Version 1.02 - 12. Mai 2011

Hydrosystemanalyse
"Grundlagen und Fallstudien"

Prof. Dr.-Ing. habil. Olaf Kolditz

TU Dresden / UFZ Leipzig
Angewandte Umweltsystemanalyse
Umweltinformatik

SS 2011

Autoren und Acknowledgements:

- Dr. Jens-Olaf Delfs: Oberflächenhydrologie
- Dr. Thomas Kalbacher: Bodenhydrologie
- Dr. Haibing Shao: Transportprozesse in porösen Medien
- Dr. Karsten Rink: OGS-GUI, Visualisierung der Fallstudie Bode-Einzugsgebiet
- Dr. Tom Fischer: OGS-GEO, Geometrische Modellierung
- last but not least Dr. Wenqing Wang (the right person if it becomes really difficult ...)

Gast-Dozenten:

- Dr. Zacharias, Dr. Wollschläger (UFZ, TERENO Projekt)
- Dr. Vienken, Dr. Dietrich (UFZ, Monitoring- und Erkundungstechniken)
- Dr. Fleckenstein (UFZ, Hyfrogeologie)
-

Vorlesungskonzept

Es geht weiter ... Nachdem wir uns im erstem Semester Hydroinformatik mit dem objekt-orientierten Programmierung mit C++ "angefreundet" haben, soll es in diesem Semester auch um die Anwendung in den Disziplinen Hydrologie, Wasser- und Abfallwirtschaft gehen. Natürlich werden wir uns mit sehr einfachen Beispielen beschäftigen (und z.B. keine kompletten Kläranlagen rechnen, dafür sind die Kollegen von der Siedlungswaaserwirtschaft zuständig). Dennoch wollen wir ihnen am Ende des Semesters auch einen Einblick in unsere laufende Forschung geben, z.B. den Einsatz von numerischen Modellen bei der Abfalllagerung [7], in der Hydrologie [3] und in der Grundwassermodellierung [9].

Der Vorlesungsfahrplan sieht wir folgt aus:

- Grundlagen - Mechanik (2-3V)
- Grundlagen - Numerik (2-3V)
- Prozessverständnis - Benchmarks (3V)
- Anwendungen (2V)
- Visual C++ mit Qt (4V)

Abbildung 1: Lehrbuch zur "Numerischen Hydromechanik" [6]

Grundlagen

Für die ersten beiden Teile: Mechanik und Numerik, verwende ich weitgehend Material aus meinem Springer-Lehrbuch "Numerical Methods in Environmental Fluid Mechanics" [6] (für Bauingenieure, siehe Abb. 1). Als deutsprachige Literatur empfehle ich ihnen natürlich die Dresdner Klassiker [5] und [1] sowie [2] (Hydromechanik, generell) [8] (Gerinnehydraulik), [4] (Grundwasserhydraulik). Die TU Dresden hat eine lange Tradition in der Wasserforschung.

Prozessverständnis

Anhand von einfachen Beispielen (Benchmarks) werden wir uns mit der Simulation von Diffusionprozessen, Grundwasser- und Gerinnehydraulik beschäftigen (siehe Anlage A). Diese Testbeispiele stammen aus der OpenGeoSys-(OGS)-Benchmarksammlung, die für die OGS-Entwicklung benutzen (komplett unter www.opengeosys.net)

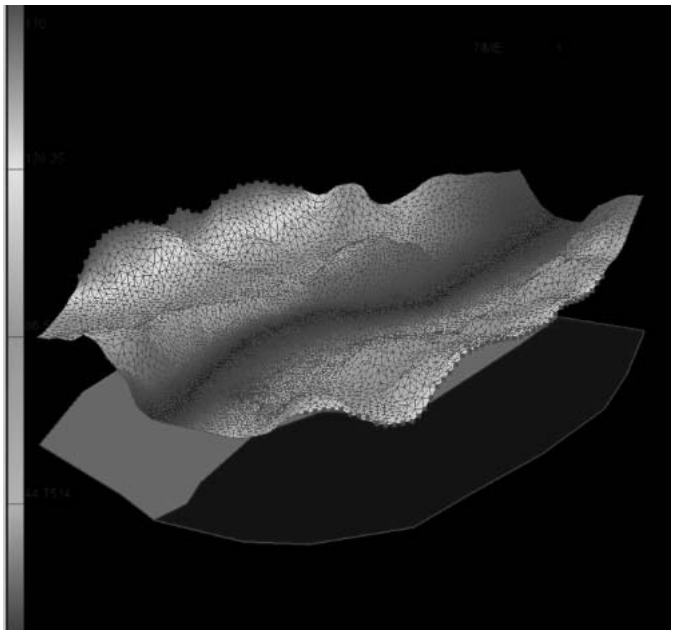

Abbildung 2: Übung zur Vorlesung "Prozesssimulation": Tübinger Grundwassermodell
(siehe http://www.uni-tuebingen.de//uni/epi/teaching/Case_studies_index.htm)

Anwendungen

Bei der Anwendung wollen wir ein (einfaches) Dresdner-Grundwassermodell entwickeln. Dabei haben wir mit echten Daten zu tun (z.B. digitalen Geländemodellen).

Programmierung

Natürlich geht's auch mit der C++ Programmierung (Qt) weiter, wir wollen soviel wie möglich selber implementieren.

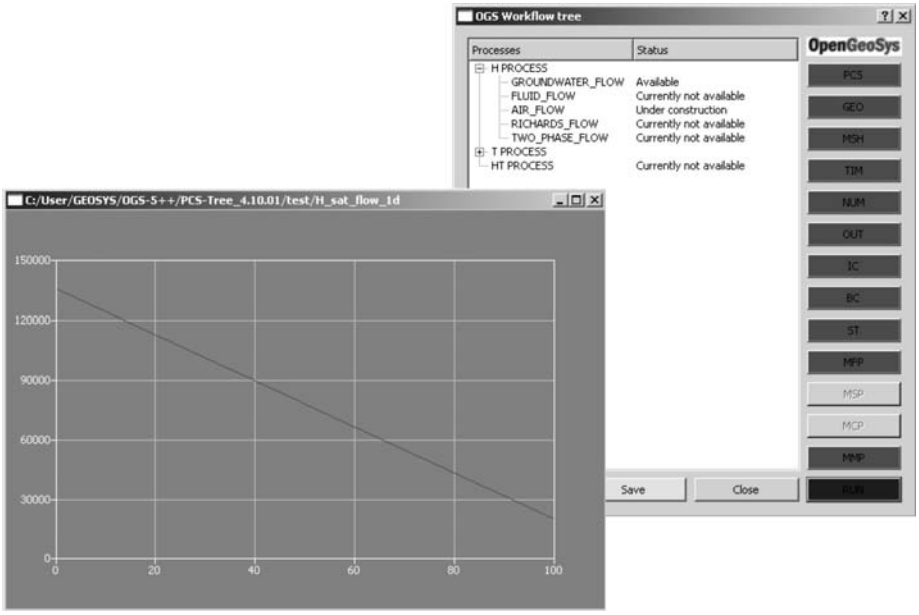

Abbildung 3: OGS (light) Studentenversion (daran werden wir gemeinsam arbeiten)

Last and least...

Um unsere Programme noch schicker zu machen, können wir sogennante SplashScreens `QSplashScreen` vor das Programm spannen. Also ein nettes Bildchen, das Aufmerksamkeit erwecken soll (Abb. 4).

.

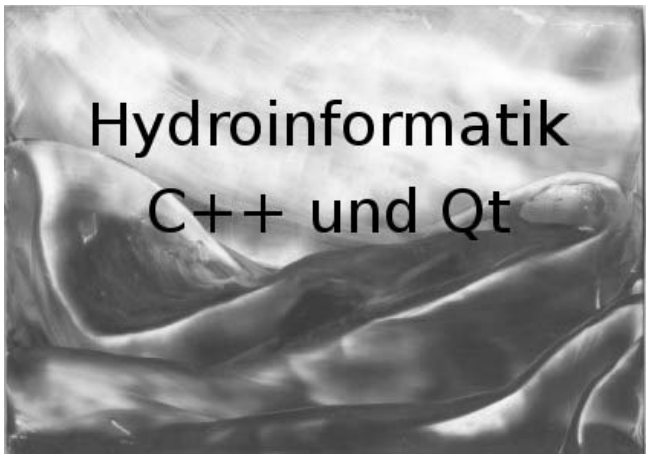

Abbildung 4: SplashScreen für unsere Vorlesung

Nun aber wirklich die letzte Übung, die mein Sohn Bastian zusammengebastelt hat.

Übung E11: SplashScreen

```
int main(int argc, char *argv[])
{
  QApplication a(argc, argv);
  //.....................................................
  QPixmap pixmap ("../bk-1.png");
  QSplashScreen splash(pixmap);
  splash.show();
  splash.showMessage(QObject::tr("Übung E10 wird geladen..."), Qt::black);
  QTime dieTime = QTime::currentTime().addSecs(3); while(QTime::currentTime()<dieTime)\
                  QCoreApplication::processEvents(QEventLoop::AllEvents, 100);
  // Warte-Funktion (http://lists.trolltech.com/qt-interest/2007-01/thread00133-0.html)
  //.....................................................
  Dialog w;
  w.setWindowTitle("Groundwater Simulator");
  w.setFixedWidth(350);
  w.show();
  return a.exec();
}
```

Organisatorisches

Die Prüfung 'Hydroinformatik I' (erstes Semester) ist eine Klausur. Die Prüfung 'Hydroinformatik II' (zweites Semester) besteht aus zwei Teilen, einer Klausur und einer Programmier-Hausarbeit. Bei den Klausuren geht es um die Beantwortung von Verständnisfragen (siehe Testfragen zu jeder Vorlesung).

Sprache: normalerweise in Deutsch. Da die Syntax von Computersprachen (wie C++) immer in Englisch ist, würde ich auch mal eine Vorlesung in Englisch anbieten.

Konsultationen: immer nach der Vorlesung (Freitag ab 15 Uhr).

Kontakt: jederzeit by e-mail (olaf.kolditz@ufz.de), in dringenden Fällen auch per Handy (0151 52739034).

Vorlesungsunterlagen: sind in bewährter Weise auf meinem UFZ Server zu finden (`http://www.ufz.de/index.php?de=11877`).

Vorlesung
Grundlagen

Kapitel 1

Grundwasserhydraulik

Wir haben uns bisher mit linearen und nicht-linearen Diffusionsgleichungen beschäftigt. Bisher war alles ein-dimensional. Mit numerischen Methoden gibt es keine Beschränkungen für die Geometrie. Im letzten Teil unserer Vorlesung lösen wir nun ein zwei-dimensionales Problem für eine horizontale Grundwasserströmung.

1.1 Grundwassergleichung

Das theoretische Grundgerüst liefert das Euler-Konzept (siehe Abschn. **??**). Wir starten mit der Massenbilanzgleichung (siehe Abschn. **??**). Für ein poröses Medium reduziert sich der für Fluide betretbare Raum um die Porosität n, die sich ändern kann.

$$\frac{\partial n\rho}{\partial t} + \nabla \cdot (n\rho \mathbf{v}) = Q_\rho \tag{1.1}$$

Für ein inkompressibles Fluid gilt dann (PF)

$$\rho \frac{\partial n}{\partial t} + \rho \nabla \cdot (n\mathbf{v}) = Q_\rho \tag{1.2}$$

oder noch besser

$$\frac{\partial n}{\partial t} + \nabla \cdot (n\mathbf{v}) = \frac{Q_\rho}{\rho_0} \tag{1.3}$$

In der Grundwasserhydraulik gilt

$$\frac{\partial n}{\partial t} = S\frac{\partial h}{\partial t} \tag{1.4}$$

$$n\mathbf{v} = \mathbf{q} = -\mathbf{K}\nabla h \quad \text{(Darcy Gesetz)} \tag{1.5}$$

Dabei sind: S der Speicherkoeffizient, h die Piezometer- oder hydraulische Höhe, $\mathbf{q}$ die Darcy- oder Filtergeschwindigkeit und $\mathbf{K}$ der hydraulische Leitfähigkeitstensor.

Die Grundwassergleichung lässt sich nun wie folgt schreiben.

$$S\frac{\partial h}{\partial t} + \nabla \cdot (n\mathbf{v}) = Q \quad (1.6)$$

$$S\frac{\partial h}{\partial t} - \nabla \cdot (\mathbf{K}\nabla h) = Q \quad (1.7)$$

$$S\frac{\partial h}{\partial t} - \frac{\partial}{\partial x}\left(K_x\frac{\partial h}{\partial x}\right) - \frac{\partial}{\partial y}\left(K_y\frac{\partial h}{\partial y}\right) - \frac{\partial}{\partial z}\left(K_z\frac{\partial h}{\partial z}\right) = Q \quad (1.8)$$

Wir begnügen uns mit einem 2-D horizontalen Modell.

$$S\frac{\partial h}{\partial t} - \frac{\partial}{\partial x}\left(K_x\frac{\partial h}{\partial x}\right) - \frac{\partial}{\partial y}\left(K_y\frac{\partial h}{\partial y}\right) = Q \quad (1.9)$$

1.2 Bilanzierung

Wir halten unser Anwendungsbeispiel natürlich so einfach wie möglich, dennoch enthält es viele wichtige "Features" von Grundwassermodellen (alles Weitere lernen sie natürlich bei Prof. Liedl ...). Das Beispiel stammt noch aus der Tübinger Zeit (Master Course in Applied Geosciences - AEG, daher auch alles in Englisch) und wurde durch Sebastian Bauer erarbeitet (jetzt Professor für GeoHydroModellierung in Kiel).

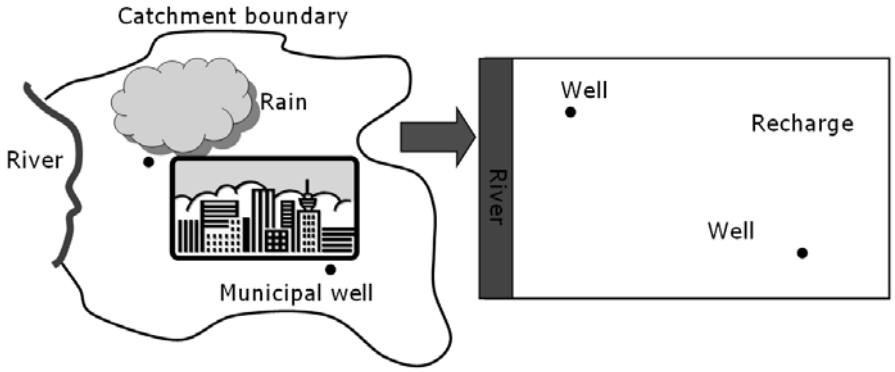

Abbildung 1.1: GW Testbeispiel

1.2.1 Definition des Testbeispiels

Die Abb. 1.7 zeigt uns ein Systemmodell für ein sogenanntes Grundwasser-Catchment mit einem Fluss (Vorfluter) und zwei Brunnen. Weitere wichtige geohydrologische Begriffe sind die Grundwasserneubildung (recharge) durch Niederschläge und Wasserscheiden. Das besondere Merkmal eines Catchments (Einzugsgebiet) ist seine Abgeschlossenheit durch Wasserscheiden, d.h. die Catchment-Randbedingung ist eine "no-flow" boundary condition (da geht nichts durch). In unserem Fall heisst das, die komplette Grundwasserneubildung geht ab in den Vorfluter.

Die Abb. 1.2 ist die mathematische Übersetzung des geohydrologischen Konzeptmodells in Abb. 1.7.

- Fluss: ist eine Randbedingung erster Art (siehe Abschn.), der Wasserstand h wird vorgegeben. Catchment: wie gesagt - no flow, also Fließgeschwindigkeit ist Null, was nach Darcy heisst, der Piezometer-Gradient ∇h ist Null.
- Grundwasserneubildung: ist ein flächenhafter Quellterm für das Grundwassersystem.
- Brunnen: sind lokale Senkenterme.

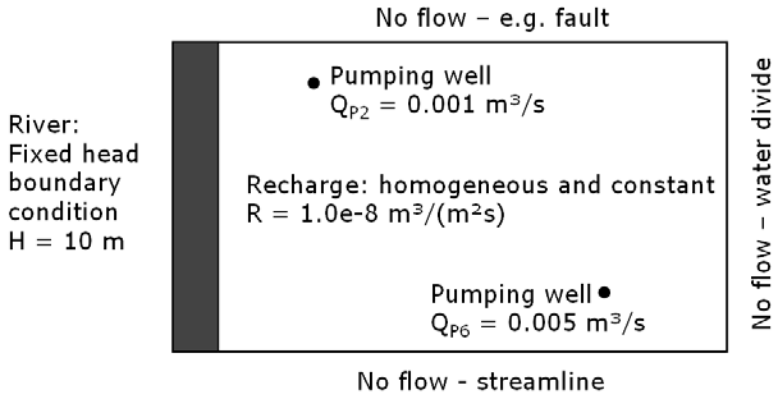

Abbildung 1.2: GW Testbeispiel - Randbedingungen

1.2.2 Rechenverfahren zur Bilanzierung

Nach der mathematischen Übersetzung (Abb. 1.2) nun die numerische Translation. Abb. 1.3 zeigt uns das finite Differenzen Gitter. Das Netz ist block-zentriert, d.h. wir nehmen das geometrische Zentrum der Zelle als Bezugspunkt.

.

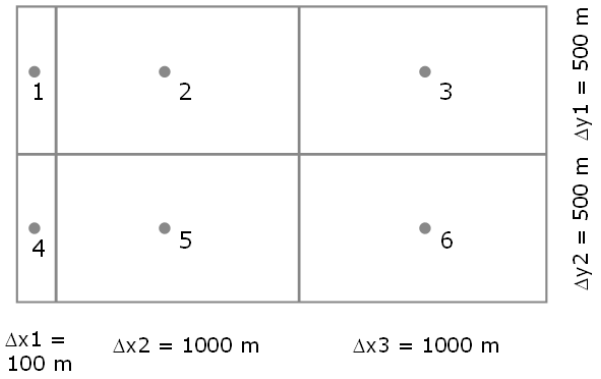

Abbildung 1.3: GW Testbeispiel - Finite Differenzen Schemata

Unser Grundwasserleiter (Aquifer) ist heterogen, d.h. es sind verschiedene geologische Schichten beteiligt (Abb. 1.4). T ist die sogenannte Transmissivität, der Quotient von hydraulischer Durchlässigkeit und Speicherkoeffizient.

$$T = \frac{K}{S} \tag{1.10}$$

Abbildung 1.4: GW Testbeispiel - Heterogener Aquifer

Schauen wir uns mal die Wasserbilanz für einen FD-Knoten genauer an. Q_{ij} ist der Zufluss zum Knoten j aus der Zelle i, also Q_{52} ist der Zufluss zum Knoten 2 aus der Zelle 5. Für den Knoten 2 gilt

$$Q_{12} + Q_{32} + Q_{52} + Q_R + Q_{P2} = 0 \tag{1.11}$$

Dabei sind Q_R der Zufluss durch die Grundwasserneubildung und Q_{P2} der Abfluss durch den Brunnen im Knoten 2.

.

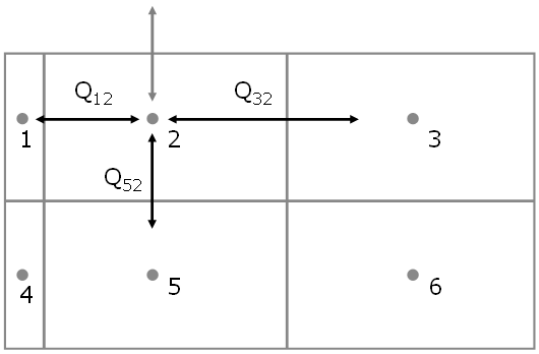

Abbildung 1.5: GW Testbeispiel - Wasserbilanz

Wir benutzen folgendes Differenzenschema.

$$\frac{\partial h}{\partial x} = \frac{h_i - h_j}{x_i - x_j} \tag{1.12}$$

$$\frac{\partial h}{\partial y} = \frac{h_i - h_j}{y_i - y_j} \tag{1.13}$$

Da unser FD-Gitter weder equidistant und unser Aquifer noch heterogen ist, schreiben wir besser.

$$\frac{\partial h}{\partial x}|_{12} = \frac{h_1 - h_2}{\Delta x_1/2 + \Delta x_5/2} \tag{1.14}$$

Damit können wir für die Flussterme schreiben.

$$Q_{12} = \Delta y_1 \frac{\Delta x_1 + \Delta x_2}{\Delta x_1/T_1 + \Delta x_2/T_2} \times \frac{h_1 - h_2}{\Delta x_1/2 + \Delta x_2/2} \tag{1.15}$$

$$Q_{52} = \Delta x_2 \frac{\Delta y_5 + \Delta y_2}{\Delta y_5/T_5 + \Delta y_2/T_2} \times \frac{h_5 - h_2}{\Delta y_5/2 + \Delta y_2/2} \tag{1.16}$$

Die Zahlen eingesetzt ergibt sich für

$$Q_{12} = 0.454545 - 0.0454545 h_2 \tag{1.17}$$

Die restlichen Terme für die Wasserbilanz in der Zelle 2 sind.

$$Q_{52} = 0.03333 h_5 - 0.03333 h_2 \tag{1.18}$$

$$Q_{32} = 0.02500 h_3 - 0.02500 h_2 \tag{1.19}$$

$$Q_R = R \Delta x_2 \Delta y_1 = 0.005 \tag{1.20}$$

$$Q_{P2} = -0.001 \tag{1.21}$$

Damit ist die Wasserbilanz für die Zelle 2.

$$2 : 0.458545 - 0.103788h_2 + 0.025h_3 + 0.03333h_5 = 0 \tag{1.22}$$

Für die anderen Zellen ergibt sich.

$$\begin{aligned} 3 &: 0.0050 + 0.0250h_2 - 0.0583h_3 + 0.0333h_6 = 0 \\ 5 &: 0.0959 + 0.0333h_2 - 0.0474h_3 + 0.0050h_6 = 0 \\ 6 &: 0.0000 + 0.0333h_3 + 0.0050h_5 - 0.0383h_6 = 0 \end{aligned} \tag{1.23}$$

Das sieht doch wieder ganz verdächtig nach

$$\mathbf{Ax} = \mathbf{b} \tag{1.24}$$

aus, wir müssen ein Gleichungssystem lösen, eine leichte Übung für uns ... (Gauss Löser).

Ergebnis:

$$\begin{aligned} h_1 &= 10.00 \\ h_2 &= 10.24 \\ h_3 &= 10.41 \\ h_4 &= 10.00 \\ h_5 &= 10.31 \\ h_6 &= 10.39 \end{aligned} \tag{1.25}$$

(1.26)

1.2.3 Programmtechnische Umsetzung

Letzter Akt: die Programmierung des Grundwassermodells (siehe Übung E10 auf der WebSeite).

Die Abb. 1.6 zeigt die Dialog-Gestaltung für unseren einfachen Grundwassersimulator.

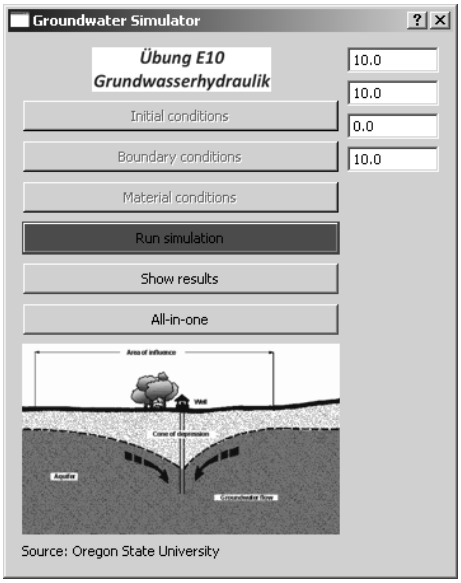

Abbildung 1.6: GW Testbeispiel - Dialog

Für den Test, dass das Gleichungssystem korrekt aufgestellt wird, schreiben wir uns eine kleine Funktion.

```
void Dialog::TestOutput(double*a,double*b)
{

  for(int i=0;i<6;i++)
  {
    for(int j=0;j<6;j++)
    {
      out_file << a[6*i+j] << "\t";
    }
    out_file << " : " << b[i] << endl;
  }
}
```

die folgendes Textfile produziert.

```
1       0       0       0       0       0      : 10
0       -0.104  0.025   0       0.0333  0      : -0.459
0       0.025   -0.0583 0       0       0.0333 : -0.005
0       0       0       1       0       0      : 10
0       0.0333  0       0       -0.0474 0.005  : -0.0959
0       0       0.0333  0       0.005   -0.383 : 0
```

Wir vergleichen mit der Theory (1.22) und (1.24) und sehen, dass alles geklappt hat.

```
void Dialog::TestOutput(double*x)
{
  for(int i=0;i<6;i++)
  {
    out_file << "h" << QString::number(i).toStdString() << ":" << x[i]
             << endl;
  }
}
```

Sie sehen, wir können die gleiche Schreibfunktion `TestOutput` für verschiedene Aufgaben nehmen. Dies nennt sich Polymorphismus in C++. Der Kompiler erkennt die verschiedenen Funktionen anhand der unterschiedlichen Parameterliste. Auch die Ergebnisse passen.

```
h1 = 10.00
h2 = 10.24
h3 = 10.41
h4 = 10.00
h5 = 10.31
h6 = 10.39
```

Für die Lösung des Gleichungssystems haben wir wieder unseren Gauss-Löser eingesetzt (siehe Abschn. **??**). Wenn noch Zeit ist, schauen wir uns alternativ noch anderes Lösungsverfahren an: Gauss-Seidel.

1.2.4 Gauss-Seidel Verfahren

- Umstellung des Gleichungssystems (1.24)

$$\begin{aligned} h_2 &= 0.2408h_3 + 0.3211h_5 + 4.4181 \\ h_3 &= 0.4285h_2 + 0.5714h_6 + 0.0857 \\ h_5 &= 0.7028h_2 + 0.1054h_6 + 2.0223 \\ h_6 &= 0.8695h_3 + 0.1304h_5 \end{aligned}$$

- Konstruktion eines iterativen Lösungsverfahrens
- Pro: Es muss kein Gleichungssystem gelöst werden.
- Con: Es kann auch mal nicht klappen (keine Konvergenz).

$$\begin{aligned} h_{2,i+1} &= 0.2408h_{3,i} + 0.3211h_{5,i} + 4.4181 \\ h_{3,i+1} &= 0.4285h_{2,i} + 0.5714h_{6,i} + 0.0857 \\ h_{5,i+1} &= 0.7028h_{2,i} + 0.1054h_{6,i} + 2.0223 \\ h_{6,i+1} &= 0.8695h_{3,i} + 0.1304h_{5,i} \end{aligned}$$

1.3 2D Finite-Differenzen-Verfahren

Unser erstes Beispiel (Abschn. 1.2) diente dazu, Wasserbilanzen mit Differenzen-Verfahren einfach zu berechnen. Jetzt schauen wir uns das Finite-Differenzen-Verfahren genauer an, insbesondere geht es uns um die Entwicklung eines 2D Berechnungsverfahrens.

1.3.1 Wiederholung Grundlagen

- PDE

$$S\frac{\partial h}{\partial t} - \frac{\partial}{\partial x}\left(K_x\frac{\partial h}{\partial x}\right) - \frac{\partial}{\partial y}\left(K_y\frac{\partial h}{\partial y}\right) = Q \tag{1.27}$$

- TSE

  in time

$$u_j^{n+1} = \sum_{m=0}^{\infty}\frac{\Delta t^m}{m!}\left[\frac{\partial^m u}{\partial t^m}\right]_j^n \tag{1.28}$$

  in space

$$u_{j+1}^{n} = \sum_{m=0}^{\infty}\frac{\Delta x^m}{m!}\left[\frac{\partial^m u}{\partial x^m}\right]_j^n \tag{1.29}$$

- Zeitableitung

By rearranging the above equations, a finite difference expression for first-order derivatives can be directly obtained.

$$\left[\frac{\partial u}{\partial t}\right]_j^n = \frac{u_j^{n+1} - u_j^n}{\Delta t} - \frac{\Delta t}{2}\left[\frac{\partial^2 u}{\partial t^2}\right]_j^n + 0(\Delta t^2) \tag{1.30}$$

- in space

The FD scheme for a second-order derivative can be obtained by adding the first-order schemes given in eqn (**??**). From the TSE can be seen that the symmetrical, central differences scheme is of second-order accuracy.

$$\left[\frac{\partial^2 u}{\partial x^2}\right]_j^n = \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{\Delta x^2} + \frac{\Delta x^2}{12}\left[\frac{\partial^4 u}{\partial x^4}\right]_j^n + ... \tag{1.31}$$

1.3.2 FDM Schema

Wir entwickeln ein explizites finite Differenzen (FD) Schema für die Lösung der 2D Grundwassergleichung.

$$\begin{aligned} S_{i,j}\frac{u_{i,j}^{n+1}-u_{i,j}^{n}}{\Delta t} \\ -K_{i,j}^{x}\frac{u_{i+1,j}^{n}-2u_{i,j}^{n}+u_{i-1,j}^{n}}{\Delta x^2} \quad - \quad K_{i,j}^{y}\frac{u_{i,j+1}^{n}-2u_{i,j}^{n}+u_{i,j-1}^{n}}{\Delta y^2} = Q_{i,j} \end{aligned} \tag{1.32}$$

$$\begin{aligned} u_{i,j}^{n+1} &= u_{i,j}^{n} \\ &+ \frac{K_{i,j}^{x}}{S_{i,j}}\frac{\Delta t}{\Delta x^2}u_{i+1,j}^{n}-2u_{i,j}^{n}+u_{i-1,j}^{n} \\ &+ \frac{K_{i,j}^{y}}{S_{i,j}}\frac{\Delta t}{\Delta y^2}u_{i,j+1}^{n}-2u_{i,j}^{n}+u_{i,j-1}^{n} \\ &+ \frac{Q_{i,j}}{S_{i,j}} \end{aligned} \tag{1.33}$$

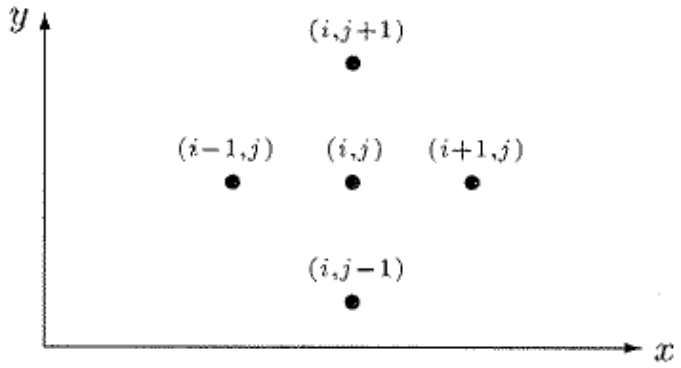

Abbildung 1.7: 5-Punkte-Stern (Knabner und Angermann 2000)

1.3.3 Programmierung I - QAD

Zunächst wollen wir versuchen, das Rechenschema (1.33) direkt umzusetzen (erstmal ohne über Objekt-Orientierung etc. nachzudenken). Daher erstmal die Frage, welche Rechengrößen haben wir denn überhaupt.

1.3.3.1 Datenstrukturen

Tabelle 1.1:

Feldgröße	u
Physikalische Parameter	S, K, Q
Numerische Parameter	$\Delta t, \Delta x, \Delta y$

Die Minimal-Datenstrukturen für die Programmierung der Gleichung (1.33) sind damit:

```
std::vector<float>u_new;
std::vector<float>u_old;
float S0,Kf,Q;
float dx,dy,dt;
```

Die Abb. 1.8 zeigt uns ein einfaches (rechteckiges) Rechen-Gitter für die erste Übung. Das finite Differenzen Gitter besteht aus 21×11 Knoten (21 in x

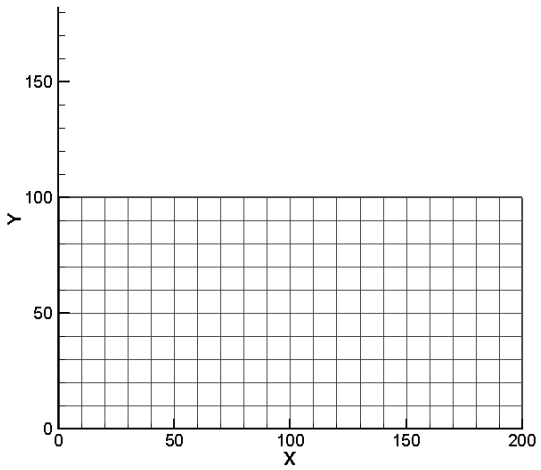

Abbildung 1.8: Rechen-Gitter für den Rechteck-Aquifer

1.3.3.2 Rechenschema

Um dieses Gitter "abtasten" zu können, schreiben wir folgende doppelte Schleife.

```
for(j=0;j<jy;j++)
{
  nn = j*ix;
  for( i=0;i<ix;i++)
  {
    n = nn+i;
    u_new[n] = u[n] \
             + Kf/S0*dt/dx2 * (u[n+1]-2*u[n]+u[n-1]) \
             + Kf/S0*dt/dy2 * (u[(j+1)*ix+i]-2*u[n]+u[(j-1)*ix+i]) \
             + Q/S0;
  }
}
```

Dabei ist j der Laufindex über die y Richtung und i der Laufindex über die x Richtung. Ganz wichtig ist natürlich, den Speicher für die Vektoren bereitzustellen, bevor es los geht.

```
u.resize(ix*jy);
u_new.resize(ix*jy);
```

- Welche Rolle spielen `ix` und `jy` bei der Speicherreservierung?

Natürlich müssen auch die Parameter vor der Berechnung initialisiert werden

```
  ix = 21;
  jy = 11;
  dx = 10.;
  dy = 10.;
  dt = 0.25e2;
  S0 = 1e-5;
  Kf = 1e-5;
  Q  = 0.;
  u0 = 0.;
```

- Welche Einheiten haben die einzelnen Parameter?

Das ist eigentlich schon alles, was wir für ein explizites Rechenschema benötigen ... oder ... Natürlich wissen wir, dass wir zur Lösung von PDEs (partiellen Differentialgleichungen für die Beschreibung von Anfangs-Randwert-Problemen) uns auch mit Anfangs- und Randbedingungen beschäftigen müssen.

1.3.3.3 Anfangsbedingungen

Das mit den Anfangsbedingungen ist eine einfache Sache. Mit der Doppelschleife über alle Knoten, können wir sehr einfach einen Wert `u0` als Anfangsbedingung überall zuweisen.

```
for(int i=0;i<ix;i++)
  for(int j=0;j<jy;j++)
  {
    u[j*(ix+1)] = u0;
    u_new[j*(ix+1)] = u0;
  }
}
```

1.3.3.4 Randbedingungen

Mit den Randbedingungen ist es etwas kniffliger. Wie der Name schon sagt, müssen diese Bedingungen auf die Gebietsränder aufgetragen werden, dass heisst, es müssen bestimmte Knoten identifiziert werden. Der nachfolgende Code zeigt die Zuweisung von Randbedingungen oben und unten sowie links und rechts.

```
//top and bottom
int l;
for(int i=0;i<ix;i++)
{
  bc_nodes.push_back(i); u[i] = u_top  u_new[i] = u_top;
  l = ix*(jy-1)+i;
  bc_nodes.push_back(l); u[l] = u_bottom;  u_new[l] = u_bottom;
}
//left and right side
for(int j=1;j<jy-1;j++)
{
  l = ix*j;
  bc_nodes.push_back(l); u[l] = u_left;  u_new[l] = u_left;
  l = ix*j+ix-1;
  bc_nodes.push_back(l); u[l] = u_right;  u_new[l] = u_right;
}
```

Sie sehen, dass wir für die Zuweisung der Randbedingungen eine neue Datenstruktur eingeführt haben.

```
std::vector<float>u_bc;
```

Das Einbauen der Randbedingungen integrieren wir direkt in die Doppelschleife zur Berechnung der Knotenwerte. Dabei kommt eine neue Funktion `IsBCNode` ins Spiel, die wir uns gleich noch näher anschauen. `IsBCNode` soll eigentlich

nichts anderes machen, als beim Auftreten einer Randbedingung nichts zu tun (i.e. `continue`). Randbedingungswerte sind gesetzt, müssen also nicht gerechnet werden.

```
for(int j=0;j<jy;j++)
{
  nn = j*ix;
  for(int i=0;i<ix;i++)
  {
    n = nn+i;
    if(IsBCNode(n,bc_nodes))
      continue;
    ...
  }
}
```

Wie funktioniert nun `IsBCNode`?

```
bool IsBCNode(int n,std::vector<int>bc_nodes)
{
  bool is_node_bc = false;
  for(int k=0;k<(size_t)bc_nodes.size();k++)
  {
    if(n==bc_nodes[k])
    {
      is_node_bc = true;
      return is_node_bc;
    }
  }
  return is_node_bc;
}
```

Struktur der Funktion:

- Rückgabewert: logischer Wert wahr oder falsch
- Parameter: aktueller Gitterpunkt und Randbedingungsknotenvektor

Die Funktion überprüft, ob der Gitterpunkt n ein Randbedingungsknoten ist und gibt den entsprechenden logischen Wert zurück.

Das Ergebnis der finite Differenzen Simulation sehen wir in der Abb. 1.9.

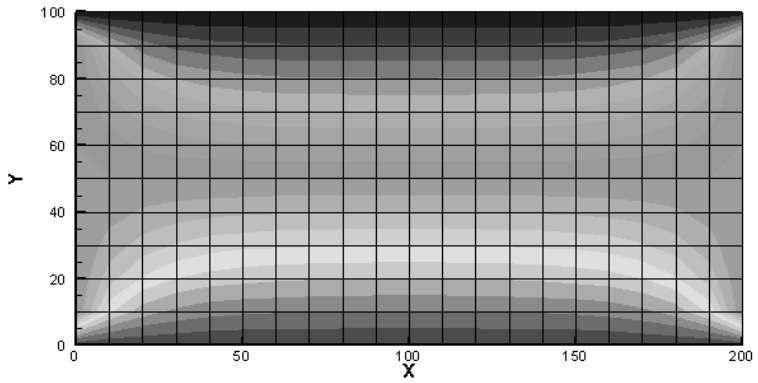

Abbildung 1.9: Berechnete Druckverteilung im Rechteck-Aquifer nach 100 Zeitschritten $\Delta t = 25$ sec

Das sieht doch schon mal sehr gut aus. Jetzt werden wir mutig und vergrößern mal den Zeitschritt, sagen wir mal verdoppeln: $\Delta t = 50$ sec. Das Maleur sehen wir in der Abb. 1.10. Was ist hier los?

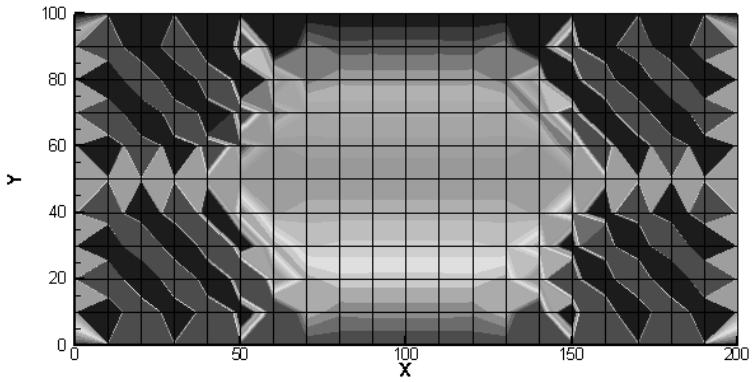

Abbildung 1.10: Berechnete Druckverteilung im Rechteck-Aquifer nach 100 Zeitschritten $\Delta t = 50$ sec

Das Stabilitätskriterium scheint wirklich ernst gemeint zu sein.

1.3.3.5 Stabilitätsbetrachtung

Wir erinnern uns noch dunkel daran, dass der Preis für das einfache explizite FDM ein strenges Stabilitätskriterium war (siehe Hydroinformatik, Teil II, Abschn. 3.2.2 und Abschn. 4.1). Dabei muss die Neumann-Zahl kleiner einhalb sein.

$$Ne = \alpha \frac{\Delta t}{\Delta x^2} \leq \frac{1}{2} \tag{1.34}$$

Prima, aber was ist jetzt α und warum steht nur Δx und nicht auch Δy in der Gleichung? Zur bestimmung des α schreiben wir die Grundwassergleichung in eine Diffusionsgleichung wie folgt um.

$$\frac{\partial h}{\partial t} = \frac{K_x}{S}\frac{\partial^2 h}{\partial x^2} + \frac{K_y}{S}\frac{\partial^2 h}{\partial y^2} + \frac{Q}{S} \tag{1.35}$$

Wir sehen, dass es eigentlich zwei α-s gibt, für jede Richtung eins.

$$\begin{aligned} \alpha_x &= \frac{K_x}{S} \\ \alpha_y &= \frac{K_y}{S} \end{aligned} \tag{1.36}$$

- Welche Einheit hat unser Grundwasser-α ?

Der richtige Zeitschritt für unser explizites FD Verfahren ergibt sich somit zu:

$$\Delta t \leq \frac{\min(\Delta x^2, \Delta y^2)}{2\alpha} \tag{1.37}$$

Die numerischen und hydraulischen Parameter sind in der nachfolgenden Tabelle 1.2 zu finden. Glücklicherweise sind die Ortdiskretisierungen und die hydraulischen Leitfähigkeiten in beide Koordinatenrichtungen gleich (isotropes Problem).

Damit ergibt sich für den maximalen Zeitschritt des expliziten FD Verfahrens:

$$\Delta t \leq \frac{100 m^2}{2 \times 1 m^2/s} = 50 s \tag{1.38}$$

Tabelle 1.2: Parameter

Symbol	Parameter	Value	Unit
Δx	Ortsdiskretisierung in x-Richtung	10	m
Δy	Ortsdiskretisierung in y-Richtung	10	m
$K_x = K_y$	Hydraulische Leitfähigkeit	10^{-5}	ms^{-1}
S	Spezifischer Speicherkoeffizient	10^{-5}	m^{-1} height

Der Nachteil des expliziten FD Verfahrens ist zwar die rigorose Zeitschrittweitenbegrenzung. Andererseits kann man mit dem expliziten Verfahren selbst auf kleinen Rechnern (wie z.B. mein Notebook) sehr große Probleme zügig rechnen. Die Abb. 1.11 zeigt die benötigten Rechenzeiten und den Speicherbedarf für verschiedene Problemgrößen (Anzahl der Gitterpunkte auf der x-Achse).

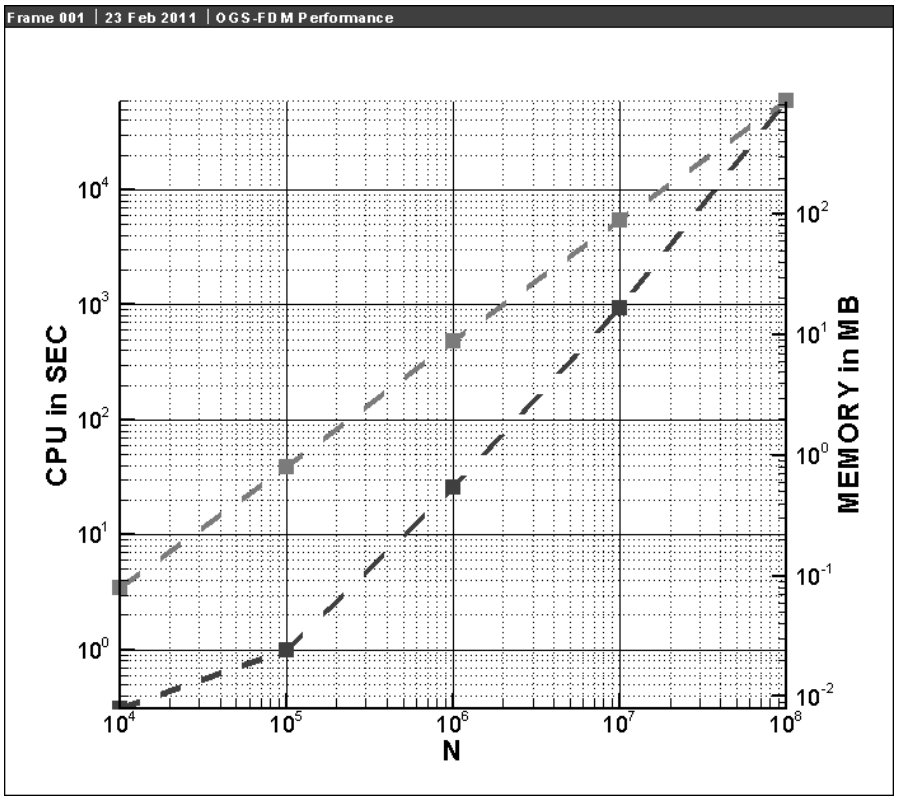

Abbildung 1.11: Rechenzeit und Speicherbedarf für explizite FDM

Wir können also bis zu 10^8 Gitterpunkte auf 1 GB RAM rechnen (das dauert dann allerdings einen ganzen Tag, nehmen sie sich mit ihrem Notebook an

diesem Tag also nichts anderes vor ...) oder 10^6 Gitterpunkte in einer halben Minute.

Wie stellen wir eine Zeitmessung in einem Programm an.

```
clock_t start, end; // Definitionen
...
start = clock();    // Beginn Zeitmessung
...
end = clock();      // Ende Zeitmessung
...
time= (end-start)/ (double)(CLOCKS_PER_SEC); // Differenz
```

Übung GW1 Der Quelltext für diese Übung befindet sich in `EXERCISES\GW1.`

1.4 Selke 2D FDM Modell

Wir haben bei der Vorstellung des Bode-Einzugsgebietes gesehen, dass reale Catchments eher weniger wie Rechtecke aussehen. Dennoch können wir mit Hilfe der FDM schon eine ganz gute geometrische Approximation von Einzugsgebieten hinbekommen. Wenn das Alles nichts nützt müssen wir halt doch auch geometrisch genauere Verfahren, wie z.B. die Finite-Elemente-Methode (FEM, dazu später in unseren Kurs), zurückgreifen.

Die Abb. 1.12 zeigt uns eine mögliche Approximation des Selke-Einzugsgebietes mit einer relativ geringen Anzahl von FD Zellen (32x42=1344).

- Aus wie vielen FD Knoten besteht das FD Mesh in Abb. 1.12?

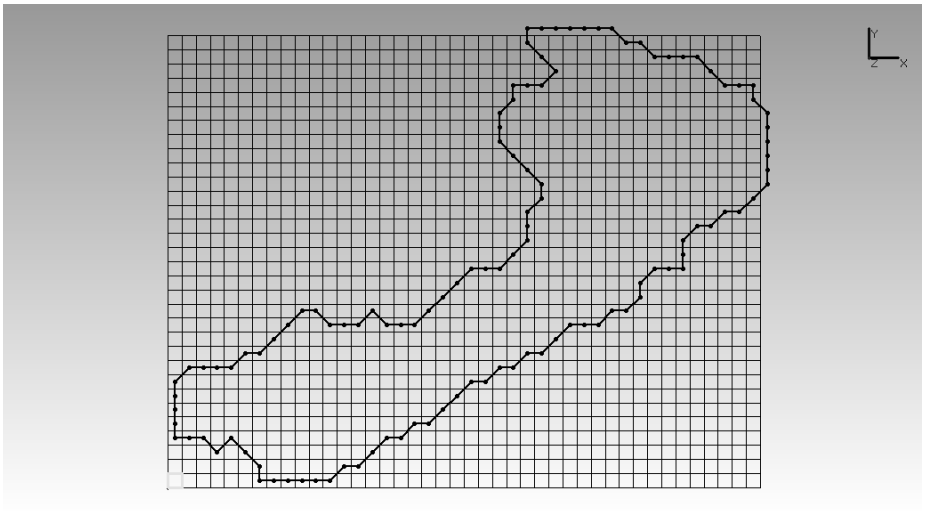

Abbildung 1.12: Rastermodell für das Selke-Einzugsgebiet dargestellt mit GINA#OGS (Herbert Kunz, BGR)

Wie bekommen wir aus unserem regelmäßigen rechteckigen Raster ein eher unregelmäßiges Catchment herausgeschnitten? Der Trick besteht darin, einzelne Zellen zu deaktivieren. Das klingt schon wieder nach Arbeit, ist aber machbar, dafür gibt's die nächste Übung (GW2). Die geometrische Analyse mit OGS liefert uns zunächst eine Liste von Gitterpunkten die ausserhalb des Catchments liegen (siehe Übung GW2):

- ExtractedSelkeMeshIDs.txt
- selke.gli

Diese Files können wir mal mit dem OGS-DatenExplorer (ogs-gui.exe) laden.

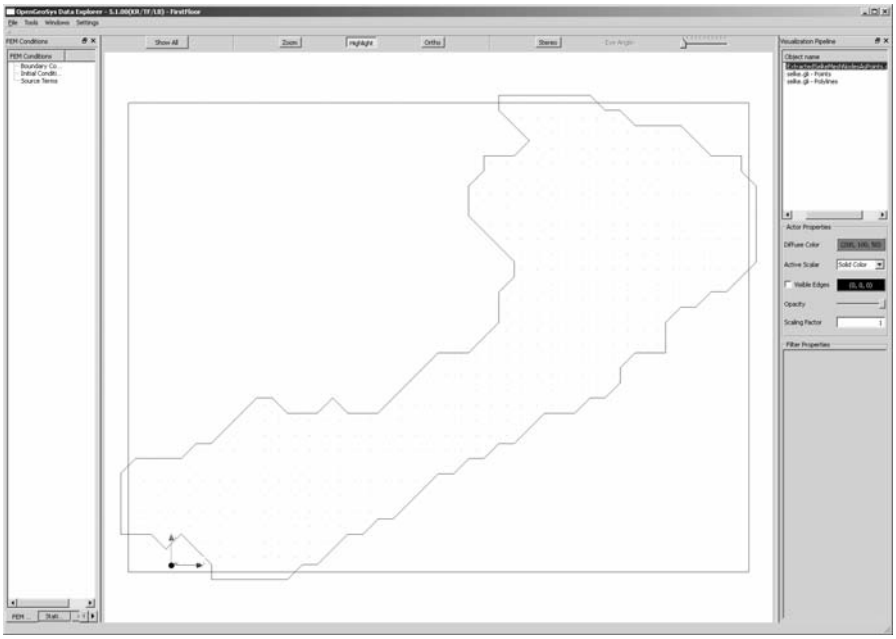

Abbildung 1.13: Das sind zwar die Daten, sieht aber noch nach nix aus ...

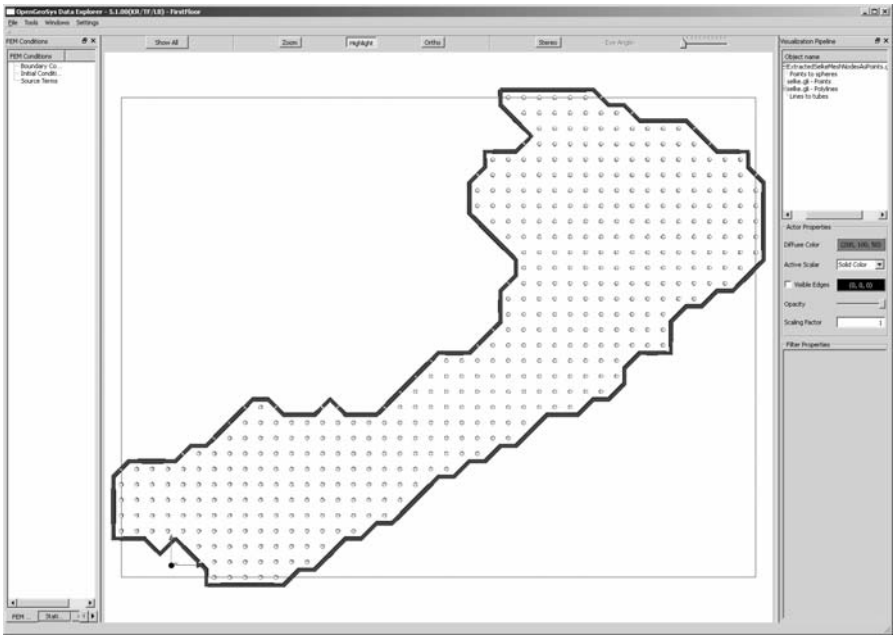

Abbildung 1.14: OGS hat ein paar nette VTK-Filter, um geometrische Objekte herauszuheben, dargestellt mit OGS5

Das ist zwar alles schön und gut, was wir aber brauchen sind die Knoten ausserhalb des Catchments, damit wir diese für das FD Verfahren deaktivieren können. Also müssen wir doch selber ran. Unser Plan ist wie folgt:

- Aktive Knoten lesen und speichern.
- Aktive Knoten sortieren (Gruß an Hydroinformatik I - Hantieren mit Listen)

```
std::list<int>nodes_active;
std::ifstream active_nodes_file;
active_nodes_file.open("ActiveNodes.txt");
int na;
while(!active_nodes_file.eof())
{
  active_nodes_file >> na;
  nodes_active.push_back(na);
}
nodes_active.sort();
```

- (das Zwischenergebnis zur Sicherheit mal rausschreiben)

```
std::ofstream active_nodes_file_test;
active_nodes_file_test.open("ActiveNodesTest.txt");
list<int>::const_iterator p = nodes_active.begin();
while(p!=nodes_active.end())
{
  active_nodes_file_test << *p << endl;
  ++p;
}
active_nodes_file_test.close();
```

- Alle Knoten rausfischen, die NICHT aktiv sind.
- Dabei kommt eine neue Hilfs-Funktion `NodeInList` ins Spiel (siehe unten).

```
for(j=0;j<jy;j++)
{
  nn = j*ix;
  for( i=0;i<ix;i++)
  {
    n = nn+i;
    if(!NodeInList(n,nodes_active))
      nodes_inactive.push_back(n);
  }
}
```

- Wir überzeugen uns vom Ergebnis (File schreiben) ...

```
std::ofstream inactive_nodes_file;
inactive_nodes_file.open("InactiveNodes.txt");
for(i=0;i<nodes_inactive.size();i++)
{
  inactive_nodes_file << nodes_inactive[i] << endl;
}
inactive_nodes_file.close();
```

- ... und natürlich graphisch, wozu haben wir denn Visual C++ gelernt!

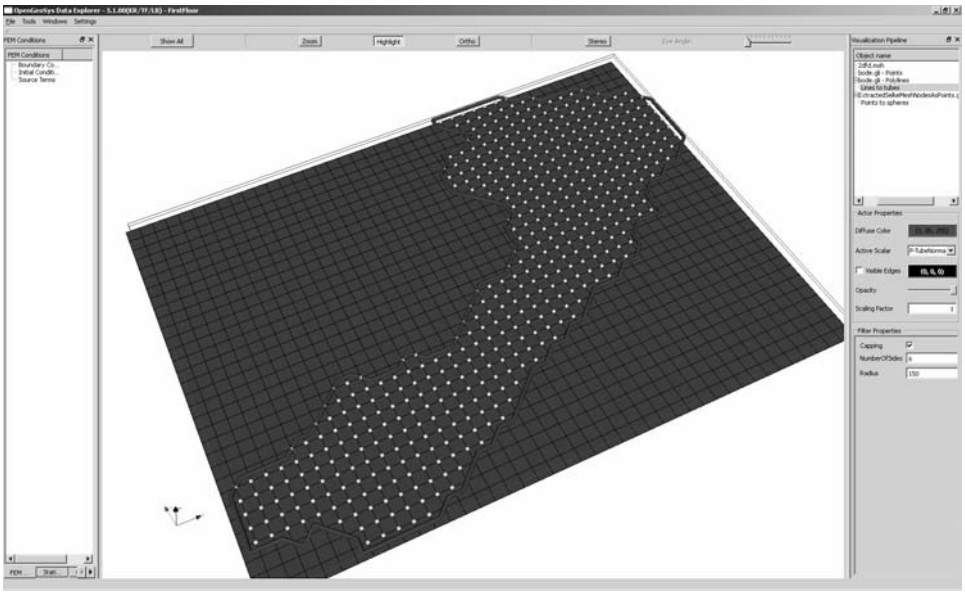

Abbildung 1.15: Darstellung aktiver / inaktiver Knoten)

Die nützliche Hilfs-Funktion, die alle Knoten raussucht, die NICHT in `nodes_active`stehen.

```
bool NodeInList(int n,std::list<int>nodes_active)
{
  list<int>::const_iterator p = nodes_active.begin();
  while(p!=nodes_active.end())
  {
   if(n==*p)
     return true;
    ++p;
  }
  return false;
}
```

1.5 FDM OOP

Wie sie bereits wissen, geht nachdem das FDM Programm funktioniert, die Arbeit erst richtig los. Aus dem QAD Programm wird nun ein OOP, d.h. wir legen eine C++ Klasse an und packen die einzelnen Rechenschritte in Methoden der Klasse. Das Ergebnis sehen wir in der neuen main Function: das ganze FDM passt nun auf eine halbe Seite Quelltext (wobei ein großer Teil noch für die Zeitmessung draufgeht).

```
#include <iostream>
#include "fdm.h"

int main(int argc, char *argv[])
{
  clock_t start, end;
  double cpuTime;
  std::ofstream aux_file;
  aux_file.open("../cputime.txt");
  start = clock();
  //-----------------------------------------------
  FDM* fdm = new FDM();
  fdm->SetActiveNodes();
  fdm->SetInactiveNodes();
  fdm->SetInitialConditions();
  fdm->SetBoundaryConditions();
  //-----------------------------------------------
  int tn = 100;
  for(int t=0;t<tn;t++)
  {
    fdm->RunTimeStep();
    fdm->SaveTimeStep();
    fdm->OutputResults(t);
  }
  //-----------------------------------------------
  end = clock();
  cpuTime= (end-start)/ (double)(CLOCKS_PER_SEC);
  aux_file << "CPU time:" << cpuTime << std::endl;
  aux_file.close();
  return 0;
}
```

Jetzt wenden wir das explizite FD Verfahren (Abschn. 1.3) auf unsere Problemstellung an. Das Ergebnis sehen wir in Abb. 1.16.

.

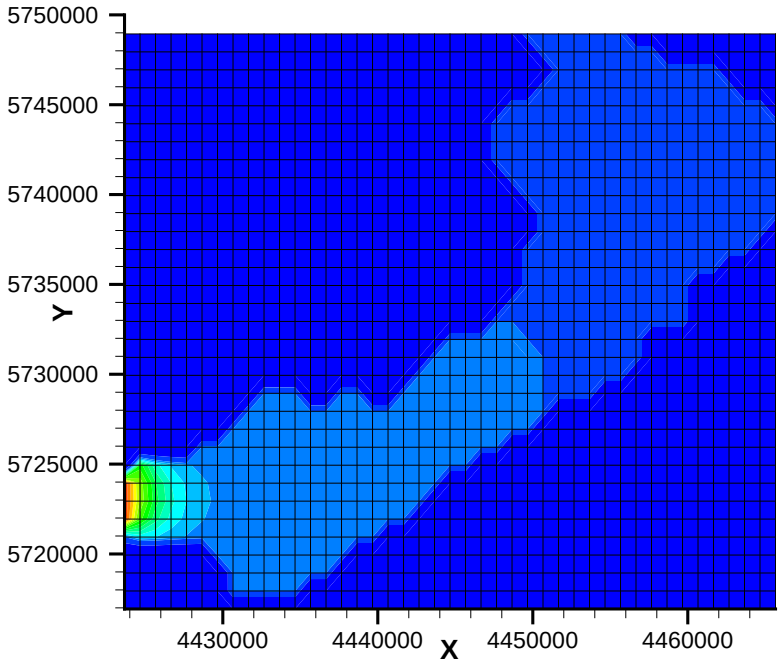

Abbildung 1.16: Ergebnis der FD Simulation für das Selke Gebiet

Übung GW3 Der Quelltext für diese Übung befindet sich in `EXERCISES\GW3`.

1.6 2D implizites Finite-Differenzen-Verfahren

siehe auch Abschn. 4.2 Hydroinformatik II

- Auswertung der Ableitungen zum neuen Zeitpunkt t^{n+1}

$$\left[\frac{\partial^2 h}{\partial x^2}\right]_{i,j}^{n+1} \approx \frac{h_{i-1,j}^{n+1} - 2h_{i,j}^{n+1} + h_{i+1,j}^{n+1}}{\Delta x^2} \tag{1.39}$$

$$\left[\frac{\partial^2 u}{\partial y^2}\right]_{i,j}^{n+1} \approx \frac{h_{i,j-1}^{n+1} - 2h_{i,j}^{n+1} + h_{i,j+1}^{n+1}}{\Delta x^2} \tag{1.40}$$

- Differenzen-Schema

$$\begin{aligned} & S_{i,j}\frac{u_{i,j}^{n+1} - u_{i,j}^{n}}{\Delta t} \\ -K_{i,j}^{x}\frac{u_{i+1,j}^{n+1} - 2u_{i,j}^{n+1} + u_{i-1,j}^{n+1}}{\Delta x^2} & - K_{i,j}^{y}\frac{u_{i,j+1}^{n+1} - 2u_{i,j}^{n+1} + u_{i,j-1}^{n+1}}{\Delta y^2} = Q_{i,j} \end{aligned} \tag{1.41}$$

- Gleichungssystem

$$\begin{aligned} & \left(\frac{S}{\Delta t} + 2\frac{K^x}{\Delta x^2} + 2\frac{K^y}{\Delta y^2}\right) u_{i,j}^{n+1} \\ - & \left(\frac{K^x}{\Delta x^2}\right)(u_{i-1,j}^{n+1} + u_{i+1,j}^{n+1}) - \left(\frac{K^y}{\Delta y^2}\right)(u_{i,j-1}^{n+1} + u_{i,j+1}^{n+1}) \\ = & \frac{S}{\Delta t}u_{i,j}^{n} + Q_{i,j} \end{aligned} \tag{1.42}$$

Wir vereinfachen die Gleichung (1.42), indem wir für den Moment annehmen, dass $K^x = K^y = K$ (Isotropie) und $\Delta x = \Delta y = \Delta l$ (gleichförmige Diskretisierung). Die Multiplikation mit $\Delta t/S$ ergibt dann folgende Beziehung.

$$\begin{aligned} & \left(1 + 4\frac{K\Delta t}{S\Delta l^2}\right) u_{i,j}^{n+1} \\ - & \left(\frac{K\Delta t}{S\Delta l^2}\right)(u_{i-1,j}^{n+1} + u_{i+1,j}^{n+1} + u_{i,j-1}^{n+1} + u_{i,j+1}^{n+1}) \\ = & u_{i,j}^{n} + \frac{\Delta t}{S}Q_{i,j} \end{aligned} \tag{1.43}$$

K : Vergleichen Sie die Beziehung (1.43) mit der Gleichung (4.10, Skript Hydroinformatik II).

Der Ausdruck $K/S = \alpha$ entspricht dem Diffusivitätskoeffizienten (Überprüfen sie dies anhand der Einheiten). Damit ist die Neumann-Zahl

$$Ne = \frac{K}{S}\frac{\Delta t}{\Delta l^2} \tag{1.44}$$

Nun versuchen wir anhand der Gleichung (1.43) die Struktur des zu lösenden Gleichungssystems zu beschreiben. Wir gehen wieder ganz genau so vor wie bei der 1D Diffusionsgleichung im Abschn. 4.2 (Hydroinformatik II).

$$\underbrace{\begin{bmatrix} 1+4Ne & -Ne & & & & & & & \\ -Ne & \dots & \dots & & & & & & \\ & \dots & \dots & \dots & & & & & \\ & & & -Ne & 1+4Ne & & & & \\ & & & & & 1+4Ne & -Ne & & \\ & & & & & & \dots & \dots & \dots \\ & & & & & & & -Ne & 1+4Ne \end{bmatrix}}_{\mathbf{A}}$$

$$\underbrace{\begin{bmatrix} u_{0,0}^{n+1} \\ u_{1,0}^{n+1} \\ \dots \\ u_{I-1,0}^{n+1} \\ u_{0,1}^{n+1} \\ \dots \\ u_{I-1,J-1}^{n+1} \end{bmatrix}}_{\mathbf{x}} = \underbrace{\begin{bmatrix} u_{0,0}^{n} + b_{0,0} \\ u_{1,0}^{n} + b_{1,0} \\ \dots \\ u_{I-1,0}^{n} + b_{I-1,0} \\ u_{0,1}^{n} + b_{0,1} \\ \dots \\ u_{I-1,J-1}^{n} + b_{I-1,J-1} \end{bmatrix}}_{\mathbf{b}} \tag{1.45}$$

Auch was die Programmierung betrifft, können wir auf unsere Erfahrungen in Hydroinformatik II aufbauen. Es gibt praktisch keinen Unterschied, ob wir es mit einem 1D oder 2D Problem zu tun haben. Wir müssen lediglich aufpassen, dass wir die Indizes richtig zählen.

Wir benutzen die Grundstruktur des objekt-orientierten Programms für das explizite FD Verfahren (Abschn. 1.5). Die wesentlichen Unterschiede der impliziten zur expliziten FDM sind, dass wir ein Gleichungssystem aufbauen und lösen müssen.

```
#include <iostream>
#include "fdm.h"
#include <time.h>

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

int main(int argc, char *argv[])
{
  //-----------------------------------------------
  FDM* fdm = new FDM();
  fdm->SetInitialConditions();
  fdm->SetBoundaryConditions();
  //-----------------------------------------------
  int tn = 2;
  for(int t=0;t<tn;t++)
  {
    fdm->AssembleEquationSystem();
    Gauss(fdm->matrix,fdm->vecb,fdm->vecx,fdm->IJ);
    fdm->SaveTimeStep();
    fdm->OutputResults(t);
  }
  //-----------------------------------------------
  fdm->out_file.close();
  return 0;
}
```

Dennoch können wir erstaunlich viel wiederverwenden, bis auf

```
fdm->AssembleEquationSystem();
Gauss(fdm->matrix,fdm->vecb,fdm->vecx,fdm->IJ);
```

Der Gleichungslöser `Gauss` ist übrigens genau der gleiche, den wir schon für die Lösung des impliziten FD Verfahrens für die Diffusionsgleichung in Hydroinformatik II benutzt haben. Der Reihe nach. Die Assemblierfunktion soll das Gleichungssystem (1.45) aufbauen. Vom Prinzip her das Gleiche wie beim 1D FD Verfahren:

- Die Hauptdiagonale bekommt den Wert $1 + 4Ne$,
- die Nebendiagonalen haben den Wert $-Ne$.

Dies lässt sich programmtechnisch recht einfach bewerkstelligen (sie erinnern sich, wie wir in einer Doppelschleife, die Hauptdiagonale herausfinden können)

```
void FDM::AssembleEquationSystem()
{
  // Matrix entries
  for(i=0;i<IJ;i++)
```

```
  {
    vecb[i] = u[i];
    for(j=0;j<IJ;j++)
    {
      matrix[i*IJ+j] = 0.0;
      if(i==j)
        matrix[i*IJ+j] = 1. + 4.*Ne;
      else if(abs((i-j))==1)
        matrix[i*IJ+j] = - Ne;
    }
  }
  // Incorporate boundary conditions
  IncorporateBoundaryConditions();
  // Matrix output
  WriteEquationSystem();
}
```

Um die Assemblierfunktion zu testen bauen wir uns ein ganz einfaches Beispiel bestehend aus nur 9 Knoten (Abb. 1.17) - je einfacher, desto besser.

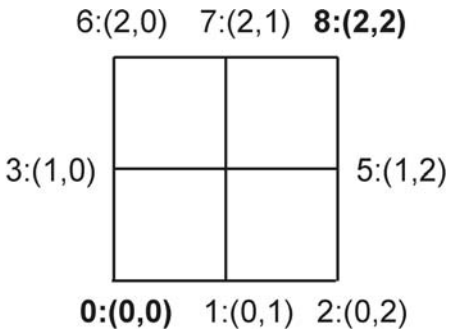

Abbildung 1.17: Testbeispiel

Mit Hilfe der nützlichen Funktion `WriteEquationSystem()`können wir das Gleichungssystem in eine Datei schreiben, das Ergebnis passt.

```
2 -0.25 0 0 0 0 0 0 0
-0.25 2 -0.25 0 0 0 0 0 0
0 -0.25 2 -0.25 0 0 0 0 0
0 0 -0.25 2 -0.25 0 0 0 0
0 0 0 -0.25 2 -0.25 0 0 0
0 0 0 0 -0.25 2 -0.25 0 0
0 0 0 0 0 -0.25 2 -0.25 0
0 0 0 0 0 0 -0.25 2 -0.25
0 0 0 0 0 0 0 -0.25 2
```

Etwas kniffliger is es mit dem Einbauen der Randbedingungen. Wir erinnern uns, der Trick war eine Manipulation der Matrix und des RHS (right-hand-side) vectors, um den vorgegebenen Wert der Randbedingung zu erzwingen. Der Code zeigt das Beispiel für den Einbau einer Randbedingung im Knoten (also oben rechts in unseren kleinen Testbeispiel).

```
void FDM::IncorporateBoundaryConditions()
{
  //-----------------------------------
  size_t i_bc;
  int i_row, k;
  for(i_bc=0;i_bc<bc_nodes.size();i_bc++)
  {
     i_row = bc_nodes[i_bc];
     // Null off-diangonal entries of the related row and columns
     // Apply contribution to RHS by BC
     for(j=0;j<IJ;j++)
     {
        if(i_row == j)
           continue; // do not touch diagonals
        matrix[i_row*(IJ)+j] = 0.0; // NUll row
        k = j*(IJ)+i_row;
        // Apply contribution to RHS by BC
        vecb[j] -= matrix[k]*u[i_row];
        matrix[k] = 0.0; // Null column
     }
     // Apply Dirichlet BC
     vecb[i_row] = u[i_row]*matrix[i_row*(IJ)+i_row];
  }
}
```

Wir schreiben wieder das Gleichungssystem mit `WriteEquationSystem()` in eine Datei und schauen uns jeden Schritt genau an.

- Diagonalelemente werden nicht angefasst.
- Reihe zu Null setzen

```
2 0 0 0 0 0 0 0 0         b: 1
-0.25 2 -0.25 0 0 0 0 0 0 b: 0
0 -0.25 2 -0.25 0 0 0 0 0 b: 0
0 0 -0.25 2 -0.25 0 0 0 0 b: 0
0 0 0 -0.25 2 -0.25 0 0 0 b: 0
0 0 0 0 -0.25 2 -0.25 0 0 b: 0
0 0 0 0 0 -0.25 2 -0.25 0 b: 0
0 0 0 0 0 0 -0.25 2 -0.25 b: 0
0 0 0 0 0 0 0 0 2         b: -1
```

- Rechte Seite manipulieren

```
2 0 0 0 0 0 0 0 0          b: 1
-0.25 2 -0.25 0 0 0 0 0 0 b: 0.25
0 -0.25 2 -0.25 0 0 0 0 0 b: 0
0 0 -0.25 2 -0.25 0 0 0 0 b: 0
0 0 0 -0.25 2 -0.25 0 0 0 b: 0
0 0 0 0 -0.25 2 -0.25 0 0 b: 0
0 0 0 0 0 -0.25 2 -0.25 0 b: 0
0 0 0 0 0 0 -0.25 2 -0.25 b: -0.25
0 0 0 0 0 0 0 0 2          b: -1
```

- Spalte Null setzen

```
2 0 0 0 0 0 0 0 0          b: 1
0 2 -0.25 0 0 0 0 0 0      b: 0.25
0 -0.25 2 -0.25 0 0 0 0 0 b: 0
0 0 -0.25 2 -0.25 0 0 0 0 b: 0
0 0 0 -0.25 2 -0.25 0 0 0 b: 0
0 0 0 0 -0.25 2 -0.25 0 0 b: 0
0 0 0 0 0 -0.25 2 -0.25 0 b: 0
0 0 0 0 0 0 -0.25 2 0      b: -0.25
0 0 0 0 0 0 0 0 2          b: -1
```

- Neumann Randbedingungen setzen

```
2 0 0 0 0 0 0 0 0          b:2
0 2 -0.25 0 0 0 0 0 0      b:0.25
0 -0.25 2 -0.25 0 0 0 0 0 b:0
0 0 -0.25 2 -0.25 0 0 0 0 b:0
0 0 0 -0.25 2 -0.25 0 0 0 b:0
0 0 0 0 -0.25 2 -0.25 0 0 b:0
0 0 0 0 0 -0.25 2 -0.25 0 b:0
0 0 0 0 0 0 -0.25 2 0      b:-0.25
0 0 0 0 0 0 0 0 2          b:-2
```

Schließlich ergibt sich folgendes Gleichungssystem zur Lösung durch das Gauss-Verfahren noch mal richtig aufgeschrieben.

$$
\begin{aligned}
2h_1^{n+1} &= 2h_1^n \\
2h_2^{n+1} - 0.25h_3^{n+1} &= h_2^n + 0.25 \\
-0.25h_2^{n+1} + 2h_3^{n+1} - 0.5h_4^{n+1} &= h_3^n \\
-0.25h_3^{n+1} + 2h_4^{n+1} - 0.5h_5^{n+1} &= h_4^n \\
-0.25h_4^{n+1} + 2h_5^{n+1} - 0.5h_6^{n+1} &= h_5^n \\
-0.25h_5^{n+1} + 2h_6^{n+1} - 0.5h_7^{n+1} &= h_6^n \\
-0.25h_6^{n+1} + 2h_7^{n+1} - 0.5h_8^{n+1} &= h_7^n
\end{aligned}
$$

$$\begin{aligned} -0.25h_7^{n+1} + 2h_8^{n+1} &= h_8^n - 0.25 \\ 2h_9^{n+1} &= -2 \end{aligned} \tag{1.46}$$

Das geschriebene Ergebnisfile sieht dann folgendermaßen aus.

```
ZONE T="0.25", I=3, J=3, DATAPACKING=POINT
0 0 1
1 0 0.127016
2 0 0.016129
0 1 0.00201613
1 1 0
2 1 -0.00201613
0 2 -0.016129
1 2 -0.127016
2 2 -1
ZONE T="100.", I=3, J=3, DATAPACKING=POINT
0 0 1
1 0 0.267857
2 0 0.0714286
0 1 0.0178571
1 1 1.80718e-19
2 1 -0.0178571
0 2 -0.0714286
1 2 -0.267857
2 2 -1
```

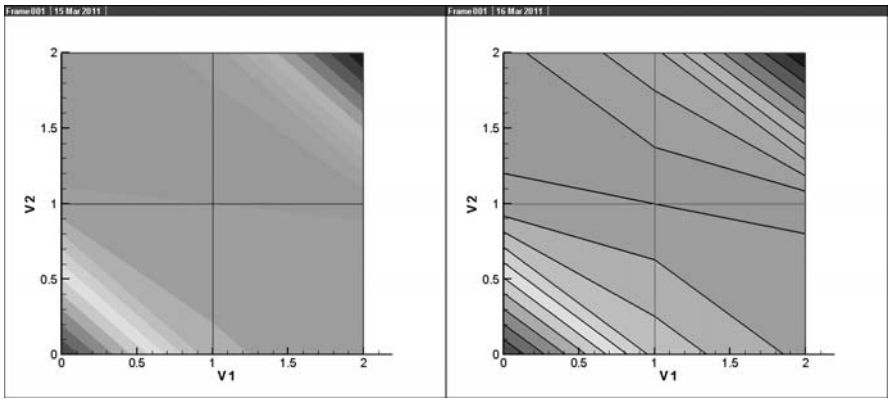

Abbildung 1.18: Ergebnisse des impliziten FD Verfahrens für t=0.25, 10 sec

Übung GW4 Der Quelltext für diese Übung befindet sich in `EXERCISES\GW4.`

1.6.1 Stationäres Problem

$S = 0$

- Gleichung

$$-K^x_{i,j}\frac{u^{n+1}_{i+1,j} - 2u^{n+1}_{i,j} + u^{n+1}_{i-1,j}}{\Delta x^2} \quad - \quad K^y_{i,j}\frac{u^{n+1}_{i,j+1} - 2u^{n+1}_{i,j} + u^{n+1}_{i,j-1}}{\Delta y^2} = Q_{i,j} \tag{1.47}$$

- Gleichungssystem

$$\begin{aligned} & 2\left(\frac{K^x}{\Delta x^2} + \frac{K^y}{\Delta y^2}\right) u^{n+1}_{i,j} \\ - \quad & \left(\frac{K^x}{\Delta x^2}\right)(u^{n+1}_{i-1,j} + u^{n+1}_{i+1,j}) - \left(\frac{K^y}{\Delta y^2}\right)(u^{n+1}_{i,j-1} + u^{n+1}_{i,j+1}) \\ = \quad & Q_{i,j} \end{aligned} \tag{1.48}$$

Wir vereinfachen die Gleichung (1.48) wieder, indem wir für den Moment annehmen, dass $K^x = K^y = K$ (Isotropie) und $\Delta x = \Delta y = \Delta l$ (gleichförmige Diskretisierung).

$$\begin{aligned} & \left(\frac{4}{\Delta l^2}\right) u^{n+1}_{i,j} \\ - \quad & \left(\frac{1}{\Delta l^2}\right)(u^{n+1}_{i-1,j} + u^{n+1}_{i+1,j} + u^{n+1}_{i,j-1} + u^{n+1}_{i,j+1}) \\ = \quad & \frac{Q_{i,j}}{K} \end{aligned} \tag{1.49}$$

- Gleichungssystem in Matrizenschreibweise

$$\underbrace{\begin{bmatrix} 4 & -1 & & & & \\ -1 & 4 & -1 & & & \\ & \cdots & \cdots & \cdots & & \\ & & & -1 & 4 & -1 \\ & & & & -1 & 4 \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} h_1 \\ \cdots \\ \cdots \\ \cdots \\ \cdots \\ h_{IJ} \end{bmatrix}}_{\mathbf{x}} = \underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}}_{\mathbf{b}} \tag{1.50}$$

- Gleichungssystem nach Einbau der Randbedingungen

```
4 0 0 0 0 0 0 0 0 b:4
0 4 -1 0 0 0 0 0 0 b:1
0 -1 4 -1 0 0 0 0 0 b:0
0 0 -1 4 -1 0 0 0 0 b:0
0 0 0 -1 4 -1 0 0 0 b:0
0 0 0 0 -1 4 -1 0 0 b:0
0 0 0 0 0 -1 4 -1 0 b:0
0 0 0 0 0 0 -1 4 0 b:-1
0 0 0 0 0 0 0 0 4 b:-4
```

- Ergebnisse

```
ZONE T="BIG ZONE", I=3, J=3, DATAPACKING=POINT
0 0 1
1 0 0.267857
2 0 0.0714286
0 1 0.0178571
1 1 -9.29613e-19
2 1 -0.0178571
0 2 -0.0714286
1 2 -0.267857
2 2 -1
```

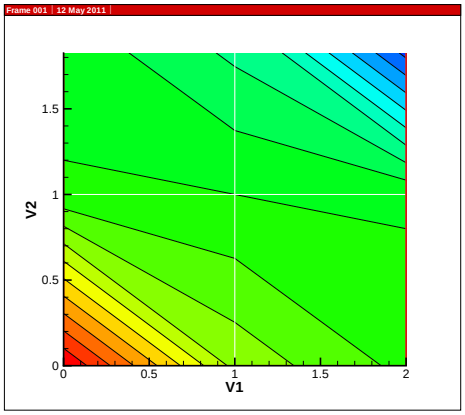

Abbildung 1.19: Testbeispiel mit Dirichlet-Randbedingungen

K : Vergleichen sie die Ergebnisse mit den der instationären Berechnung.

K : Ändern sie die Problemgröße (Vergrößern von `ix,iy`) und vergleichen sie die Ergebnisse..

Übung GW4a Der Quelltext für diese Übung befindet sich in `EXERCISES\GW4a.`

1.6.2 Neumann-Randbedingungen

- Differenzengleichung für Knoten mit Neumann-Randbedingungen
- Richtungsabhängigkeiten

$$\frac{-u_{i-1,j} - u_{i+1,j} + 4u_{i,j} - u_{i,j-1} - u_{i,j+1}}{\Delta x^2} \tag{1.51}$$

$$\frac{\partial u}{\partial x} = 0 \quad \rightarrow \quad u_{i-1,j} = u_{i+1,j} \tag{1.52}$$

$$\frac{\partial u}{\partial y} = 0 \quad \rightarrow \quad u_{i,j-1} = u_{i,j+1} \tag{1.53}$$

- $\partial u/\partial x$, left boundary

$$\frac{-2u_{i+1,j} + 4u_{i,j} - u_{i,j-1} - u_{i,j+1}}{\Delta l^2} \tag{1.54}$$

- $\partial u/\partial x$, right boundary

$$\frac{-2u_{i-1,j} + 4u_{i,j} - u_{i,j-1} - u_{i,j+1}}{\Delta l^2} \tag{1.55}$$

- $\partial u/\partial y$, top boundary

$$\frac{-u_{i-1,j} - u_{i+1,j} + 4u_{i,j} - 2u_{i,j-1}}{\Delta l^2} \tag{1.56}$$

- $\partial u/\partial y$, bottom boundary

$$\frac{-u_{i-1,j} - u_{i+1,j} + 4u_{i,j} - 2u_{i,j+1}}{\Delta l^2} \tag{1.57}$$

- Knotengleichungen

$$\begin{aligned} 4u_1 - u_2 - u_4 &= 0 \\ -u_1 + 4u_2 - u_3 - 2u_5 &= 0 \\ -2u_2 + 4u_3 - 2u_6 &= 0 \\ -u_1 + 4u_4 + 2u_5 - u_7 &= 0 \end{aligned}$$

$$
\begin{aligned}
-u_2 - u_4 + 4u_5 - u_6 - u_8 &= 0 \\
-u_3 - 2u_5 + 4u_6 - u_9 &= 0 \\
-2u_4 + 4u_7 - 2u_8 &= 0 \\
-2u_5 - u_7 + 4u_8 - u_9 &= 0 \\
-u_6 - u_8 + u_9 &= 0
\end{aligned}
\tag{1.58}
$$

```
 4 -1  0 -1  0 0  0 0  0
-1 4 -1 0 -1 0  0 0  0
 0 -1  4 -1  0 -1  0 0  0
-1 0 -1 4 -1 0 -1 0  0
 0 -1  0 -1  4 -1  0 -1  0
 0 0 -1 0 -1 4 -1 0 -1
 0 0  0 -1  0 -1  4 -1  0
 0 0  0 0 -1 0 -1 4 -1
 0 0  0 0  0 -1  0 -1  4
```

$$
\begin{array}{lccccccccc}
1:(1,1) & 4u_1 & -u_2 & & -u_4 & & & & & = 1 \\
2:(2,1) & -u_1 & +4u_2 & -u_3 & & -2u_5 & & & & = 0 \\
3:(3,1) & & -2u_2 & +4u_3 & & & -2u_6 & & & = 0 \\
4:(1,2) & -u_1 & & & +4u_4 & -2u_5 & & -u_7 & & = 0 \\
5:(2,2) & & -u_2 & & -u_4 & +4u_5 & -u_6 & & -u_8 & = 0 \\
6:(3,2) & & & -u_3 & & -2u_5 & +4u_6 & & & -u_9 = 0 \\
7:(1,3) & & & & -2u_4 & & & +4u_7 & -2u_8 & = 0 \\
8:(2,3) & & & & & -2u_5 & & -u_7 & +4u_8 & -u_9 = 0 \\
9:(3,3) & & & & & & -u_6 & & -u_8 & +4u_9 = -1
\end{array}
$$

$$
\begin{bmatrix}
+4 & -1 & & -1 & & & & & \\
-1 & +4 & -1 & & -2 & & & & \\
 & -2 & +4 & & & -2 & & & \\
-1 & & & +4 & -2 & & -1 & & \\
 & -1 & & -1 & +4 & -1 & & -1 & \\
 & & -1 & & -2 & +4 & & & -1 \\
 & & & -2 & & & +4 & -2 & \\
 & & & & -2 & & -1 & +4 & -1 \\
 & & & & & -1 & & & +4
\end{bmatrix}
\begin{Bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \\ u_8 \\ u_9 \end{Bmatrix}
=
\begin{Bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -1 \end{Bmatrix}
$$

Neumann BC

Gleichungssystem

$$4u_1 - 2u_2 - 2u_4 = 0$$

$$\begin{aligned}
-u_1 + 4u_2 - u_3 - 2u_5 &= 0 \\
-2u_2 + 4u_3 - 2u_6 &= 0 \\
-u_1 + 4u_4 - 2u_5 - u_7 &= 0 \\
-u_2 - u_4 + 4u_5 - u_6 - u_8 &= 0 \\
-u_3 - 2u_5 + 4u_6 - u_9 &= 0 \\
-2u_4 + 4u_7 - 2u_8 &= 0 \\
-2u_5 - u_7 + 4u_8 - u_9 &= 0 \\
-2u_6 - 2u_8 + 4u_9 &= 0
\end{aligned} \tag{1.59}$$

Matrix

```
 4 -2  0 -2  0 0  0 0  0
-1 4 -1 0 -2 0  0 0  0
 0 -2  4 0  0 -2  0 0  0
-1 0  0 4 -2 0 -1 0  0
 0 -1  0 -1  4 -1  0 -1  0
 0 0 -1 0 -2 4  0 0 -1
 0 0  0 -2  0 0  4 -2  0
 0 0  0 0 -2 0 -1 4 -1
 0 0  0 0  0 -2  0 -2  4
```

+DBC

```
4 0 0 0 0 0 0 0 0 b:4
0 4 -1 0 -2 0 0 0 0 b:1
0 -2 4 0 0 -2 0 0 0 b:0
0 0 -1 4 -2 0 0 0 0 b:0
0 0 0 -1 4 -1 0 0 0 b:0
0 0 0 0 -2 4 -1 0 0 b:0
0 0 0 -2 0 0 4 -2 0 b:0
0 0 0 0 -2 0 -1 4 0 b:-1
0 0 0 0 0 0 0 0 4 b:-4
```

```
ZONE T="BIG ZONE", I=3, J=3, DATAPACKING=POINT
0 0 1
1 0 0.267857
2 0 0.0714286
0 1 0.0178571
1 1 0
2 1 -0.0178571
0 2 -0.0714286
1 2 -0.267857
2 2 -1

ZONE T="BIG ZONE", I=3, J=3, DATAPACKING=POINT
```

```
0 0 1
1 0 0.28125
2 0 0.125
0 1 0.03125
1 1 0
2 1 -0.03125
0 2 -0.125
1 2 -0.28125
2 2 -1

ZONE T="BIG ZONE", I=3, J=3, DATAPACKING=POINT
0 0 1
1 0 0.25
2 0 0
0 1 0.25
1 1 -1.8955e-17
2 1 -0.25
0 2 -7.57586e-18
1 2 -0.25
2 2 -1

std::vector<int>bc_neumann_nodes;
std::vector<double>bc_neumann_values;
```

1.6.3 Vergrößerung des Testbeispiels

Nachdem wir das implizite FD Verfahren mit dem Mini-Testbeispiel(3 links, 3 rechts) entwickelt haben, probieren wir, ob es auch für größere FD Gitter funktioniert. Aber nicht gleich übermütig werden, wir nehmen erstmal ein 4×4 Problem. Was ist hierfür zu tun.

- Vergrößerung des Gitters.

```
FDM::FDM()
{
  I = 4; //43;
  J = 4; //33;
...}
```

- Anpassen der Randbedingungen.

```
void FDM::SetBoundaryConditions()
{
  //-------------------------------------
  //Neumann boundary condition
  //....................
  bc = new BC();
  bc->node_number = 1; ->2,3,4,5,6,7,8,9,10,11,12,13,14
...}
```

Das war's schon.

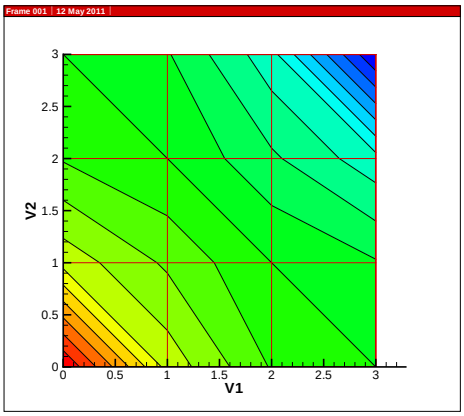

Abbildung 1.20: 4×4 Testbeispiel

Übung GW4c Der Quelltext für diese Übung befindet sich in `EXERCISES\GW4c`.

- Gleichungssystem

```
4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 b:4
0 4 -1 0 0 -2 0 0 0 0 0 0 0 0 0 0 b:1
0 -1 4 -1 0 0 -2 0 0 0 0 0 0 0 0 0 b:0
0 0 -2 4 0 0 0 -2 0 0 0 0 0 0 0 0 b:0
0 0 0 0 4 -2 0 0 -1 0 0 0 0 0 0 0 b:1
0 -1 0 0 -1 4 -1 0 0 -1 0 0 0 0 0 0 b:0
0 0 -1 0 0 -1 4 -1 0 0 -1 0 0 0 0 0 b:0
0 0 0 -1 0 0 -2 4 0 0 0 -1 0 0 0 0 b:0
0 0 0 0 -1 0 0 0 4 -2 0 0 -1 0 0 0 b:0
0 0 0 0 0 -1 0 0 -1 4 -1 0 0 -1 0 0 b:0
0 0 0 0 0 0 -1 0 0 -1 4 -1 0 0 -1 0 b:0
0 0 0 0 0 0 0 -1 0 0 -2 4 0 0 0 0 b:-1
0 0 0 0 0 0 0 0 -2 0 0 0 4 -2 0 0 b:0
0 0 0 0 0 0 0 0 0 -2 0 0 -1 4 -1 0 b:0
0 0 0 0 0 0 0 0 0 0 -2 0 0 -1 4 0 b:-1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 b:-4
```

1.6.4 Prinzipbeispiel

Jetzt gehen wir auf's Ganze und probieren unser Testbeispiel (20×10 Rechteck).

```
FDM::FDM()
{
  I = 21;
  J = 11;
...}
```

Wir setzen folgende (Dirichlet)Randbedingungen.

- Oben: konstanter Wert = -1 (North),
- Unten: konstanter Wert = 1 (South),
- Links: konstanter Wert = 0 (West),
- Rechts: konstanter Wert = 0 (East),

```
void FDM::SetBoundaryConditions()
{
  //-----------------------------------
  for(int i=0;i<I;i++)
  { //S
    bc_nodes.push_back(i); u[i] = 1;
    bc_file << "S: " << i << " value= " << u[i] << std::endl;
  }
  for(int i=0;i<I;i++)
```

```
  { //N
    l = I*(J-1)+i;
    bc_nodes.push_back(l); u[l] = -1;
    bc_file << "N: " << l << " value= " << u[l] << std::endl;
  }
  for(int j=1;j<J-1;j++)
  { // E
    l = I*j;
    bc_nodes.push_back(l); u[l] = 0;
    bc_file << "E: " << l << " value= " << u[l] << std::endl;
  }
  for(int j=1;j<J-1;j++)
  { // W
    l = I*j+I-1;
    bc_nodes.push_back(l); u[l] = 0;
    bc_file << "W: " << l << " value= " << u[l] << std::endl;
  }
}
```

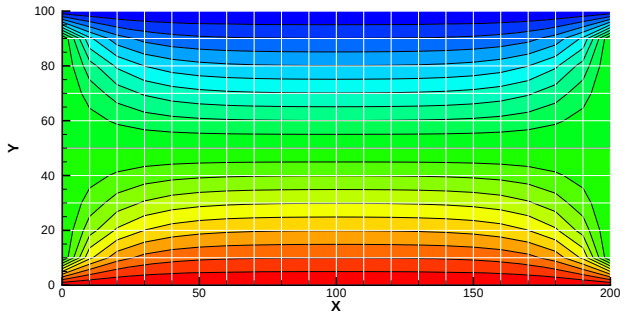

Abbildung 1.21: Prinzipbeispiel - Ergebnisse des impliziten staionären FD Verfahrens

Das Gleichungssystem sparen wir uns lieber ... (231×231 Matrix).

Übung GW4d Der Quelltext für diese Übung befindet sich in `EXERCISES\GW4d`.

Zusammengefasst nochmal unsere Beispiele:

1. Test case: hat nur ganz wenige Konten (3×3), kann per Hand nachgerechnet werden.
2. Principal case: immer noch einfache Geometrie, unterschiedliche $x - y$ Ausdehnungen, zu groß für Handrechnung (Programm muss also tun).

3. Application case: Selke im Bode Einzugsgebiet, ein Beispiel mit richtigen Daten.

1.7 Finite-Element-Methode

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. 1.22 zeigt uns ein aktuelles Beispiel aus einem Forschungsvorhaben zusammen mit der Bundesanstalt für Geowissenschaften und Rohstoffe (BGR) in Hannover

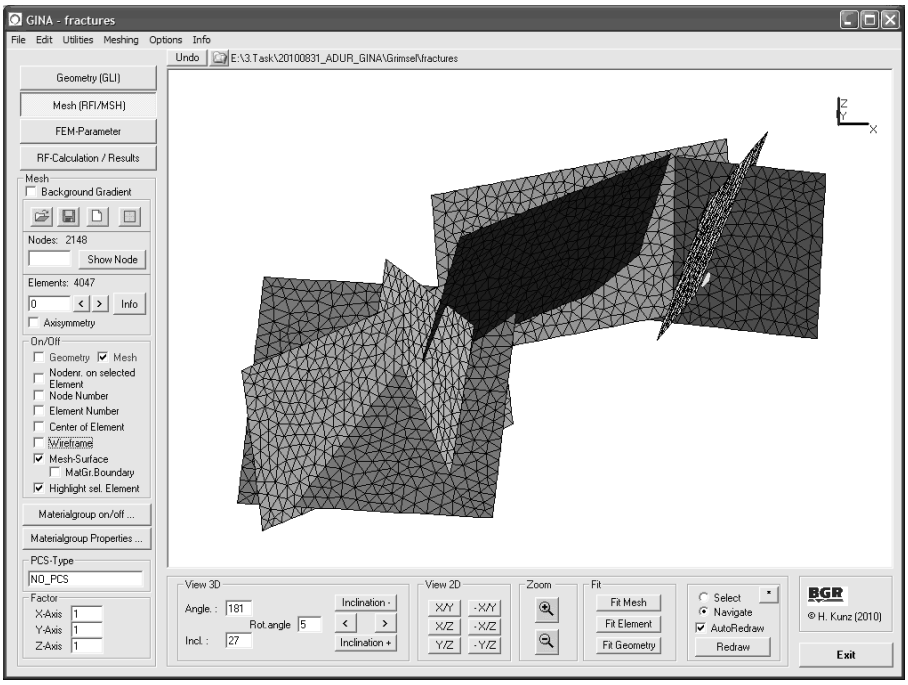

Abbildung 1.22: Modellierung eines Kluftsystems im Kristallin (Herbert Kunz, BGR)

Bei diesem Forschungsvorhaben geht es um die Modellierung von Strömungs- und Ausbreibuntsprozessen im geologischen Untergrund. Um mögliche Transportpfade im Untergrund möglichst genau beschreiben zu können, muss die Geometrie von Kluftsystemen genau beschrieben werden. Hierzu benötigt man dreiecksbasierte Elemente (Dreiecke, Tetraeder, Prismen, Pyramiden). Derartige Rechengitter können nicht mehr mit finiten Differenzen abgebildet werden,

hierfür benötigt man Finite Element oder Finite Volumen.

Zur Erläuterung des FE Methoden schauen wir uns ein ganz einfaches Beispiel von Jonathan Istok (1989) an. In den VISLab Übungen werden wir uns dann von den Möglichkeiten der FE Methode überzeugen können. So werden uns z.B. ein Grundwassermodell fast der gesamtem Arabischen Halbinsel ansehen und damit arbeiten können (ich hatte ihnen die Saudi Arabia Case Study in der Christmas Lecture 2010 schon einmal vorgestellt). Bei dem einfachen Beispiel handelt es sich um ein Säulenmodell (Abb.

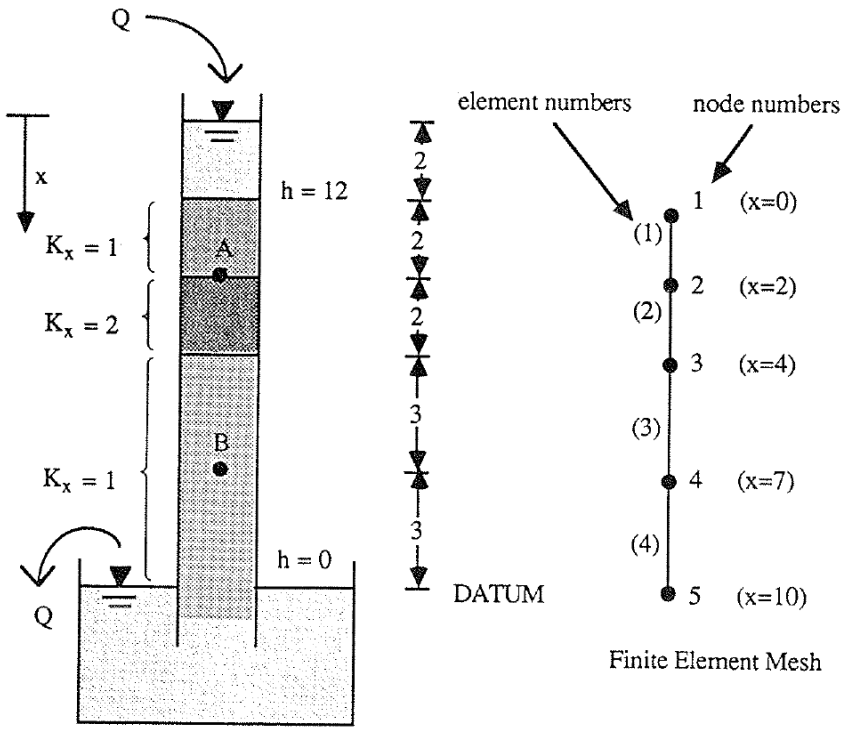

Abbildung 1.23: Bodensäulenmodell zur Erläuterung der FE Methode nach Istok (1989)

Wir betrachten ein 1D stationäres Grundwasserströmungsproblem.

$$\frac{\partial}{\partial x}\left(K_x \frac{\partial h}{\partial x}\right) = 0 \tag{1.60}$$

Ein Näherungsverfahren wird uns eine Näherungslösung $\hat{h}$ liefern, welche die Bilanzgleichung (1.60) nicht mehr ganz korrekt erfüllt.

$$\frac{\partial}{\partial x}\left(K_x \frac{\partial \hat{h}}{\partial x}\right) = R(x) \neq 0 \tag{1.61}$$

Dabei ist $R(x)$ der Fehler, das sogenannte Residuum. Das Residuum kann in den verschiedenen Gitterpunkten i, j unterschiedliche Wert $R_i \neq R_j$ annehmen. Am Knoten 3 hängen die Elemente (2) und (3) (Abb. . Daher ergibts sich das Residuum im Knoten 3 aus den Elementwerten wie folgt.

$$R_3 = R_3^{(2)} + R_3^{(3)} \tag{1.62}$$

Ohne Beschränkung der Allgemeinheit können wir für jeden Knoten i, das Residuum wie folgt schreiben.

$$R_i = \sum_{e=1}^{p} R_i^{(e)} \tag{1.63}$$

Dabei ist p die Anzahl der Elemente, die am Knoten i angebunden sind. Der Elementbeitrag zum Residuum lässt sich wie folgt berechnen.

$$R_i^{(e)} = \int_{x_i^e}^{x_j^e} N_i^{(e)} \left(K_x^{(e)} \frac{\partial^2 \hat{h}^{(e)}}{\partial x^2} \right) dx \tag{1.64}$$

Dabei ist $N_i^{(e)} \equiv W_i(x)$ eine Interpolationsfunktion auf dem Element (e) (Abb. 1.24).

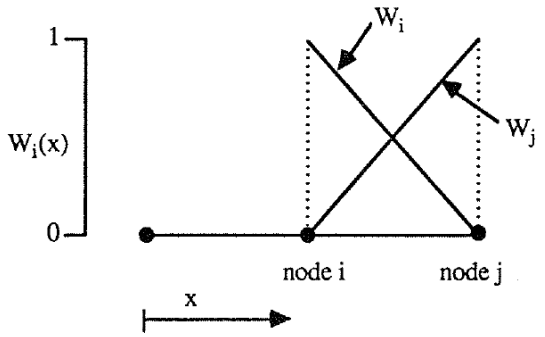

Abbildung 1.24: Interpolationsfunktion für die Galerkin-Methode nach Istok (1989)

Die gleiche Beziehung lässt sich den anderen Element-Knoten j schreiben.

$$R_j^{(e)} = \int_{x_i^{(e)}}^{x_j^{(e)}} N_j^{(e)} \left(K_x^{(e)} \frac{\partial^2 \hat{h}^{(e)}}{\partial x^2} \right) dx \tag{1.65}$$

Die linearen Interpolationsfunktionen für 1D Elemente sind

$$N_i^{(e)}(x) = \frac{x_j^{(e)} - x}{L^{(e)}} \quad , \quad N_j^{(e)}(x) = \frac{x - x_i^{(e)}}{L^{(e)}} \tag{1.66}$$

Die approximierte Feldgröße h kann nun wie folgt auf dem 1D finiten Element interpoliert werden (Abb. 1.24).

$$\begin{aligned} \hat{h}^{(e)}(x) &= N_i^{(e)} h_i + N_j^{(e)} h_j \\ &= \frac{x_j^{(e)} - x}{L^{(e)}} h_i + \frac{x - x_i^{(e)}}{L^{(e)}} h_j \end{aligned} \tag{1.67}$$

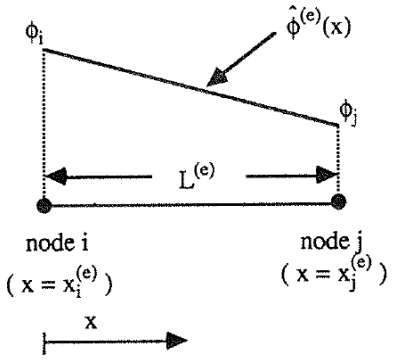

Abbildung 1.25: Interpolierte Näherungslösung auf einem 1D Element nach Istok (1989)

Jetzt stoßen wir auf ein ernsthaftes Problem. In den Gleichungen (1.64) und (1.65) müssten wir Ableitungen zweiter Ordnung berechnen, unsere Interpolationsfunktionen sind aber linear - also existieren keine zweiten Ableitung - was tun? Wir machen eine mathematischen Trick in diesen Gleichungen. Ein partielle Integration von (1.64) ergibt.

$$\int_{x_i^e}^{x_j^e} N_i^{(e)} \left(K_x^{(e)} \frac{\partial^2 \hat{h}^{(e)}}{\partial x^2} \right) dx = - \int_{x_i^e}^{x_j^e} K_x^{(e)} \frac{\partial N_i^{(e)}}{\partial x} \frac{\partial \hat{h}^{(e)}}{\partial x} dx + N_i^{(e)} K_x^{(e)} \frac{\partial \hat{h}^{(e)}}{\partial x} \Bigg|_{x_i^e}^{x_j^e} \tag{1.68}$$

- Wie können wir die Umformung in Gleichung (1.68) überprüfen?

Der zweite Term auf der rechten Seite von (1.68)

$$N_i^{(e)} K_x^{(e)} \frac{\partial \hat{h}^{(e)}}{\partial x} \Bigg|_{x_i^e}^{x_j^e} \tag{1.69}$$

entspricht der Vorgabe von Werten auf den Randknoten x_j^e und x_i^e des Elements (e). Handelt es sich um einen Randknoten, dann geht es um Randbedingungen.

- Um welchen Randbedingungstypen handelt es bei (1.69)?

- Welche Randbedingung ist es, wenn der Wert von (1.69) gleich Null ist?
- Was passiert mit inneren Knoten?

Nun setzen wir die Beziehung (1.68) in die Gleichung (1.64) ein und erhalten.

$$\begin{aligned} R_i^{(e)} &= \int_{x_i^e}^{x_j^e} N_i^{(e)} \left(K_x^{(e)} \frac{\partial^2 \hat{h}^{(e)}}{\partial x^2} \right) dx \\ &= -\int_{x_i^e}^{x_j^e} K_x^{(e)} \frac{\partial N_i^{(e)}}{\partial x} \frac{\partial \hat{h}^{(e)}}{\partial x} dx + N_i^{(e)} K_x^{(e)} \frac{\partial \hat{h}^{(e)}}{\partial x} \Bigg|_{x_i^e}^{x_j^e} \end{aligned} \tag{1.70}$$

Jetzt müssen wir uns um $\partial \hat{h}^{(e)}/\partial x$ kümmern.

$$\frac{\partial \hat{h}^{(e)}}{\partial x} = \frac{\partial}{\partial x} \left(N_i^{(e)} h_i + N_j^{(e)} h_j \right) = \frac{\partial N_i^{(e)}}{\partial x} h_i + \frac{\partial N_j^{(e)}}{\partial x} h_j \tag{1.71}$$

Nach Einsetzen der Interpolationsfunktionen erhalten wir schließlich.

$$\frac{\partial \hat{h}^{(e)}}{\partial x} = \frac{1}{L^{(e)}} (-h_i + h_j) \tag{1.72}$$

Für die Ableitungen der linearen Interpolationsfunktionen folgt.

$$\frac{\partial N_i^{(e)}}{\partial x} = \frac{\partial}{\partial x} \left(\frac{x_j^{(e)} - x}{L^{(e)}} \right) = -\frac{1}{L^{(e)}} \tag{1.73}$$

$$\frac{\partial N_j^{(e)}}{\partial x} = \frac{\partial}{\partial x} \left(\frac{x - x_i^{(e)}}{L^{(e)}} \right) = \frac{1}{L^{(e)}} \tag{1.74}$$

Setzen wir jetzt die Beziehungen in die Gleichung (1.70) ein, ergibt sich.

$$\begin{aligned} R_i^{(e)} &= \int_{x_i^e}^{x_j^e} K_x^{(e)} \left(-\frac{1}{L^{(e)}} \right) \left(\frac{1}{L^{(e)}} \right) (-h_i + h_j) \\ &= \frac{K_x^{(e)}}{L^{(e)}} (h_i - h_j) \end{aligned} \tag{1.75}$$

In gleicher Weise bekommen wir.

$$R_j^{(e)} \frac{K_x^{(e)}}{L^{(e)}} (-h_i + h_j) \tag{1.76}$$

Beide Gleichungen (1.75) und (1.76) lassen sich zusammen in einer Matrizen-Form schreiben.

$$\left\{ \begin{array}{c} R_i^{(e)} \\ R_j^{(e)} \end{array} \right\} = \frac{K_x^{(e)}}{L^{(e)}} \underbrace{\left[\begin{array}{cc} +1 & -1 \\ -1 & +1 \end{array} \right]}_{2\times 2} \left\{ \begin{array}{c} h_i \\ h_j \end{array} \right\} \tag{1.77}$$

Leitfähigkeitsmatrix

$$[K^{(e)}] = \frac{K_x^{(e)}}{L^{(e)}} \underbrace{\left[\begin{array}{cc} +1 & -1 \\ -1 & +1 \end{array} \right]}_{2\times 2} \tag{1.78}$$

Aufgrund der Elementgeomtrien $L^{(e)}$ (Abb. 1.7) ergeben sich folgende Elementmatrizen.

$$\begin{gathered} [K^{(1)}] = \left[\begin{array}{cc} +1/2 & -1/2 \\ -1/2 & +1/2 \end{array} \right] \quad , \quad [K^{(2)}] = \left[\begin{array}{cc} +1 & -1 \\ -1 & +1 \end{array} \right] \\ \mathrm{K}^{(1)} = \left[\begin{array}{cc} +1/3 & -1/3 \\ -1/3 & +1/3 \end{array} \right] \quad , \quad [K^{(2)}] = \left[\begin{array}{cc} +1/3 & -1/3 \\ -1/3 & +1/3 \end{array} \right] \end{gathered} \tag{1.79}$$

Zusammenbauen des Gleichungssystems.

$$\{\mathbf{R}\} = [\mathbf{K}]\{\mathbf{h}\} = 0 \tag{1.80}$$

$$\{\mathbf{R}\} = \left[\begin{array}{c} R_1 \\ R_2 \\ R_3 \\ R_4 \\ R_5 \end{array} \right] \quad , \quad \{\mathbf{h}\} = \left[\begin{array}{c} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \end{array} \right] \tag{1.81}$$

$$[\mathbf{K}] = [\mathbf{K}^{(1)}] + [\mathbf{K}^{(2)}] + [\mathbf{K}^{(3)}] + [\mathbf{K}^{(4)}] \tag{1.82}$$

$$\begin{aligned} [\mathbf{K}] &= \left[\begin{array}{ccccc} 1/2 & -1/2 & 0 & 0 & 0 \\ -1/2 & 1+1/2 & -1 & 0 & 0 \\ 0 & -1 & 1+1/3 & -1/3 & 0 \\ 0 & 0 & -1/3 & 1/3+1/3 & -1/3 \\ 0 & 0 & 0 & -1/3 & 1/3 \end{array} \right] \\ &= \left[\begin{array}{ccccc} 1/2 & -1/2 & 0 & 0 & 0 \\ -1/2 & 3/2 & -1 & 0 & 0 \\ 0 & -1 & 4/3 & -1/3 & 0 \\ 0 & 0 & -1/3 & 2/3 & -1/3 \\ 0 & 0 & 0 & -1/3 & 1/3 \end{array} \right] \end{aligned} \tag{1.83}$$

$$\begin{bmatrix} 1/2 & -1/2 & 0 & 0 & 0 \\ -1/2 & 3/2 & -1 & 0 & 0 \\ 0 & -1 & 4/3 & -1/3 & 0 \\ 0 & 0 & -1/3 & 2/3 & -1/3 \\ 0 & 0 & 0 & -1/3 & 1/3 \end{bmatrix} \begin{Bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{Bmatrix} \tag{1.84}$$

1.8 Implementierung des FE Verfahrens

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

```
#include <iostream>
#include "fem.h"
#include <time.h>

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

int main()
{
  clock_t start, end;
  double cpuTime;
  std::ofstream aux_file;
  aux_file.open("../cputime.txt");
  start = clock();
  //-----------------------------------------------
  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);
  }
  //-----------------------------------------------
  end = clock();
  cpuTime= (end-start)/ (double)(CLOCKS_PER_SEC);
  aux_file << "CPU time:" << cpuTime << std::endl;
  aux_file.close();
  fem->out_file.close();
  return 0;
}
```

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 habe wir die Netzgenerierung ja während der Programmausführung erledigt. Das ist ja auch nicht weiter schwer, da bei der FDM immmer regelmäßige Rechteckgitter die Ausgangsbasis sind.

Der beste Weg zur Flexibilität ist die Möglichkeit zu schaffen, FE Gitter über eine Datei einlesen zu können. Alsü 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, wir ddas Lesen beendet.

```
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;
    ...
  }
}
```

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
```

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.

```
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);
      }
    }
  }
}
```

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;
}
```

1.8.1 Anfangsbedingungen

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

1.8.2 Randbedingungen

In der Aufgabenbeschreibung (Abb. 1.7) 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;
}
```

1.8.3 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);
  }
}
```

`CalculateElementMatrix(int)` berechnet die Element-Leitfähigkeitsmatrix gemäß Gleichung (1.78). 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`.

```
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);
}
```

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;
    }
  }
}
```

Elementmatrizen

```
-------------------------
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
```

1.8.4 Gleichungssystem

Nun müssen die Elementmatrizen zum Gleichungssystem zusammengefügt werden gemäß Gleichung (1.84). 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];
  }
}
```

```
 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
```

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()
```

```
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
```

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

```
12
9.33333
8
4
0
```

Übung GW5 Der Quelltext für diese Übung befindet sich in `EXERCISES\GW5_FEM`.

Kapitel 2

Oberflächenydrologie

Kapitel 3

Bodenhydrologie

Kapitel 4

Transportprozesse in porösen Medien

Übungen

Fallbeispiele

Kapitel 5

Fallbeispiele

Literaturverzeichnis

[1] K.-F. Busch, L. Luckner, and K. Tiemer. *Geohydraulik*. Borntraeger, Berlin-Stuttgart, 1993.

[2] D. Gross, W. Hauger, and P. Wriggers. *Formeln und Aufgaben zur Technischen Mechanik*. Springer-Lehrbuch, 2009.

[3] Delfs J-O, Kalbus E, Park C-H, and Kolditz O. Ein physikalisch basiertes Modellkonzept zur Transportmodellierung in gekoppelten Hydrosystemen. *Grundwasser*, 14(3):219–230, 2009.

[4] W. Kinzelbach. *Numerische Methoden zur Modellierung des Transports von Schadstoffen im Grundwasser*. Oldenburg Verlag, München, 1992.

[5] H. Martin and R. Pohl. *Technische Hydromechanik*. Verlag Bauwesen, 2000.

[6] Kolditz O. *Computational methods in environmental fluid mechanics*. Springer, Berlin-Heidelberg, 2002.

[7] H. Shao, S.V. Dmytrieva, O. Kolditz, D.A. Kulik, W. Pfingsten, and G. Kosakowski. Modeling reactive transport in non-ideal aqueous-solid solution system. *Applied Geochemistry*, 24:1287–1300, 2009.

[8] J. Stribny. *Ohne Panik - Strömungsmechanik*. Oldenburg Verlag, München, 2002.

[9] Wu Y, Mathias Toll, Wenqing Wang, Thomas Kalbacher, Martin Sauter, and Olaf Kolditz. Development of a groundwater flow model in the wadi kafrein area. *Environmental Earth Science*, submitted, bei mir erhältlich.

Inhaltsverzeichnis

Vorlesung - Grundlagen **8**

1 Grundwasserhydraulik **9**

1.1 Grundwassergleichung 9

1.2 Bilanzierung 10

1.2.1 Definition des Testbeispiels 11

1.2.2 Rechenverfahren zur Bilanzierung 11

1.2.3 Programmtechnische Umsetzung 15

1.2.4 Gauss-Seidel Verfahren 17

1.3 2D Finite-Differenzen-Verfahren 18

1.3.1 Wiederholung Grundlagen 18

1.3.2 FDM Schema 19

1.3.3 Programmierung I - QAD 20

1.4 Selke 2D FDM Modell 28

1.5 FDM OOP 32

1.6 2D implizites Finite-Differenzen-Verfahren 34

1.6.1 Stationäres Problem 41

1.6.2 Neumann-Randbedingungen 43

1.6.3 Vergrößerung des Testbeispiels 47

1.6.4 Prinzipbeispiel 48

1.7 Finite-Element-Methode 51

1.8 Implementierung des FE Verfahrens 58

1.8.1 Anfangsbedingungen 61

1.8.2 Randbedingungen 61

1.8.3 Elementmatrizen 61
1.8.4 Gleichungssystem 63

2 Oberflächenydrologie **65**

3 Bodenhydrologie **66**

4 Transportprozesse in porösen Medien **67**

Übungen - Fallbeispiele **68**

5 Fallbeispiele **69**