Angewandte Umweltsystemanalyse: Finite-Elemente-Methode (FEM)

Prof. Dr.-Ing. habil. Olaf Kolditz

[1]Helmholtz Centre for Environmental Research – UFZ, Leipzig

[2]Technische Universität Dresden – TUD, Dresden

Dresden, 22. Juni 2012

Vorlesungsplan SoSe 2012: USA-I Numerische Methoden

#	Datum	Vorlesung	Übung	Skript
1	13.04.12	Einführung, Systemanalyse		
2	13.04.12	Grundwasserhydraulik, Prinzip-Beispiel	USA1	1.1+2
3	20.04.12	Einführung in geotechnische Modellierung		PDF
4	27.04.12	FDM#1: 2D, explizit	USA2	1.3
5	27.04.12	FDM#2: Selke-Modell, Q&D	USA3	1.4
6	04.05.12	FDM#3: Selke-Modell, OOP,	USA4	1.5
7	11.05.12	FDM#3: Selke-Modell, VTK	USA5	1.5
8	18.05.12	FH-DGG Tagung		
9	25.05.12	FDM#4: implizit, instationär	USA6	1.6
10	01.06.12	vorlesungsfrei		
11	08.06.12	FDM#5: stationär, Randbedingungen	USA7+8	1.7+8
12	08.06.12	FDM#6: Rechteck-Aquifer	USA9+10	1.9+10
13	15.06.12	FEM#1: Grundlagen		2.1
14	15.06.12	FEM#2: 1D Modell	USA11	2.2
15	22.06.12	FEM#3: Implementierung, 2D Modell	USA12	2.3
16	29.06.12	UFZ-Exkursion: MOSAIC / TERENO		
17	06.07.12	Qt, Klausurvorbereitung		
18	13.07.12	UFZ-Exkursion: VISLab		

Lecture Table of Contents

- Wir fangen an mit dem Ergebnis ...

Finite-Elemente-Methode (FEM)

Wir haben uns sehr intensiv mit der Methode der finiten Differenzen beschäftigt. Bei der Einführung der numerischen Berechnungsmethoden in der Hydroinformatik II Veranstaltung haben wir gesehen, dass es ein ganzes Arsenal von Verfahren gibt (Abb. 2.1, Hydroinformatik II Skript), welche für bestimmte Problemstellungen geeignet oder ungeeignet sind. In den Visualisierungsübungen im VISLab werden wir sehen, dass FD Verfahren Grenzen haben, wenn es um die exakte Beschreibung komplexer Geometrien geht. Hier sind Verfahren im Vorteil, die sogenannte unstrukturierte Rechengitter benutzen können. Hierzu zählt z.B. die Finite Elemente Methode, mit der wir uns nun etwas näher beschäftigen möchten. Die Abb. 1 zeigt uns ein aktuelles Beispiel aus einem Forschungsvorhaben zusammen mit der Bundesanstalt für Geowissenschaften und Rohstoffe (BGR) in Hannover

Finite-Elemente-Methode (FEM)

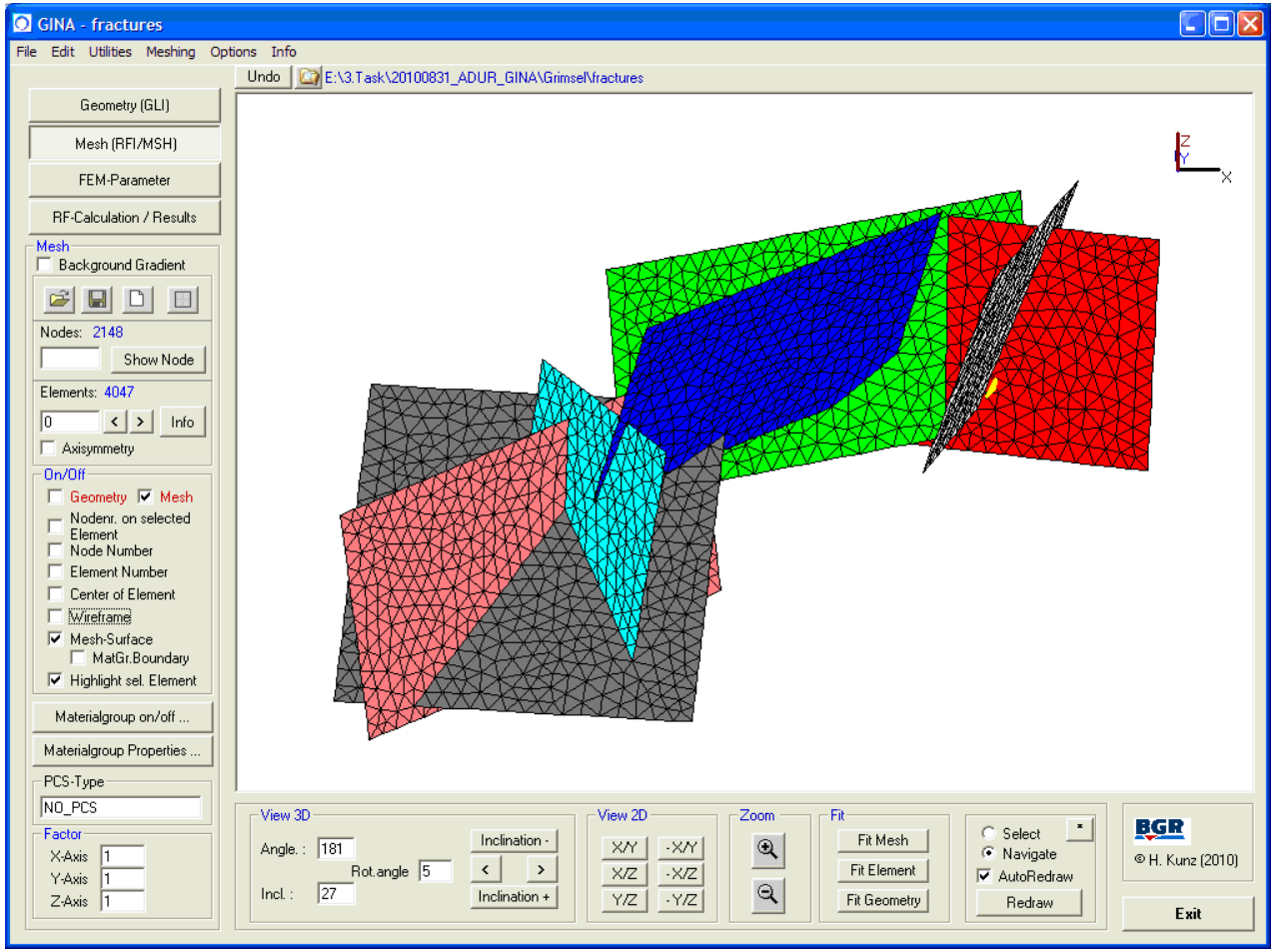

Abbildung: Modellierung eines Kluftsystems im Kristallin (Herbert Kunz, BGR)

Finite-Elemente-Methode (FEM) - Implementierung

Vergleichen wir die Quelltexte der `main` Funktionen für FD und FE Verfahren, sehen wir kaum Unterschiede. Das heisst die Abläufe (Algorithmen) sind sehr ähnlich.

Finite-Elemente-Methode (FEM)

```
extern void Gauss(double*,double*,double*,int);

int main()
{
  //-----------------------------------------
  FEM* fem = new FEM();
  fem->SetInitialConditions();
  fem->SetBoundaryConditions();
  //-----------------------------------------
  int tn = 10;
  for(int t=0;t<tn;t++)
  {
    fem->AssembleEquationSystem();
    Gauss(fem->matrix,fem->vecb,fem->vecx,fem->IJ);
    fem->SaveTimeStep();
    fem->OutputResults(t);
  }
  //-----------------------------------------
  return 0;
}
```

Finite-Elemente-Methode (FEM)

Im Unterschied zur Implementierung des FD Verfahrens wollen wir Flexibilität erreichen bezüglich der Definition des Modellgebietes. Das ist auch der große Vorteil der FEM - die flexible Geometrie. Bei der Programmierung des FD Verfahrens haben wir die Netzgenerierung ja während der Programmausfürhung erledigt. Das ist ja auch nicht weiter schwer, da bei der FDM immmer regelmäßige Rechteckgitter die Ausgangsbasis sind.

Finite-Elemente-Methode (FEM)

Der beste Weg zur Flexibilität ist die Möglichkeit zu schaffen, FE Gitter über eine Datei einlesen zu können. Also brauchen wir eine Lesefunktion und können auch unsere Erfahrungen mit File-Streams (Hydroinformatik I, Kapitel 6) zurückgreifen. Beim Schreib der Lesefunktion lernen wir viel über die Art der Datenstrukturen, die wir benötigen. Aus dem Theorieteil wissen wir schon, dass wir es mit Matrizen zu tun haben werden. Wir benutzen das gleiche Konzept wie für die Lesefunktion der Studentenklasse `CStudent` (Übung 6.3, Hydroinformatik I).
Die Lesefunktion `ReadMesh()` bekommt den File-Stream als Argument und liefert die aktuelle Position im Stream zurück. In der Funktion gibt es eine Schleife, die solange lüft, bis das Dateiende erreicht wird. Dabei wird das File zeilenweise ausgelesen (`msh_file.getline()`) und die Zeile ausgewertet. Leerzeilen werden übersprungen. Wenn das Keyword `#STOP` auftritt, wird das Lesen beendet.

Finite-Elemente-Methode (FEM)

```
std::ios::pos_type FEM::ReadMesh(std::ifstream& msh_file)
{
  std::ios::pos_type position;
  std::string input_line;
  char buffer[256]; // MAX_LINE
  while(!msh_file.eof())
  {
    position = msh_file.tellg();
    msh_file.getline(buffer,256);
    input_line = buffer;
    if(input_line.size()<1) // skip empty lines
      continue;
    if(input_line.find("#STOP")!=std::string::npos)
      return;
    ...
  }
}
```

Finite-Elemente-Methode (FEM)

Das Eingabefile für das 1D FE Gitter sieht wie folgt aus.

```
#FDMESH
$NODES
5
0.
2.
4.
7.
10.
$ELEMENTS
4
0 1
1 2
2 3
3 4
#STOP
```

Finite-Elemente-Methode (FEM)

Der spezielle Leseteil sucht nach den Schlüsselwörtern für Knoten und Elemente, die wir natürlich in entsprechenden Vektoren speichern.

```
$NODES    - std::vector<double>node_vector;
$ELEMENTS - std::vector<int*>element_vector;
```

Bei den Knoten merken wir uns zunächst nur den x-Koordinatenwert. Bei den Elementen geht es um die beiden Knoten, die dazugehören. Daher braucht der Elementvektor als Argument eine Datenstruktur des Typs `int*`. Für diese Datenstruktur `int* element_nodes` müssen wir vor Benutzung Speicher für zwei Integer-Zahlen für die Knotennummern des Elements reservieren.

Finite-Elemente-Methode (FEM)

```
void FEM::ReadMesh(std::ifstream& msh_file)
{
  ...
  while(!msh_file.eof())
  {
  ...
    // Dealing with subkeywords
    if(input_line.find("$NODES")!=std::string::npos)
    {
      msh_file >> nn;
      for(int i=0;i<nn;i++)
      {
        msh_file >> x;
        node_vector.push_back(x);
      }
    }
    if(input_line.find("$ELEMENTS")!=std::string::npos)
    {
      msh_file >> ne;
      for(int i=0;i<ne;i++)
      {
        element_nodes = new int[2];
        msh_file >> element_nodes[0];
        msh_file >> element_nodes[1];
        element_vector.push_back(element_nodes);
      }
    }
  }
}
```

Finite-Elemente-Methode (FEM)

Um zu prüfen, ob das Lesen des Eingabefiles geklappt hat, schreiben wir gleich noch eine dazu passende Output-Funktion.

```
void FEM::OutputMesh(std::ofstream& msh_file_test)
{
  msh_file_test << "#FEM_MSH" << std::endl;
  msh_file_test << "$NODES" << std::endl;
  for(int n=0;n<(int)node_vector.size();n++)
  {
    msh_file_test << node_vector[n] << std::endl;
  }
  msh_file_test << "$ELEMENTS" << std::endl;
  for(int e=0;e<(int)element_vector.size();e++)
  {
    msh_file_test << element_vector[e][0] << " " \
                  << element_vector[e][1] << std::endl;
  }
  msh_file_test << "#STOP" << std::endl;
}
```

Anfangsbedingungen

```
void FEM::SetInitialConditions()
{
  for(int n=0;n<(int)node_vector.size();n++)
  {
    node_vector[n] = h_initial;
  }
}
```

Randbedingungen

In der Aufgabenbeschreibung (Abb. **??**) sehen wir die gesetzten Randbedingungen an des Säulenmodells, $h = 12$m oben (Knoten 0) und $h = 0$m unten (Knoten 4).

```
void FEM::SetBoundaryConditions()
{
  bc_nodes.push_back(0);
  u[bc_nodes[0]] = h_top;  u_new[bc_nodes[0]] = h_top;
  bc_nodes.push_back(4);
  u[bc_nodes[1]] = h_bottom;  u_new[bc_nodes[1]] = h_bottom;
}
```

FEM: Elementmatrizen

Die Berechnung der Elementmatrizen teilen wir auf in zwei Funktionen.
`CalculateElementMatrices()` ist eine Schleife über alle Elemente und ruft die eigentliche Berechnungsfunktion jede Elementmatrix auf.

```
void FEM::CalculateElementMatrices()
{
  for(int e=0;e<(int)element_vector.size();e++)
  {
    CalculateElementMatrix(e);
  }
}
```

FEM:...

`CalculateElementMatrix(int)` berechnet die Element-Leitfähigkeitsmatrix gemäß Gleichung (**??**). Für lineare 1D Elemente (mit 2 Knoten) ist die Elementmatrix eine 2×2 Matrix. Für die Berechnung benötigen wir die Elementlänge L und die jeweilige hydraulische Leitfähigkeit des Elements $K[e]$. Gespeichert werden die Elementmatrizen in einem Vektor `element_matrix_vector`.

FEM:...

```
void FEM::CalculateElementMatrix(int e)
{
  element_matrix = new double[4];
  element_nodes = element_vector[e];
  x0 = node_vector[element_nodes[0]];
  x1 = node_vector[element_nodes[1]];
  L = x1-x0;
  element_matrix[0] = K[e]/L;
  element_matrix[1] = -K[e]/L;
  element_matrix[2] = -K[e]/L;
  element_matrix[3] = K[e]/L;
  element_matrix_vector.push_back(element_matrix);
}
```

FEM:...

Zum Test legen wir die Output-Funktion für Elementmatrizen an.

```
void FEM::DumpElementMatrices(std::ofstream& file)
{
  int ii;
  for(int e=0;e<(int)element_matrix_vector.size();e++)
  {
    file << "--------------------------" << std::endl;
    element_matrix = element_matrix_vector[e];
    for(int j=0;j<2;j++)
    {
      for(int i=0;i<2;i++)
      {
        ii= 2*j+i;
        file << element_matrix[ii] << " ";
      }
      file << std::endl;
    }
  }
}
```

FEM: Elementmatrizen - Ergebnis

```
--------------------------
5e-006 -5e-006
-5e-006 5e-006
--------------------------
1e-005 -1e-005
-1e-005 1e-005
--------------------------
3.33333e-006 -3.33333e-006
-3.33333e-006 3.33333e-006
--------------------------
3.33333e-006 -3.33333e-006
-3.33333e-006 3.33333e-006
```

FEM: Gleichungssystem

Nun müssen die Elementmatrizen zum Gleichungssystem zusammengefügt werden gemäß Gleichung (**??**). Index im Gleichungssystem

```
void FEM::AssembleEquationSystem()
{
  int i,j;
  for(int e=0;e<ne;e++)
  {
    element_nodes = element_vector[e];
    element_matrix = element_matrix_vector[e];
    i = element_nodes[0];
    j = element_nodes[1];
    matrix[i*nn+i] += element_matrix[0];
    matrix[i*nn+i+1] += element_matrix[1];
    matrix[j*nn+j-1] += element_matrix[2];
    matrix[j*nn+j] += element_matrix[3];
  }
}
```

FEM:...

```
 5e-006  -5e-006        0                0              0
-5e-006 1.5e-005      -1e-005           0              0
 0      -1e-005 1.33333e-005 -3.33333e-006            0
 0       0      -3.33333e-006  6.66667e-006 -3.33333e-006
 0       0                0   -3.33333e-006 3.33333e-006
```

FEM:...

Unser OOP zahlt sich aus. Für den Einbau der Randbedingungen können wir exakt die gleiche Funktion wie für das FD Verfahren nehmen. Warum eigentlich?

```
void FDM::IncorporateBoundaryConditions()
void FEM::IncorporateBoundaryConditions()
```

FEM:...

```
5e-006 0 0 0 0 b:6e-005
0 1.5e-005 -1e-005 0 0 b:6e-005
0 -1e-005 1.33333e-005 -3.33333e-006 0 b:0
0 0 -3.33333e-006 6.66667e-006 0 b:0
0 0 0 0 3.33333e-006 b:0
```

FEM:...

Auch für das Lösen des Gleichungssystems benutzten wir den bewährten Gauss-Löser - wie er ist.

```
12
9.33333
8
4
0
```

FEM:...

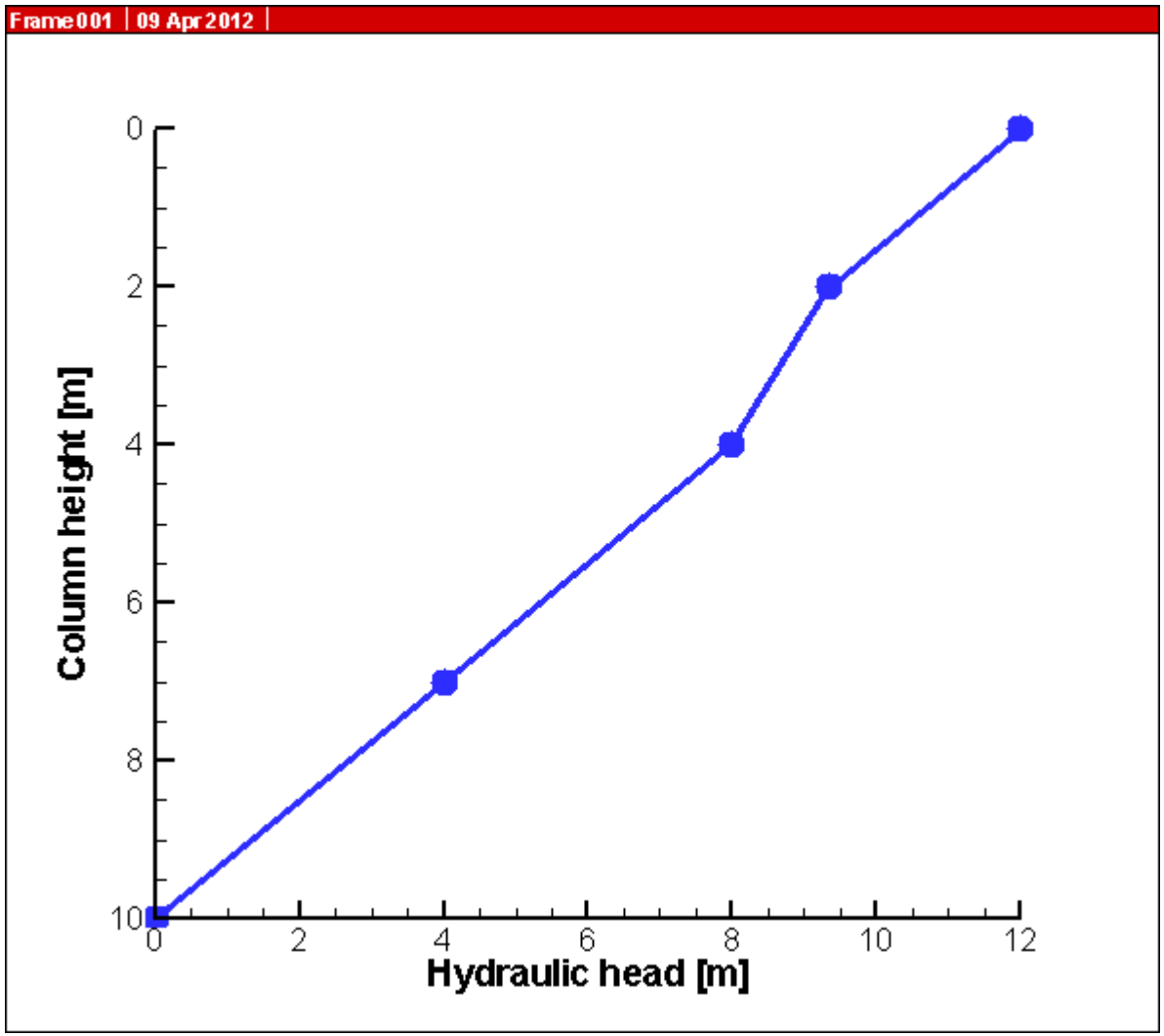

Abbildung: Hydraulisches Potenzial in der Bodensäule