2, 4) = 0.0; nodesCoordtemp
```

```
(2, 5) = 0.5; nodesCoordtemp(2, 6) = 0.5;
    nodesCoordtemp(2, 7) = 0.0; nodesCoordtemp
    (2, 8) = 0.0; nodesCoordtemp(2, 9) = 0.25;
256 nodesCoordtemp(3, 1) = 0.0; nodesCoordtemp(3, 2)
     = 0.0; nodesCoordtemp(3, 3) = 1.0;
    nodesCoordtemp(3, 4) = 0.0; nodesCoordtemp
    (3, 5) = 0.0; nodesCoordtemp(3, 6) = 0.5;
    nodesCoordtemp(3, 7) = 0.5; nodesCoordtemp
    (3, 8) = 0.0; nodesCoordtemp(3, 9) = 0.25;
257 nodesCoordtemp(4, 1) = 0.0; nodesCoordtemp(4, 2)
     = 0.0; nodesCoordtemp(4, 3) = 0.0;
    nodesCoordtemp(4, 4) = 1.0; nodesCoordtemp
    (4, 5) = 0.0; nodesCoordtemp(4, 6) = 0.0;
    nodesCoordtemp(4, 7) = 0.5; nodesCoordtemp
    (4, 8) = 0.5; nodesCoordtemp(4, 9) = 0.25;
258 Matrix Ntemp(Gauss, 4), ztemp(*z), Nxitemp(Gauss
    , 4), Netatemp(Gauss, 4);
259 for (int i(1); i <= Gauss; i++)
260 {
261     Ntemp(Gauss, 1) = ztemp(1, i)*(ztemp(1,
        i) - 1.0)*ztemp(2, i)*(ztemp(2, i) -
         1.0) / 4.0;
262     Ntemp(Gauss, 2) = ztemp(1, i)*(ztemp(1,
        i) + 1.0)*ztemp(2, i)*(ztemp(2, i) -
         1.0) / 4.0;
263     Ntemp(Gauss, 3) = ztemp(1, i)*(ztemp(1,
        i) + 1.0)*ztemp(2, i)*(ztemp(2, i) +
         1.0) / 4.0;
264     Ntemp(Gauss, 4) = ztemp(1, i)*(ztemp(1,
        i) - 1.0)*ztemp(2, i)*(ztemp(2, i) +
         1.0) / 4.0;
265     Ntemp(Gauss, 5) = (1 - pow(ztemp(1, i),
        2))*ztemp(2, i)*(ztemp(2, i) - 1.0)
        / 2;
266     Ntemp(Gauss, 6) = ztemp(1, i)*(ztemp(1,
        i) + 1.0)*(1.0 - pow(ztemp(2, i), 2)
        ) / 2;
267     Ntemp(Gauss, 7) = (1.0 - pow(ztemp(1, i)
        , 2))*ztemp(2, i)*(ztemp(2, i) +
        1.0) / 2;
268     Ntemp(Gauss, 8) = ztemp(1, i)*(ztemp(1,
        i) - 1.0)*(1.0 - pow(ztemp(2, i), 2)
        ) / 2;
269     Ntemp(Gauss, 9) = (1.0 - pow(ztemp(1, i)
```

```
                                        , 2))*(1.0 - pow(ztemp(2, i), 2));
270
271                                     Netatemp(Gauss, 1) = ztemp(1, i)*(ztemp
                                            (1, i) - 1.0)*(ztemp(2, i) - 1.0 /
                                            2.0) / 2.0;
272                                     Netatemp(Gauss, 2) = ztemp(1, i)*(ztemp
                                            (1, i) + 1.0)*(ztemp(2, i) - 1.0 /
                                            2.0) / 2.0;
273                                     Netatemp(Gauss, 3) = ztemp(1, i)*(ztemp
                                            (1, i) + 1.0)*(ztemp(2, i) + 1.0 /
                                            2.0) / 2.0;
274                                     Netatemp(Gauss, 4) = ztemp(1, i)*(ztemp
                                            (1, i) - 1.0)*(ztemp(2, i) + 1.0 /
                                            2.0) / 2.0;
275                                     Netatemp(Gauss, 5) = (1.0 - pow(ztemp(1,
                                             i), 2))*(ztemp(2, i) - 1.0 / 2.0);
276                                     Netatemp(Gauss, 6) = ztemp(1, i)*(ztemp
                                            (1, i) + 1.0)*(-ztemp(2, i));
277                                     Netatemp(Gauss, 7) = (1.0 - pow(ztemp(1,
                                             i), 2))*(ztemp(2, i) + 1.0 / 2.0);
278                                     Netatemp(Gauss, 8) = ztemp(1, i)*(ztemp
                                            (1, i) - 1.0)*(-ztemp(2, i));
279                                     Netatemp(Gauss, 9) = (1.0 - pow(ztemp(1,
                                             i), 2))*(-2.0*ztemp(2, i));
280                             }
281                             N = new Matrix(Ntemp); Nxi = new Matrix(Nxitemp)
                                    ; Neta = new Matrix(Netatemp); nodesCoord =
                                    new Matrix(nodesCoordtemp);
282                     }
283             }
284             else if (type == Tetahedra)
285             {
286
287             }
288
289 }
290
291 void Element::create_Gauss()
292 {
293             if (type == Linear)
294             {
295                     if (order == Lin)
296                     {
297                             Matrix zp(1, 1), wp(1, 1); zp(1, 1) = 0.0; wp(1,
```

```
                                1) = 2.0;
298                         z = new Matrix(zp);
299                         w = new Matrix(wp);
300                         Gauss = int(order);
301                 }
302                 else if (order == Quad)
303                 {
304                         Matrix zp(1, 2), wp(1, 2); zp(1, 1) = -1.0 /
                                sqrt(3); zp(1, 2) = 1.0 / sqrt(3); wp(1, 1)
                                = 1.0; wp(1, 2) = 1.0;
305                         z = new Matrix(zp);
306                         w = new Matrix(wp);
307                         Gauss = int(order);
308                 }
309         }
310         else if (type == Triangle)
311         {
312                 if (order == Lin)
313                 {
314                         Matrix zp(1, 2), wp(1, 2); zp(1, 1) = 1.0 / 3.0;
                                zp(1, 2) = 1.0 / 3.0; wp(1, 1) = 1.0;
315                         z = new Matrix(zp);
316                         w = new Matrix(wp);
317                         Gauss = int(order);
318                 }
319                 else if (order == Quad)
320                 {
321                         Matrix zp(3, 2), wp(1, 3); zp(1, 1) = 2.0 / 3.0;
                                zp(1, 2) = 1.0 / 6.0; zp(2, 1) = 1.0 / 6.0;
                                zp(2, 2) = 2.0 / 3.0; zp(3, 1) = 1.0 / 6.0;
                                zp(3, 2) = 1.0 / 6.0; wp(1, 1) = 1.0 / 3.0;
                                wp(1, 2) = 1.0 / 3.0; wp(1, 3) = 1.0/3.0;
322                         z = new Matrix(zp);
323                         w = new Matrix(wp);
324                         Gauss = int(order);
325                 }
326         }
327         else if (type == Quadrilateral)
328         {
329                 type = Linear;
330                 create_Gauss();
331                 Matrix ztemp(Gauss, Gauss), wtemp(Gauss, Gauss), etatemp
                        (Gauss, Gauss);
332                 Matrix zp(*z), wp(*w);
```

```
333                     ztemp.redimension(1, pow(Gauss,2));
334                     etatemp.redimension(1, pow(Gauss, 2));
335                     wtemp.redimension(1, pow(Gauss, 2));
336                     for (int i(1); i <= Gauss;i++)
337                             for (int j(1); j <= Gauss; j++)
338                             {
339                             ztemp(1,(i-1)*Gauss+ j) = zp(1,i);
340                             etatemp(1, (i - 1)*Gauss + j) = zp(1, j);
341                             wtemp(1, (i - 1)*Gauss + j) = wp(1, i);
342                             }
343                     zp.redimension(2, pow(Gauss, 2));
344                     for (int i(1); i <= pow(Gauss, 2); i++)
345                             for (int j(1); j <= 2; j++)
346                             {
347                             if (j == 1)
348                             {
349                                     zp(j, i) = ztemp(1, i);
350                             }
351                             if (j == 2)
352                             {
353                                     zp(j, i) = etatemp(1, i);
354                             }
355                             }
356                     z = new Matrix(zp);
357                     w = new Matrix(wp);
358                     type = Quadrilateral;
359                     Gauss = pow(Gauss, 2);
360             }
361             else if (type == Tetahedra)
362             {
363                     if (order == Lin)
364                     {
365
366                     }
367                     else if (order == Quad)
368                     {
369
370                     }
371             }
372 }
```

```
1 #include <stdlib.h>
2 #include <algorithm>
3 #include <iostream>
```

```
4  #include <cmath>
5  #include <memory>
6  #include <string>
7
8  using namespace std;
9
10 #define Verify(s)                                                    \
11         if (!(s))                                                    \
12         {                                                            \
13                 cerr << "Error: " << "No more memory available\n";  \
14                 exit(1);                                             \
15         }                                                            \
16
17 #define Error(ToStringStream)                                        \
18   {                                                                  \
19     cerr << "Error: " << ToStringStream << "\n";                     \
20     cin.get();                                                       \
21         exit(1);                                                     \
22   }
23
24
25 class Matrix
26 {
27 protected:
28         int nr, nc, size;
29         double*s;
30         Matrix(int m, int n, double* d)
31                 : nr(m), nc(n), s(d), size(m*n) {}
32         friend class vector;
33         void        release()
34         {
35                 s = 0; nr = nc = size = 0;
36         }
37         void        remove()
38         {
39                 free(s);
40                 release();
41         }
```

```
42 public:
43         Matrix(); //Init matrix
44         Matrix(int m, int n); //Init size matrix
45         Matrix(const Matrix& m); //Copy matrix
46         Matrix(const vector& r, int n, int m); //Matrix of a vector
47         ~Matrix();
48
49
50         int         nrows       ()              const { return nr; };
51         int         ncols       ()              const { return nc; };
52         int         isNull      ()              const { return size ==
               0; };
53         int         Size        ()                    const
               { return size; };
54         double*     Pointer     ()
                    { return s; };
55         int         sameSize(const Matrix& m) const      { return nr == m
               .nr && nc == m.nc; }
56         void        redimension(int m, int n);
57         void        plot        (string str)
58         {
59                 cout <<" "  <<str << " :"<<endl;
60                 for (int i(1); i <= size; i++)
61                 {
62                         if (i == 1){ cout << " " << endl; cout <<" "<< i
                                << "       "; }
63                         cout << s[i-1] <<" , ";
64                         if ((i%nc) == 0 && i != size){ cout << '\n' <<
                               endl; cout<<" " << i/nc+1<< "       "; }
65                 }
66                 cout << endl << endl;
67                 return;
68         }
69
70         // Operators
71
72         inline double& Matrix::operator() (int m, int n) const
73         {
74 #ifdef DEBUG_NORM
75                 if (m > nr || n > nc || m < 1 || n < 1)
76                         Error("Index r=" << m << ", c=" << n <<
77                                 " out of range. Matrix size: nr=" << nr
                                        << ", nc=" << nc);
78 #endif
```

```
79                  return s[nc*(m - 1) + n - 1];
80          }
81
82          inline double& Matrix::operator[] (int m) // m = nc*ic + ir
83          {
84  #ifdef DEBUG
85                  if (m >= size || m < 0)
86                          Error("Index m=" << m << " out of range in
                               Matrix operator [].");
87  #endif
88                  return s[m];
89          }
90
91
92
93          Matrix&     operator =  (const Matrix& m);
94          Matrix&     operator =  (const double fill_value);
95          Matrix&     operator +  ()                  const {return (Matrix&)
                *this;}
96          friend Matrix  operator +  (const Matrix& a, const Matrix& b);
97          Matrix&     product(const Matrix& m, register const double d);
98          Matrix&     add_product(const Matrix& m, register const double d)
                ;
99          Matrix&     sub_product(const Matrix& m, register const double d)
                ;
100         Matrix&     product(const Matrix& a, const Matrix& b, register
                const double d = 1.0);
101         Matrix&     add_product(const Matrix& a, const Matrix& b,
                register const double d = 1.0);
102         Matrix&     sub_product(const Matrix& a, const Matrix& b,
                register const double d = 1.0);
103         Matrix&     i(const Matrix& m, int verbo = 1); // Inverse
104         double      det() const;
105         double      trace() const;
106
107         Matrix&  operator += (const double d);
108         Matrix&  operator += (const Matrix& m);
109         Matrix&  operator -= (const double d);
110         Matrix&  operator -= (const Matrix& m);
111         Matrix&  operator *= (const double d);
112
113 };
```

```
1 #include "stdafx.h"
```

```
2  #include "Matrix.h"
3  #include "vector.h"
4  #include <stdlib.h>
5  #include <algorithm>
6  #include <iostream>
7  #include <cmath>
8  #include <memory>
9  #include <cstring>
10 #include <string.h>
11 inline void bcopy(const void *src, void *dst, int length)
12 {
13         memcpy(dst, src, length);
14 }
15
16
17
18 Matrix::Matrix()  //Init matrix
19 {
20         s = 0; ; nr = 0; nc = 0; size = 0;
21
22
23
24 }
25
26
27 Matrix::Matrix(int m, int n) //Init size matrix
28 {
29         nr=m; nc=n; size=m*n;
30         if (m<0) Error("Can't create a Matrix of size " << nr << "," <<
               nc);
31         Verify(s = (double*)malloc(size * sizeof(double)));
32 }
33 Matrix::Matrix(const Matrix& m)   //Copy matrix
34 {
35         nr=m.nrows(); nc=m.ncols(); size=nr*nc;
36         s = m.s;
37 }
38 Matrix::Matrix(const vector& r, int n, int m) //Matrix of a vector
39 {
40         nr = n; nc = m; size = n*m;
41         if (r.len != size) Error("Error: Cannot initialise Matrix.");
42         s = r.s;
43 }
44 Matrix::~Matrix()
```

```
45 {
46 }
47
48 void Matrix::redimension(int m, int n)
49 {
50         if (m<0 || n<0)
51                 Error("Can't redimension to a negative values");
52         nr = m; nc = n; size = m*n;
53 }
54
55
56 // OPERATORS
57
58 Matrix& Matrix::operator = (const Matrix& m)
59 {
60         if (this == &m) return *this;
61
62         if (size != m.size)
63         {
64                 //if (s) delete [] s;
65                 //Verify(s = new double [m.size]);
66                 if (s) free(s);
67                 Verify(s = (double*)malloc(m.size * sizeof(double)));
68         }
69
70         nr = m.nr;
71         nc = m.nc;
72         size = m.size;
73
74         bcopy(m.s, s, size * sizeof(double));
75
76         return *this;
77 }
78
79 Matrix& Matrix::operator = (register const double fill_value)
80 {
81         if (size == 0)
82                 Error("Can't fill with " << fill_value << " a zero size
                    Matrix");
83
84         double* up = s + (size & ~07);
85         register double* t = s;
86         while (t < up)
87         {
```

```
88                 *t++ = fill_value; *t++ = fill_value;
89                 *t++ = fill_value; *t++ = fill_value;
90                 *t++ = fill_value; *t++ = fill_value;
91                 *t++ = fill_value; *t++ = fill_value;
92         }
93         up = s + size;
94         while (t < up) *t++ = fill_value;
95
96         return *this;
97 }
98 Matrix& Matrix::product(const Matrix& m, register const double d)
99 {
100 #ifdef DEBUG
101         if (m.size == 0) Error("Can't multiply by " << d << " a zero
               size Matrix");
102         if (m.size != size) Error("Matrix::product wrong Matrix
               dimension");
103 #endif
104         double* up = s + (size & ~07);
105         register double* ns = s;
106         register const double* t = m.s;
107         while (ns < up)
108         {
109                 *ns++ = *t++ * d; *ns++ = *t++ * d; *ns++ = *t++ * d; *
                       ns++ = *t++ * d;
110                 *ns++ = *t++ * d; *ns++ = *t++ * d; *ns++ = *t++ * d; *
                       ns++ = *t++ * d;
111         }
112         up = s + size;
113         while (ns < up) *ns++ = *t++ * d;
114
115         return *this;
116 }
117
118 Matrix& Matrix::add_product(const Matrix& m, register const double d)
119 {
120 #ifdef DEBUG
121         if (m.size == 0) Error("Can't multiply by " << d << " a zero
               size Matrix");
122         if (m.size != size) Error("Matrix::add_product wrong Matrix
               dimension");
123 #endif
124         double* up = s + (size & ~07);
125         register double* ns = s;
```

```
126         register const double* t = m.s;
127         while (ns < up)
128         {
129                 *ns++ += *t++ * d; *ns++ += *t++ * d; *ns++ += *t++ * d;
                        *ns++ += *t++ * d;
130                 *ns++ += *t++ * d; *ns++ += *t++ * d; *ns++ += *t++ * d;
                        *ns++ += *t++ * d;
131         }
132         up = s + size;
133         while (ns < up) *ns++ += *t++ * d;
134
135         return *this;
136 }
137
138 Matrix& Matrix::sub_product(const Matrix& m, register const double d)
139 {
140 #ifdef DEBUG
141         if (m.size == 0) Error("Can't multiply by " << d << " a zero
                size Matrix");
142         if (m.size != size) Error("Matrix::sub_product wrong Matrix
                dimension");
143 #endif
144         double* up = s + (size & ~07);
145         register double* ns = s;
146         register const double* t = m.s;
147         while (ns < up)
148         {
149                 *ns++ -= *t++ * d; *ns++ -= *t++ * d; *ns++ -= *t++ * d;
                        *ns++ -= *t++ * d;
150                 *ns++ -= *t++ * d; *ns++ -= *t++ * d; *ns++ -= *t++ * d;
                        *ns++ -= *t++ * d;
151         }
152         up = s + size;
153         while (ns < up) *ns++ -= *t++ * d;
154
155         return *this;
156 }
157
158 Matrix& Matrix::product(const Matrix& a, const Matrix& b, register const
        double d)
159 {
160 #ifdef DEBUG
161         if (a.size == 0 && b.size == 0) Error("Can't multiply zero sized
                matrices");
```

```
162         if (a.nc != b.nr)
163                 Error("Incompatible dimensions for multiply -A-\n" <<
164                         "\tMatrix 1 : nrows=" << a.nr << ", ncols=" << a
                                .nc << "\n" <<
165                         "\tMatrix 2 : nrows=" << b.nr << ", ncols=" << b
                                .nc);
166         if (a.nr != nr) Error("Matrix::product wrong Matrix row
                dimension");
167         if (b.nc != nc) Error("Matrix::product wrong Matrix column
                dimension");
168 #endif
169
170         double* ns = s;
171         double* aS = a.s;
172
173         int anc = a.nc;
174         int bnc = b.nc;
175         int nr = a.nr;
176         while (nr--)
177         {
178                 const double * const up1 = aS + (anc & ~07);
179                 const double * const up2 = aS + anc;
180
181                 double* bS = b.s;
182
183                 int nc = bnc;
184                 while (nc--)
185                 {
186                         register const double* bs = bS++;
187                         register const double* as = aS;
188                         register double res = 0;
189                         while (as<up1)
190                         {
191                                 res += *as++ * *bs; bs += bnc; res += *
                                        as++ * *bs; bs += bnc;
192                                 res += *as++ * *bs; bs += bnc; res += *
                                        as++ * *bs; bs += bnc;
193                                 res += *as++ * *bs; bs += bnc; res += *
                                        as++ * *bs; bs += bnc;
194                                 res += *as++ * *bs; bs += bnc; res += *
                                        as++ * *bs; bs += bnc;
195                         }
196                         if (as < aS + (anc & ~03))
197                         {
```

```
198                                 res += *as++ * *bs; bs += bnc; res += *
                                        as++ * *bs; bs += bnc;
199                                 res += *as++ * *bs; bs += bnc; res += *
                                        as++ * *bs; bs += bnc;
200                         }
201                         while (as<up2) { res += *as++ * *bs; bs += bnc;
                                }
202                         if (d == 1.0) *ns++ = res;
203                         else  *ns++ = d*res;
204                 }
205                 aS += a.nc;
206         }
207
208         return *this;
209 }
210
211 Matrix& Matrix::add_product(const Matrix& a, const Matrix& b, register
      const double d)
212 {
213 #ifdef DEBUG
214         if (a.size == 0 && b.size == 0) Error("Can't multiply zero sized
              matrices");
215         if (a.nc != b.nr)
216                 Error("Incompatible dimensions for multiply -A-\n" <<
217                         "\tMatrix 1 : nrows=" << a.nr << ", ncols=" << a
                            .nc << "\n" <<
218                         "\tMatrix 2 : nrows=" << b.nr << ", ncols=" << b
                            .nc);
219         if (a.nr != nr) Error("Matrix::product wrong Matrix row
              dimension");
220         if (b.nc != nc) Error("Matrix::product wrong Matrix column
              dimension");
221 #endif
222
223         double* ns = s;
224         double* aS = a.s;
225
226         int anc = a.nc;
227         int bnc = b.nc;
228         int nr = a.nr;
229         while (nr--)
230         {
231                 const double * const up1 = aS + (anc & ~07);
232                 const double * const up2 = aS + anc;
```

```
233
234                 double* bS = b.s;
235
236                 int nc = bnc;
237                 while (nc--)
238                 {
239                         register const double* bs = bS++;
240                         register const double* as = aS;
241                         register double res = 0;
242                         while (as<up1)
243                         {
244                                 res += *as++ * *bs; bs += bnc; res += *
                                        as++ * *bs; bs += bnc;
245                                 res += *as++ * *bs; bs += bnc; res += *
                                        as++ * *bs; bs += bnc;
246                                 res += *as++ * *bs; bs += bnc; res += *
                                        as++ * *bs; bs += bnc;
247                                 res += *as++ * *bs; bs += bnc; res += *
                                        as++ * *bs; bs += bnc;
248                         }
249                         if (as < aS + (anc & ~03))
250                         {
251                                 res += *as++ * *bs; bs += bnc; res += *
                                        as++ * *bs; bs += bnc;
252                                 res += *as++ * *bs; bs += bnc; res += *
                                        as++ * *bs; bs += bnc;
253                         }
254                         while (as<up2) { res += *as++ * *bs; bs += bnc;
                                }
255                         if (d == 1.0) *ns++ += res;
256                         else  *ns++ += d*res;
257                 }
258                 aS += a.nc;
259         }
260
261         return *this;
262 }
263
264 Matrix& Matrix::sub_product(const Matrix& a, const Matrix& b, register
        const double d)
265 {
266 #ifdef DEBUG
267         if (a.size == 0 && b.size == 0) Error("Can't multiply zero sized
              matrices");
```

```
268         if (a.nc != b.nr)
269                 Error("Incompatible dimensions for multiply -A-\n" <<
270                         "\tMatrix 1 : nrows=" << a.nr << ", ncols=" << a
                                .nc << "\n" <<
271                         "\tMatrix 2 : nrows=" << b.nr << ", ncols=" << b
                                .nc);
272         if (a.nr != nr) Error("Matrix::product wrong Matrix row
                dimension");
273         if (b.nc != nc) Error("Matrix::product wrong Matrix column
                dimension");
274 #endif
275
276         double* ns = s;
277         double* aS = a.s;
278
279         int anc = a.nc;
280         int bnc = b.nc;
281         int nr = a.nr;
282         while (nr--)
283         {
284                 const double * const up1 = aS + (anc & ~07);
285                 const double * const up2 = aS + anc;
286
287                 double* bS = b.s;
288
289                 int nc = bnc;
290                 while (nc--)
291                 {
292                         register const double* bs = bS++;
293                         register const double* as = aS;
294                         register double res = 0;
295                         while (as<up1)
296                         {
297                                 res += *as++ * *bs; bs += bnc; res += *
                                        as++ * *bs; bs += bnc;
298                                 res += *as++ * *bs; bs += bnc; res += *
                                        as++ * *bs; bs += bnc;
299                                 res += *as++ * *bs; bs += bnc; res += *
                                        as++ * *bs; bs += bnc;
300                                 res += *as++ * *bs; bs += bnc; res += *
                                        as++ * *bs; bs += bnc;
301                         }
302                         if (as < aS + (anc & ~03))
303                         {
```

```
304                                 res += *as++ * *bs; bs += bnc; res += *
                                        as++ * *bs; bs += bnc;
305                                 res += *as++ * *bs; bs += bnc; res += *
                                        as++ * *bs; bs += bnc;
306                         }
307                         while (as<up2) { res += *as++ * *bs; bs += bnc;
                                }
308                         if (d == 1.0) *ns++ -= res;
309                         else  *ns++ -= d*res;
310                 }
311                 aS += a.nc;
312         }
313
314         return *this;
315 }
316
317 #define m(r,c) s[nc*(r-1) + c-1]
318
319 #define VerifyZero(d) if((d)==0) \
320 Error("Inverse of a singular Matrix. Size = (" << nr << " X " << nr <<")
        ")
321
322 #define VerifyZeroVerbo(d,v) if((d)==0) { \
323     if(verbo) { Error("Inverse of a singular Matrix. Size = (" << nr <<
            " X " << nr <<")") } \
324         else { remove(); return *this; } }
325
326
327
328 Matrix& Matrix::i(const Matrix& n, int verbo) // Inverse
329 {
330         if (size == 0) {
331                 if (verbo) { Error("Can't compute the inverse of a zero
                        size Matrix"); }
332                 else { remove(); return *this; }
333         }
334         if (nr != nc) {
335                 if (verbo) {
336                         Error("Can't compute the inverse of a
                            rectangular Matrix\n" <<
337                                 "\tnrows =" << nr << ", ncols =" << nc);
338                 }
339                 else { remove(); return *this; }
340         }
```

```
341         double deter, denom;
342
343         switch (nr)
344         {
345         case 1:
346         {
347                 VerifyZeroVerbo(deter = n(1, 1), verbo);
348                 (*this)(1, 1) = 1 / deter;
349                 break;
350         }
351         case 2:
352         {
353                 VerifyZeroVerbo(deter = n(1, 1)*n(2, 2) - n(2, 1)*n(1,
                        2), verbo);
354                 denom = 1 / deter;
355                 (*this)(1, 1) = n(2, 2)*denom;
356                 (*this)(1, 2) = -n(1, 2)*denom;
357                 (*this)(2, 1) = -n(2, 1)*denom;
358                 (*this)(2, 2) = n(1, 1)*denom;
359                 break;
360         }
361         case 3:
362         {
363                 double t1 = n(2, 2)*n(3, 3) - n(3, 2)*n(2, 3);
364                 double t2 = n(3, 2)*n(1, 3) - n(1, 2)*n(3, 3);
365                 double t3 = n(1, 2)*n(2, 3) - n(2, 2)*n(1, 3);
366                 VerifyZeroVerbo(deter = n(1, 1)*t1 + n(2, 1)*t2 + n(3,
                        1)*t3, verbo);
367                 denom = 1 / deter;
368                 (*this)(1, 1) = t1*denom;
369                 (*this)(1, 2) = t2*denom;
370                 (*this)(1, 3) = t3*denom;
371                 (*this)(2, 1) = (n(3, 1)*n(2, 3) - n(2, 1)*n(3, 3))*
                        denom;
372                 (*this)(2, 2) = (n(1, 1)*n(3, 3) - n(3, 1)*n(1, 3))*
                        denom;
373                 (*this)(2, 3) = (n(2, 1)*n(1, 3) - n(1, 1)*n(2, 3))*
                        denom;
374                 (*this)(3, 1) = (n(2, 1)*n(3, 2) - n(3, 1)*n(2, 2))*
                        denom;
375                 (*this)(3, 2) = (n(1, 2)*n(3, 1) - n(1, 1)*n(3, 2))*
                        denom;
376                 (*this)(3, 3) = (n(1, 1)*n(2, 2) - n(2, 1)*n(1, 2))*
                        denom;
```

```
377                 break;
378             }
379             case 4:
380             {
381                 double t1 = n(2, 4)*(n(3, 2)*n(4, 3) - n(3, 3)*n(4, 2))
382                         + n(2, 3)*(n(3, 4)*n(4, 2) - n(3, 2)*n(4, 4))
383                         + n(2, 2)*(n(3, 3)*n(4, 4) - n(3, 4)*n(4, 3));
384                 double t2 = n(1, 4)*(n(3, 3)*n(4, 2) - n(3, 2)*n(4, 3))
385                         + n(1, 3)*(n(3, 2)*n(4, 4) - n(3, 4)*n(4, 2))
386                         + n(1, 2)*(n(3, 4)*n(4, 3) - n(3, 3)*n(4, 4));
387                 double t3 = n(1, 4)*(n(2, 2)*n(4, 3) - n(2, 3)*n(4, 2))
388                         + n(1, 3)*(n(2, 4)*n(4, 2) - n(2, 2)*n(4, 4))
389                         + n(1, 2)*(n(2, 3)*n(4, 4) - n(2, 4)*n(4, 3));
390                 double t4 = n(1, 4)*(n(2, 3)*n(3, 2) - n(2, 2)*n(3, 3))
391                         + n(1, 3)*(n(2, 2)*n(3, 4) - n(2, 4)*n(3, 2))
392                         + n(1, 2)*(n(2, 4)*n(3, 3) - n(2, 3)*n(3, 4));
393                 VerifyZeroVerbo(deter = n(1, 1)*t1 + n(2, 1)*t2 + n(3,
                        1)*t3 + n(4, 1)*t4, verbo);
394                 denom = 1 / deter;
395                 (*this)(1, 1) = t1*denom;
396                 (*this)(1, 2) = t2*denom;
397                 (*this)(1, 3) = t3*denom;
398                 (*this)(1, 4) = t4*denom;
399                 (*this)(2, 1) = (n(2, 4)*(n(3, 3)*n(4, 1) - n(3, 1)*n(4,
                        3))
400                         + n(2, 3)*(n(3, 1)*n(4, 4) - n(3, 4)*n(4, 1))
401                         + n(2, 1)*(n(3, 4)*n(4, 3) - n(3, 3)*n(4, 4)))*
                            denom;
402                 (*this)(2, 2) = (n(1, 4)*(n(3, 1)*n(4, 3) - n(3, 3)*n(4,
                        1))
403                         + n(1, 3)*(n(3, 4)*n(4, 1) - n(3, 1)*n(4, 4))
404                         + n(1, 1)*(n(3, 3)*n(4, 4) - n(3, 4)*n(4, 3)))*
                            denom;
405                 (*this)(2, 3) = (n(1, 4)*(n(2, 3)*n(4, 1) - n(2, 1)*n(4,
                        3))
406                         + n(1, 3)*(n(2, 1)*n(4, 4) - n(2, 4)*n(4, 1))
407                         + n(1, 1)*(n(2, 4)*n(4, 3) - n(2, 3)*n(4, 4)))*
                            denom;
408                 (*this)(2, 4) = (n(1, 4)*(n(2, 1)*n(3, 3) - n(2, 3)*n(3,
                        1))
409                         + n(1, 3)*(n(2, 4)*n(3, 1) - n(2, 1)*n(3, 4))
410                         + n(1, 1)*(n(2, 3)*n(3, 4) - n(2, 4)*n(3, 3)))*
                            denom;
411                 (*this)(3, 1) = (n(2, 4)*(n(3, 1)*n(4, 2) - n(3, 2)*n(4,
```

```
                1))
412                 + n(2, 2)*(n(3, 4)*n(4, 1) - n(3, 1)*n(4, 4))
413                 + n(2, 1)*(n(3, 2)*n(4, 4) - n(3, 4)*n(4, 2)))*
                        denom;
414             (*this)(3, 2) = (n(1, 4)*(n(3, 2)*n(4, 1) - n(3, 1)*n(4,
                2))
415                 + n(1, 2)*(n(3, 1)*n(4, 4) - n(3, 4)*n(4, 1))
416                 + n(1, 1)*(n(3, 4)*n(4, 2) - n(3, 2)*n(4, 4)))*
                        denom;
417             (*this)(3, 3) = (n(1, 4)*(n(2, 1)*n(4, 2) - n(2, 2)*n(4,
                1))
418                 + n(1, 2)*(n(2, 4)*n(4, 1) - n(2, 1)*n(4, 4))
419                 + n(1, 1)*(n(2, 2)*n(4, 4) - n(2, 4)*n(4, 2)))*
                        denom;
420             (*this)(3, 4) = (n(1, 4)*(n(2, 2)*n(3, 1) - n(2, 1)*n(3,
                2))
421                 + n(1, 2)*(n(2, 1)*n(3, 4) - n(2, 4)*n(3, 1))
422                 + n(1, 1)*(n(2, 4)*n(3, 2) - n(2, 2)*n(3, 4)))*
                        denom;
423             (*this)(4, 1) = (n(2, 3)*(n(3, 2)*n(4, 1) - n(3, 1)*n(4,
                2))
424                 + n(2, 2)*(n(3, 1)*n(4, 3) - n(3, 3)*n(4, 1))
425                 + n(2, 1)*(n(3, 3)*n(4, 2) - n(3, 2)*n(4, 3)))*
                        denom;
426             (*this)(4, 2) = (n(1, 3)*(n(3, 1)*n(4, 2) - n(3, 2)*n(4,
                1))
427                 + n(1, 2)*(n(3, 3)*n(4, 1) - n(3, 1)*n(4, 3))
428                 + n(1, 1)*(n(3, 2)*n(4, 3) - n(3, 3)*n(4, 2)))*
                        denom;
429             (*this)(4, 3) = (n(1, 3)*(n(2, 2)*n(4, 1) - n(2, 1)*n(4,
                2))
430                 + n(1, 2)*(n(2, 1)*n(4, 3) - n(2, 3)*n(4, 1))
431                 + n(1, 1)*(n(2, 3)*n(4, 2) - n(2, 2)*n(4, 3)))*
                        denom;
432             (*this)(4, 4) = (n(1, 3)*(n(2, 1)*n(3, 2) - n(2, 2)*n(3,
                1))
433                 + n(1, 2)*(n(2, 3)*n(3, 1) - n(2, 1)*n(3, 3))
434                 + n(1, 1)*(n(2, 2)*n(3, 3) - n(2, 3)*n(3, 2)))*
                        denom;
435             break;
436         }
437         default:
438         {
439             *this = n;
```

```
440                 for (int n = 1; n <= nr; n++)
441                 {
442                         double d = (*this)(n, n); VerifyZero(d);
443                         for (int k = 1; k <= nr; k++) (*this)(n, k) /= -
                                d;
444                         for (int i = 1; i <= nr; i++)
445                         {
446                                 if (n != i)
447                                         for (int j = 1; j <= nr; j++)
448                                                 if (j != n) (*this)(i, j
                                                        ) += (*this)(i, n)
                                                        *(*this)(n, j);
449                                 (*this)(i, n) /= d;
450                         }
451                         (*this)(n, n) = 1 / d;
452                 }
453                 break;
454         }
455         }
456         return *this;
457 }
458
459 double Matrix::det() const
460 {
461         if (size == 0)
462                 Error("Can't compute the determinant of a zero size
                        Matrix");
463         if (nr != nc)
464                 Error("Can't compute the determinant of a rectangular
                        Matrix\n" <<
465                         "\tnrows =" << nr << ", ncols =" << nc);
466
467         double deter;
468
469         switch (nr)
470         {
471         case 1:
472         {
473                 deter = m(1, 1);
474                 break;
475         }
476         case 2:
477         {
478                 deter = m(1, 1)*m(2, 2) - m(2, 1)*m(1, 2);
```

```
479                 break;
480         }
481         case 3:
482         {
483                 deter = m(1, 1)*(m(2, 2)*m(3, 3) - m(3, 2)*m(2, 3))
484                         + m(2, 1)*(m(3, 2)*m(1, 3) - m(1, 2)*m(3, 3))
485                         + m(3, 1)*(m(1, 2)*m(2, 3) - m(2, 2)*m(1, 3));
486                 break;
487         }
488         case 4:
489         {
490                 deter = m(1, 1)*(m(2, 4)*(m(3, 2)*m(4, 3) - m(3, 3)*m(4,
                        2))
491                             + m(2, 3)*(m(3, 4)*m(4, 2) - m(3, 2)*m(4, 4))
492                             + m(2, 2)*(m(3, 3)*m(4, 4) - m(3, 4)*m(4, 3)))
493                             + m(2, 1)*(m(1, 4)*(m(3, 3)*m(4, 2) - m(3, 2)*m
                                (4, 3))
494                                     + m(1, 3)*(m(3, 2)*m(4, 4) - m(3, 4)*m
                                        (4, 2))
495                                     + m(1, 2)*(m(3, 4)*m(4, 3) - m(3, 3)*m
                                        (4, 4)))
496                             + m(3, 1)*(m(1, 4)*(m(2, 2)*m(4, 3) - m(2, 3)*m
                                (4, 2))
497                                     + m(1, 3)*(m(2, 4)*m(4, 2) - m(2, 2)*m
                                        (4, 4))
498                                     + m(1, 2)*(m(2, 3)*m(4, 4) - m(2, 4)*m
                                        (4, 3)))
499                             + m(4, 1)*(m(1, 4)*(m(2, 3)*m(3, 2) - m(2, 2)*m
                                (3, 3))
500                                     + m(1, 3)*(m(2, 2)*m(3, 4) - m(2, 4)*m
                                        (3, 2))
501                                     + m(1, 2)*(m(2, 4)*m(3, 3) - m(2, 3)*m
                                        (3, 4)));
502                 break;
503         }
504         default:
505         {
506                 Error("Not implemented yet determinant for a Matrix
                        greater than 4");
507                 deter = 0;
508                 break;
509         }
510         }
511
```

```
512         return deter;
513 }
514 double Matrix::trace() const
515 {
516         if (size == 0)
517                 Error("Can't compute the trace of a zero size Matrix");
518         if (nr != nc)
519                 Error("Can't compute the trace of a non square Matrix");
520
521         register double tr = 0;
522         register double* t = s;
523         double * const up = s + size;
524         const int w = nc + 1;
525
526         while (t<up)
527         {
528                 tr += *t;
529                 t += w;
530         }
531
532         return tr;
533 }
534
535 Matrix& Matrix::operator += (register const double d)
536 {
537         if (size == 0)
538                 Error("Can't add " << d << " to a zero size Matrix");
539
540         double* up = s + (size & ~07);
541         register double* t = s;
542         while (t < up)
543         {
544                 *t++ += d; *t++ += d; *t++ += d; *t++ += d;
545                 *t++ += d; *t++ += d; *t++ += d; *t++ += d;
546         }
547         up = s + size;
548         while (t < up) *t++ += d;
549
550         return *this;
551 }
552
553 Matrix& Matrix::operator += (const Matrix& m)
554 {
555         if (size == 0 && m.size != 0) return *this = m; // this is valid
```

```
                only for +=
556
557         if (size == 0 && m.size == 0)
558                 Error("Can't add zero sized matrices");
559         if (!sameSize(m))
560                 Error("Can't add two Matrix with diferent sizes\n" <<
561                         "\tMatrix 1 : nrows=" << nr << ", ncols=" << nc
                                << "\n" <<
562                         "\tMatrix 2 : nrows=" << m.nr << ", ncols=" << m
                                .nc);
563
564         double* up = s + (size & ~07);
565         register double* t = s;
566         register double* u = m.s;
567         while (t < up)
568         {
569                 *t++ += *u++; *t++ += *u++; *t++ += *u++; *t++ += *u++;
570                 *t++ += *u++; *t++ += *u++; *t++ += *u++; *t++ += *u++;
571         }
572         up = s + size;
573         while (t < up) *t++ += *u++;
574
575         return *this;
576 }
577
578 Matrix& Matrix::operator -= (register const double d)
579 {
580         if (size == 0)
581                 Error("Can't subtract " << d << " from a zero size
                            Matrix");
582
583         double* up = s + (size & ~07);
584         register double* t = s;
585         while (t < up)
586         {
587                 *t++ -= d; *t++ -= d; *t++ -= d; *t++ -= d;
588                 *t++ -= d; *t++ -= d; *t++ -= d; *t++ -= d;
589         }
590         up = s + size;
591         while (t < up) *t++ -= d;
592
593         return *this;
594 }
595 Matrix& Matrix::operator -= (const Matrix& m)
```

```
596 {
597         if (size == 0 && m.size == 0)
598                 Error("Can't subtract zero sized matrices");
599         if (!sameSize(m))
600                 Error("Can't subtract two Matrix with diferent sizes\n"
                        <<
601                         "\tMatrix 1 : nrows=" << nr << ", ncols=" << nc
                            << "\n" <<
602                         "\tMatrix 2 : nrows=" << m.nr << ", ncols=" << m
                            .nc);
603
604         double* up = s + (size & ~07);
605         register double* t = s;
606         register double* u = m.s;
607         while (t < up)
608         {
609                 *t++ -= *u++; *t++ -= *u++; *t++ -= *u++; *t++ -= *u++;
610                 *t++ -= *u++; *t++ -= *u++; *t++ -= *u++; *t++ -= *u++;
611         }
612         up = s + size;
613         while (t < up) *t++ -= *u++;
614
615         return *this;
616 }
617 Matrix& Matrix::operator *= (register const double d)
618 {
619         if (size == 0)
620                 Error("Can't multiply by " << d << " a zero size Matrix"
                        );
621
622         double* up = s + (size & ~07);
623         register double* t = s;
624         while (t < up)
625         {
626                 *t++ *= d; *t++ *= d; *t++ *= d; *t++ *= d;
627                 *t++ *= d; *t++ *= d; *t++ *= d; *t++ *= d;
628         }
629         up = s + size;
630         while (t < up) *t++ *= d;
631
632         return *this;
633 }
634
635 Matrix operator + (const Matrix& a, const Matrix& b)
```

```
636 {
637         if (a.size == 0 && b.size == 0)
638                 Error("Can't add zero sized matrices");
639         if (!a.sameSize(b))
640                 Error("Can't add two Matrix with diferent sizes\n" <<
641                         "\tMatrix 1 : nrows=" << a.nr << ", ncols=" << a
                            .nc << "\n" <<
642                         "\tMatrix 2 : nrows=" << b.nr << ", ncols=" << b
                            .nc);
643
644         //double* news = new double [a.size]; Verify(news);
645         double* news = (double*)malloc(a.size * sizeof(double)); Verify(
                news);
646         double* up = news + (a.size & ~07);
647         register double* ns = news;
648         register double* as = a.s;
649         register double* bs = b.s;
650         while (ns < up)
651         {
652                 *ns++ = *as++ + *bs++; *ns++ = *as++ + *bs++;
653                 *ns++ = *as++ + *bs++; *ns++ = *as++ + *bs++;
654                 *ns++ = *as++ + *bs++; *ns++ = *as++ + *bs++;
655                 *ns++ = *as++ + *bs++; *ns++ = *as++ + *bs++;
656         }
657         up = news + a.size;
658         while (ns < up) *ns++ = *as++ + *bs++;
659
660         return Matrix(a.nr, a.nc, news);
661 }
```

```
 1
 2 #include <stdlib.h>
 3 #include <algorithm>
 4 #include <iostream>
 5 #include <cmath>
 6 #include <memory>
 7 #include <string>
 8
 9 using namespace std;
10
11 #define Verify(s)                                                    \
12         if (!(s))
                                                                     \
13         {
```

```
                                                                    \
14                  cerr << "Error: " << "No more memory available\n";
                        \
15                  exit(1);
                                                                    \
16          }
                                                                    \
17
18
19  #define Error(ToStringStream)                                    \
20    {                                                              \
21      cerr << "Error: " << ToStringStream << "\n";                  \
22          cin.get();
                                                          \
23      exit(1);                                                     \
24    }
25
26  class vector
27  {
28  private:
29          int unsigned len;
30          double* s;
31          friend class Matrix;
32  public:
33          // CONSTRUCTORS , COPY , DESTRUCTOR
34          vector();
                // Init with len=0, s=0
35          vector(int unsigned a);                          // Init with len
                =a and s=rand
36          vector(int unsigned a, double b);
37          vector(int unsigned a,double* b);    // Init with an array
38          vector(int unsigned a, double b, int unsigned i, int unsigned I)
                ;                // Init with len=a and s=scalar
39      vector(const vector& v);                                     // Copy
            vector
40          vector(const Matrix& v);                         // Copy Matrix
41          ~vector();
42          vector&         fill(double val, unsigned from = 1, unsigned n =
                0);
43          virtual vector& resize(unsigned newlen, double val = 0);
44          void            plot(string str)
45          {
46                  cout << " " << str << " :" << endl;
47                  for (int i(1); i <= len; i++)
```

```
48                {
49                        cout << " " << endl; cout << " " << i << "
                                  ";
50
51                        cout << s[i - 1] << endl;
52
53                }
54                cout << endl << endl;
55                return;
56        }
57
58
59        // OPERATORS
60        vector&         operator =  (const vector& v);
61        vector&         operator =  (const Matrix& v);
62        vector&         operator =  (double fill_value);
63        inline double& operator ()      (unsigned n)
64        {
65        #ifdef DEBUG
66                if (n>len || n<1) Error("Index out of range in ReaVec.
                     Length=" << len << " index=" << n);
67        #endif
68                return s[n - 1];
69        }
70        inline double& operator [] (unsigned n)
71        {
72        #ifdef DEBUG
73                if (n >= len || n<0)    Error("Index out of range in
                     ReaVec[]. Length=" << len << " index=" << n);
74        #endif
75                return s[n];
76        }
77        int      operator == (double d);
78
79        int      operator != (double d);
80
81        friend   int      operator == (const vector& a, const vector& b);
82
83
84        friend int      operator != (const vector& a, const vector& b);
85        // vector by scalar -> vector operations
86
87        friend vector&                  operator +  (const vector& a,
             double  b);
```

```
88          friend vector&                  operator - (const vector& a,
                double b);
89
90          friend vector&                  operator * (const vector& a,
                double b);
91          friend vector&                  operator / (const vector& a,
                double b);
92
93          // vector by vector -> vector operations
94
95          friend vector&                  operator + (const vector& a,
                const vector& b);
96          friend vector&                  operator - (const vector& a,
                const vector& b);
97          vector&                                 vectorial       (const
                vector& a, const vector& b);
98          friend double                   dot                     (const
                vector& a, const vector& b);
99
100         // vector by matrix -> vector operations
101
102         vector&         product         (const Matrix& a, const vector&
                b);
103         vector&         add_product     (const Matrix& a, const vector&
                b);
104         vector&         sub_product     (const Matrix& a, const vector&
                b);
105
106         // GET FUNCTIONS
107         double* vector::S() { return s; };
108         int vector::L() { return len; };
109
110 };
```

```
1  #include "stdafx.h"
2  #include "vector.h"
3  #include "Matrix.h"
4  #include <stdlib.h>
5  #include <algorithm>
6  #include <iostream>
7  #include <cmath>
8  #include <memory>
9  #include <cstring>
10 #include <cstdlib>
```

```
11
12 using namespace std;
13
14 // CONSTRUCTORS
15 vector::vector()
16 {
17         len = 0;          s = 0;
18 }
19 vector::vector(int unsigned a)
20 {
21         len = a;
22         Verify(s = (double*)malloc(len * sizeof(double)));
23 }
24 vector::vector(unsigned a, double b)
25 {
26         len=a;
27         if (a<0) Error("Can't create a vector of size " << a);
28         Verify(s = (double*)malloc(len * sizeof(double)));
29         fill(b);
30 }
31 vector::vector(unsigned a, double* b)
32 {
33         len = a;
34         s = b;
35 }
36 vector::vector(int unsigned a, double b,int unsigned i , int unsigned I
       = 0)
37 {
38         len = a;
39         s = new double[a];
40         if (a <= i)
41         {
42                 Error("Error: length " << a-1 << " < " << i << endl)
43
44         }
45         else if (a <= I)
46         {
47                 Error("Error: length " << a-1 << " < " << I << endl)
48         }
49         else if (i<=I)
50         {
51                 for (int j(i); j<=I; j++)
52                 {
53                         s[j] = b;
```

```
54                      }
55              }
56              else if (i > I)
57              {
58                      for (int j(I); j <= i; j++)
59                      {
60                              s[j] = b;
61                      }
62              }
63      }
64
65      vector::vector(const vector& v) : len(v.len)
66      {
67              Verify(s = (double*)malloc(len * sizeof(double)));
68              memcpy(s, v.s, len * sizeof(double));
69      }
70      // DESTRUCTOR
71      vector::~vector()
72      {
73
74      }
75
76      vector& vector::fill(double val, unsigned from, unsigned n)
77      {
78              //cout << from << "\t" << n << "\t" << len << endl;
79              if (from < 0) from += len;
80              else from--;
81              if (n == 0) n = len - from;
82              unsigned to = from + n - 1;
83              //cout << from << "\t" << n << "\t" << to << endl;
84              if (from > to || to >= len)
85                      Error("Index " << to << " out of range.");
86
87              double* t = s + from;
88              double* up = s + to;
89              while (t <= up) *t++ = val;
90
91              return *this;
92      }
93
94      vector& vector::resize(unsigned newl, double val)
95      {
96              if (newl == 0)
97              {
```

```
98                  if (s) delete[] s;
99                  s = 0;
100         }
101         else if (s && newl>len)
102         {
103                 double* temp = s;                   // from RAMSAN
104                 s = new double[newl];
105                 Verify(s);
106                 for (unsigned i = 0; i<len; i++) s[i] = temp[i];
107                 delete[] temp;
108         }
109         else if (s && newl<len)
110         {
111                 double* temp = s;                   // from RAMSAN
112                 s = new double[newl];
113                 Verify(s);
114                 for (unsigned i = 0; i<newl; i++) s[i] = temp[i];
115                 delete[] temp;
116         }
117         else if (!s)
118         {
119                 s = new double[newl];
120                 Verify(s);
121         }
122
123         if (newl > len)
124         {
125                 double* t = s + len;
126                 double* up = s + newl;
127                 while (t < up) *t++ = val;
128         }
129         len = newl;
130
131         return *this;
132 }
133 // OPERATORS
134
135 vector&         vector::operator =  (const vector& v)
136 {
137         if (this == &v) return *this;
138         if (len != v.len)
139         {
140                 free(s);
141                 len = v.len;
```

```
142                 Verify(s = (double*)malloc(len * sizeof(double)));
143         }
144         memcpy(s, v.s, len * sizeof(double));
145
146         return *this;
147 }
148 vector&         vector::operator = (const Matrix& v)
149 {
150         if (v.ncols() > 1 || v.nrows() > 1)
151         {
152                 Error("Error: vector can't store the matrix. \n Rows: "
                        << v.nrows() << "\n Columns: " << v.ncols() << endl)
153         }
154         else
155         {
156                 len = v.size;
157                 free(s);
158                 s = v.s;
159                 return *this;
160         }
161 }
162
163 vector&         vector::operator = (double fill_value)
164 {
165         if (len == 0)
166                 Error("Can't fill with " << fill_value << " a zero
                        lenght vector.");
167
168         double* up = s + (len & ~07);
169         register double* t = s;
170         while (t < up)
171         {
172                 *t++ = fill_value; *t++ = fill_value;
173                 *t++ = fill_value; *t++ = fill_value;
174                 *t++ = fill_value; *t++ = fill_value;
175                 *t++ = fill_value; *t++ = fill_value;
176         }
177         up = s + len;
178         while (t < up) *t++ = fill_value;
179
180         return *this;
181 }
182
183 // Comparison
```

```
184
185 int       vector::operator == (double d)
186 {
187         register double* t = s;
188         double* up = s + len;
189         while (t < up) if (*t++ != d) return 0;
190
191         return 1;
192 }
193
194 int       operator == (const vector& a, const vector& b)
195         {
196                 {
197                         if (a.len != b.len) return 0;
198
199                         register double* u = b.s;
200                         register double* t = a.s;
201                         double* up = a.s + a.len;
202                         while (t < up) if (*t++ != *u++) return 0;
203
204                         return 1;
205                 }
206         }
207
208
209 int       vector::operator != (double d)
210 {
211         return !(*this == d);
212 }
213 int       operator != (const vector& a, const vector& b)
214 {
215         return !(a == b);
216 }
217
218 // vector by scalar -> vector operations
219
220 vector&                 operator +  (const vector& a, double  b)
221 {
222         double* news = (double*)malloc(a.len * sizeof(double)); Verify(
            news);
223         double* p = news;
224         double* up = a.s + a.len;
225         double* t = a.s;
226         while (t < up) *p++ = *t++ + b;
```

```
227         return vector(a.len, news);
228 }
229
230 vector&                 operator -  (const vector& a, double  b)
231 {
232         double* news = (double*)malloc(a.len * sizeof(double)); Verify(
                news);
233         double* p = news;
234         double* up = a.s + a.len;
235         double* t = a.s;
236         while (t < up) *p++ = *t++ - b;
237         return vector(a.len, news);
238 }
239
240 vector&                 operator *  (const vector& a, double  b)
241 {
242         double* news = (double*)malloc(a.len * sizeof(double)); Verify(
                news);
243         double* p = news;
244         double* up = a.s + a.len;
245         double* t = a.s;
246         while (t < up) *p++ = *t++ * b;
247         return vector(a.len, news);
248 }
249
250 vector&                 operator /  (const vector& a, double  b)
251 {
252         double* news = (double*)malloc(a.len * sizeof(double)); Verify(
                news);
253         double* p = news;
254         double* up = a.s + a.len;
255         double* t = a.s;
256         while (t < up) *p++ = *t++ / b;
257         return vector(a.len, news);
258 }
259
260 // vector by vector -> vector operations
261
262 vector&                 operator +  (const vector& a, const vector& b)
263 {
264         if (a.len != b.len)
265                 Error("Different lenght vectors " << a.len << " != " <<
                        b.len);
266         //double* news = new double [a.len]; Verify(news);
```

```
267         double* news = (double*)malloc(a.len * sizeof(double)); Verify(
                news);
268         double* p = news;
269         double* up = a.s + a.len;
270         double* t = a.s;
271         double* u = b.s;
272         while (t < up) *p++ = *t++ + *u++;
273         return vector(a.len, news);
274 }
275 vector&                 operator -  (const vector& a, const vector& b)
276 {
277         if (a.len != b.len)
278                 Error("Different lenght vectors " << a.len << " != " <<
                        b.len);
279         //double* news = new double [a.len]; Verify(news);
280         double* news = (double*)malloc(a.len * sizeof(double)); Verify(
                news);
281         double* p = news;
282         double* up = a.s + a.len;
283         double* t = a.s;
284         double* u = b.s;
285         while (t < up) *p++ = *t++ - *u++;
286         return vector(a.len, news);
287 }
288 vector& vector::vectorial(const vector& a, const vector& b)
289 {
290 #ifdef DEBUG
291         if (a.len != b.len)
292                 Error("Different lenght vectors in vectorial " << a.len
                        << " != " << b.len);
293         if (len && (len != a.len))
294                 Error("Wrong lenght vectors in vectorial " << a.len << "
                         != " << len);
295 #endif
296         if (!len)
297         {
298                 //s = new double[a.len]; Verify(s);
299                 s = (double*)malloc(a.len * sizeof(double)); Verify(s);
300         }
301         if (a.len == 2)
302         {
303                 s[0] = 0.0;
304                 s[1] = 0.0;
305                 s[2] = a.s[0] * b.s[1] - a.s[1] * b.s[0];
```

```
306         }
307         else if (a.len == 3)
308         {
309                 s[0] = a.s[1] * b.s[2] - a.s[2] * b.s[1];
310                 s[1] = a.s[2] * b.s[0] - a.s[0] * b.s[2];
311                 s[2] = a.s[0] * b.s[1] - a.s[1] * b.s[0];
312         }
313         else Error("ReaVec::vectorial only works for two or three
               dimensions");
314         return vector(len, s);
315         }
316 double dot(const vector& a, const vector& b)
317 {
318 #ifdef DEBUG
319         if (a.len != b.len)
320                 Error("Different lenght vectors " << a.len << " != " <<
                    b.len);
321 #endif
322         register double* t = a.s;
323         register double* u = b.s;
324         register double res = 0;
325         double* up = t + a.len;
326         while (t < up)
327                 res += *t++ * *u++;
328         return res;
329 }
330 vector& vector::product(const Matrix& a, const vector& b) // JULIO
      adding
331 {
332 #ifdef DEBUG
333   if (a.size  == 0) Error("Can't multiply zero sized matrix");
334   if (b.len == 0) Error("Can't multiply zero sized vector");
335   if (unsigned(a.nc) != b.len)
336     Error("Incompatible dimensions for multiply\n" <<
337      "\tMatrix : nrows=" << a.nr << ", ncols=" << a.nc << "\n" <<
338      "\tVector : length="<< b.len);
339   if (len != b.len) Error("Incompatible vector dimensions for multiply\n
        ");
340 #endif
341
342   double* aS = a.s;
343   double* ns = s;
344   int anc = a.nc;
345   int bnc = b.len;
```

```
346   int nr = a.nr;
347
348   while(nr--)
349     {
350           double *  up1 = aS + (anc & ~07);
351           double *  up2 = aS + anc;
352
353       register  double* bs = b.s;
354       register  double* as = aS;
355       register double res = 0;
356       while(as<up1)
357         {
358           res += *as++ * *bs++; res += *as++ * *bs++;
359           res += *as++ * *bs++; res += *as++ * *bs++;
360           res += *as++ * *bs++; res += *as++ * *bs++;
361           res += *as++ * *bs++; res += *as++ * *bs++;
362         }
363       if(as < aS + (anc & ~03))
364         {
365           res += *as++ * *bs++; res += *as++ * *bs++;
366           res += *as++ * *bs++; res += *as++ * *bs++;
367         }
368       while(as<up2) { res += *as++ * *bs++; }
369       *ns++ = res;
370       aS += a.nc;
371     }
372
373   return *this;
374 }
375
376
377 vector& vector::add_product(const Matrix& a, const vector& b) // JULIO
      adding
378 {
379 #ifdef DEBUG
380   if (a.size  == 0) Error("Can't multiply zero sized matrix");
381   if (b.len == 0) Error("Can't multiply zero sized vector");
382   if (unsigned(a.nc) != b.len)
383     Error("Incompatible dimensions for multiply\n" <<
384      "\tMatrix : nrows=" << a.nr << ", ncols=" << a.nc << "\n" <<
385      "\tVector : length="<< b.len);
386   if (len != b.len) Error("Incompatible vector dimensions for multiply\n
       ");
387 #endif
```

```
388
389   double* aS = a.s;
390   double* ns = s;
391   int anc = a.nc;
392   int bnc = b.len;
393   int nr = a.nr;
394
395   while(nr--)
396     {
397           double *  up1 = aS + (anc & ~07);
398           double *  up2 = aS + anc;
399
400       register  double* bs = b.s;
401       register  double* as = aS;
402       register double res = 0;
403       while(as<up1)
404         {
405           res += *as++ * *bs++; res += *as++ * *bs++;
406           res += *as++ * *bs++; res += *as++ * *bs++;
407           res += *as++ * *bs++; res += *as++ * *bs++;
408           res += *as++ * *bs++; res += *as++ * *bs++;
409         }
410       if(as < aS + (anc & ~03))
411         {
412           res += *as++ * *bs++; res += *as++ * *bs++;
413           res += *as++ * *bs++; res += *as++ * *bs++;
414         }
415       while(as<up2) { res += *as++ * *bs++; }
416       *ns++ += res;
417       aS += a.nc;
418     }
419
420   return *this;
421 }
422
423
424 vector& vector::sub_product(const Matrix& a, const vector& b) // JULIO
      adding
425 {
426 #ifdef DEBUG
427   if (a.size  == 0) Error("Can't multiply zero sized matrix");
428   if (b.len == 0) Error("Can't multiply zero sized vector");
429   if (unsigned(a.nc) != b.len)
430     Error("Incompatible dimensions for multiply\n" <<
```

```
431       "\tMatrix : nrows=" << a.nr << ", ncols=" << a.nc << "\n" <<
432       "\tVector : length="<< b.len);
433    if (len != b.len) Error("Incompatible vector dimensions for multiply\n
          ");
434 #endif
435
436    double* aS = a.s;
437    double* ns = s;
438    int anc = a.nc;
439    int bnc = b.len;
440    int nr = a.nr;
441
442    while(nr--)
443      {
444            double *  up1 = aS + (anc & ~07);
445            double *  up2 = aS + anc;
446
447        register  double* bs = b.s;
448        register  double* as = aS;
449        register double res = 0;
450        while(as<up1)
451          {
452            res += *as++ * *bs++; res += *as++ * *bs++;
453            res += *as++ * *bs++; res += *as++ * *bs++;
454            res += *as++ * *bs++; res += *as++ * *bs++;
455            res += *as++ * *bs++; res += *as++ * *bs++;
456          }
457        if(as < aS + (anc & ~03))
458          {
459            res += *as++ * *bs++; res += *as++ * *bs++;
460            res += *as++ * *bs++; res += *as++ * *bs++;
461          }
462        while(as<up2) { res += *as++ * *bs++; }
463        *ns++ -= res;
464        aS += a.nc;
465      }
466
467    return *this;
468 }
```