FEP Assignment 4

Justin Clough, RIN:661682899

July 11, 2017

1 Introduction

introduction text here.

2 Technical Description

design text here.

2.1 Finite Element Method

FEM text here.

2.2 Numerical Integration

Numerical integration text here.

2.3 Code Description

Psuedo code intro text here.

2.3.1 Class Description

Class description text here.

2.3.2 Psuedo Code

Psuedo code text here.

3 Testing

Testing text here.

3.1 Linear Triangular Elements

Linear Tri element text here.

3.2 Linear Quadrilateral Elements

Linear quad text here.

3.3 Quadratic Triangular Elements

Quadratic Tri element text here.

3.4 Quadratice Quadtrilateral Elements

Quadratice quad element text here.

4 Conclusion

conclusion text here.

A Code

A.1 a4.cc

```
1  #include "eigen_headers/Sparse"
2
3  // PUMI Headers
4  #include <PCU.h>
5  #include <pumi.h>
6
7  // APF Headers
8  #include <apfNumbering.h>
9  #include <apfShape.h>
10
11 // GMI Headers
12 #include "gmi_mesh.h"
13 #include "gmi_sim.h"
14
15 //STL Headers
16 #include <iostream>
17 #include <fstream>
18 #include <set>
19 #include <vector>
20 #include <deque>
21 #include <iterator>
22 #include <string>
23 #include <sstream>
24 #include <cstdlib>
25 #include <math.h>
26
27
28 using std::cout;
29 using std::endl;
30 using std::string;
31
32 class boundaryCond_t
33 {
34   public:
35     char type; // N for Neumann, D for Dirichlet
36     int geom_dim;
37     int geom_ID;
```

```
38      int direction;
39      double value;
40      bool DOG_zero;
41      bool DOG;
42      void print();
43      boundaryCond_t();
44  };
45  class paramList
46  {
47    public:
48      int dimension;
49      int order;
50      int numSides;
51      pGeom geom;
52      pMesh mesh;
53      std::vector<boundaryCond_t> BCs;
54      void assign_BC(boundaryCond_t BonCon)
55      { BCs.push_back(BonCon); }
56      void print();
57  };
58
59  void start(int argc, char** argv);
60  void finish(paramList& list);
61  void read_control(const char* ctrl, paramList& list);
62  void parse_control(std::ifstream& file, paramList& list);
63  void line_parse(string& line, size_t& pos, paramList& ↩
       list);
64  void set_BC(string& cmd, string& action,paramList& list);
65  void print_error( string message);
66
67  int main( int argc, char** argv)
68  {
69    if(argc != 2)
70    {
71      printf("Usage: %s <Control>.ctrl\n", argv[0]);
72      return 0;
73    }
74
75    start(argc, argv);
76
```

```
77    paramList list;
78
79    Eigen::SparseMatrix<double> K;
80
81    K.conservativeResize(4,4);
82    K.setZero();
83    std::cout << K << std::endl;
84
85    read_control(argv[1], list);
86
87    finish(list);
88    cout << "Success!" << endl;
89    return 0;
90  }
91
92  void start(int argc, char** argv)
93  {
94    MPI_Init(&argc,&argv);
95    pumi_start();
96    return;
97  }
98
99  void finish(paramList& list)
100 {
101   pumi_mesh_delete(list.mesh);
102   pumi_finalize();
103   MPI_Finalize();
104   return;
105 }
106
107 void parse_control(std::ifstream& file, paramList& list)
108 {
109   list.dimension = 2; // Hard code to only solve 2D ↩
        problems
110   string line;
111   string delim = " ";
112   while(std::getline(file, line))
113   {
114     string cmd;
115     size_t pos = line.find(delim);
```

```
116       size_t null_pos = std::string::npos;
117       if(pos!=null_pos)
118       {
119         line_parse(line, pos, list);
120       }
121     }
122     return;
123 }
124
125 void read_control(const char* ctrl, paramList& list)
126 {
127     std::ifstream ctrlFile (ctrl);
128     if (ctrlFile.is_open())
129     {
130       parse_control(ctrlFile, list);
131       ctrlFile.close();
132     }
133     else
134     { print_error( "ERROR_OPENING_CONTROL_FILE"); }
135
136     return;
137 }
138
139 void line_parse(string& line, size_t& pos, paramList& ↩
      list)
140 {
141     string cmd = line.substr(0, pos);
142     string action = line.substr(pos+1, line.length());
143     if(cmd.compare("ELEMENT_ORDER")==0)
144     {
145       if(action.compare("linear")==0)
146       {
147         list.order = 1;
148       }
149       else if (action.compare("quadratic"))
150       {
151         list.order = 2;
152       }
153       else { print_error("ERROR_READING_ELEMENT_ORDER"); }
154     }
```

```
155     else if (cmd.compare("ELEMENT_SHAPE") ==0)
156     {
157       if(action.compare("three_sided")==0)
158       {
159         list.numSides = 3;
160       }
161       else if(action.compare("four_sided") ==0)
162       {
163         list.numSides = 4;
164       }
165       else { print_error("ERROR_READING_ELEMENT_SHAPE"); }
166     }
167
168     //TODO: Read in dmg instead of this mess
169     else if (cmd.compare("GEOM_FILE")==0)
170     {
171
172       list.geom = pumi_geom_load(action.c_str(), "mesh");
173       cout << "MODEL_INFORMATION:_\n"
174            << "Number_of_Vertices:_" << pumi_geom_getNumEnt↩
                 ( list.geom, 0) << endl
175            << "Number_of_Edges:_" << pumi_geom_getNumEnt( ↩
                 list.geom, 1)<< endl
176            << "Number_of_Faces:_" << pumi_geom_getNumEnt( ↩
                 list.geom, 2)<< endl;
177     }
178     else if (cmd.compare("NEUMANN")==0 || cmd.compare("↩
          DIRICHLET")==0)
179     {
180       set_BC(cmd, action, list);
181     }
182     else if (cmd.compare("#")==0)
183     {
184       // is a commented line, do nothing
185     }
186     else { print_error("ERROR_READING_CONTROL_FILE");}
187
188     return;
189 }
190
```

```
191 void set_BC(string& cmd, string& action,paramList& list)
192 {
193   int geom_dim;
194   int geom_ID;
195   bool zero_flag = false;
196   double vector[3] = {0.0,0.0,0.0};
197   boundaryCond_t BC;
198 
199   string delim = " ";
200   size_t pos = action.find(delim);
201   size_t null_pos = std::string::npos;
202   int inst = 0;
203   while(pos!=null_pos)
204   {
205     string value = action.substr(0, pos);
206     inst++;
207     if(inst==1)
208     {
209       geom_dim=std::atoi(value.c_str());
210     }
211     else if (inst==2)
212     {
213       geom_ID=std::atoi(value.c_str());
214     }
215     else if (inst==3)
216     {
217       if(value.compare("X")==0)
218       {
219         BC.direction = 0;
220       }
221       else if(value.compare("Y")==0)
222       {
223         BC.direction = 1;
224       }
225       else if(value.compare("Z")==0)
226       {
227         BC.direction = 2;
228       }
229       else { print_error("ERROR READING BOUNDARY ↩
          CONDITION DIRECTION");}
```

```
230       }
231       else if (inst==4)
232       {
233         BC.value = std::atof(value.c_str());
234         if(BC.value == 0)
235         {
236           zero_flag = true;
237         }
238       }
239       else { print_error("ERROR READING BOUNDARY CONDITION ↩
              VALUES");}
240       action.erase(0, pos+delim.size());
241       pos = action.find(delim);
242     }
243
244     if(cmd.compare("NEUMANN")==0)
245     {
246       BC.type = 'N';
247     }
248     else if (cmd.compare("DIRICHLET")==0)
249     {
250       BC.type = 'D';
251       BC.DOG = true;
252     }
253     else { print_error("ERROR READING BOUNDARY CONDITION ↩
            TYPE");}
254
255     BC.geom_dim = geom_dim;
256     BC.geom_ID = geom_ID;
257     BC.DOG_zero = zero_flag;
258     list.assign_BC(BC);
259     return;
260 }
261
262 void print_error( string message)
263 {
264     cout << message << endl;
265     std::abort();
266 }
267
```

```
268 void boundaryCond_t::print()
269 {
270   cout << "Type = " << type << endl;
271   cout << "geom_dim = " << geom_dim << endl;
272   cout << "geom_ID = " << geom_ID << endl;
273   cout << "Direction = " << direction << endl;
274   cout << "Value = " << value << endl;
275   cout << "DOG_zero = " << DOG_zero << endl;
276 }
277
278 boundaryCond_t::boundaryCond_t ()
279 {
280   DOG = false;
281   DOG_zero = false;
282 }
283
284 void paramList::print()
285 {
286   cout << "Dimension = " << dimension << endl;
287   cout << "Element Order = " << order << endl;
288   cout << "Element Sides = " << numSides << endl;
289   cout << "Mesh Refinement Level = " << refinement << ↩
        endl;
290   cout << "Boundary Conditions set: " << endl;
291   for(int i=0; i<(int)BCs.size(); i++)
292   {
293     boundaryCond_t BC = BCs[i];
294     BC.print();
295     cout << endl;
296   }
297   return;
298 }
```