Numerische Simulation einer Geothermieanlage

Dominic Kempf

14. April 2011

Inhaltsverzeichnis

1 Einführung

Die vorliegende Arbeit dokumentiert mein Softwarepraktikum für Fortgeschrittene in der AG Wissenschaftliches Rechnen der Universität Heidelberg.

Ziel der Arbeit ist es mit Hilfe des Numerik-Frameworks **DUNE** den Stoff- und Wärmetransport in einer Geothermieanlage zu simulieren.

Unter dem Begriff Geothermie sind verschiedene Verfahren der Energiegewinnung zusammengefasst. Ich befasse mich ausschließlich mit hydrothermalen Verfahren. Diese haben gemein, dass eine Bohrung in einen in etwa 3-4 km Tiefe liegenden Grundwasserleiter (Aquifer) durchgeführt wird und durch Zufuhr und Entnahme von Kalt- bzw. Heißwasser Energie gewonnen wird. Hydrothermale Geothermieanlagen werden in Gebieten mit erhöhtem Erdwärmegradienten (z.B. durch vulkanische Aktivität) gebaut, da dort die Temperatur im Aquifer entsprechend hoch ist (ca. 140 $^\circ$ C).

Dabei werden im Folgenden zwei verschiedene Typen von Anlagen betrachtet, deren Funktionsprinzip an dieser Stelle kurz zusammengefasst sei:

Einlochanlage: Es gibt ein Bohrloch, das durch eine Ummantelung gleichzeitig Zu- und Abfluss ist. Der Zufluss befindet sich unterhalb der Entnahmestelle, der Bereich des Bohrlochs dazwischen ist isoliert. Durch die Rotationssymmetrie des entstehenden Problems, lässt sich dieses auch 2-dimensional betrachten.

Zweilochanlage: Es gibt 2 Bohrlöcher im Abstand von ca. 2 km (Abstand im Aquifer - an der Erdoberfläche führen sie aus praktischen Gründen zusammen). Durch ein Loch wird hineingepumpt, durch das andere entnommen.

Die numerische Simulation kann eingesetzt werden, um qualitative Aussagen über die Leistung einer solchen Anlage bei z.B. bestimmten geologischen Begebenheiten oder bei Betrieb über mehrere Jahrzehnte zu erhalten.

2 Mathematisches Modell

Die Gleichungen zur Beschreibung des Stoff- und Wärmetransports in der Anlage wurden mir im Vorfeld meines Praktikums zur Verfügung gestellt. Ich gebe diese hier nur kurz wieder.

2.1 Stoff- und Wärmetransport

Sei $\Omega \subset \mathbb{R}^d$ das Rechengebiet und $\Sigma = [a, b]$ das betrachtete Zeitintervall. Die partiellen Differentialgleichungen zur Beschreibung des Stoff- und Wärmetransports sind

$$\nabla \cdot u = f \qquad\qquad u = -\frac{K}{\mu}(\nabla p - \rho_w(T(x,t))G)$$

$$\frac{\partial(c_e \rho_e T)}{\partial t} + \nabla \cdot q + g^- T = g^+ \qquad\qquad q = c_w \rho_w u T - \lambda \nabla T$$

Als Randbedingungen auf den Dirichlet-Rändern Γ_D^F und Γ_D^H und den Neumann-Rändern Γ_N^F und Γ_N^H sind gegeben:

- $p = \phi$ auf Γ_D^F
- $u \cdot n = U$ auf Γ_N^F
- $T = \psi$ auf Γ_D^H
- $q \cdot n = Q$ auf Γ_N^H

Zudem sind im Gebiet Anfangsbedingungen $T(x, a) = T_a(x)$ für die Temperatur zu setzen.

Diese PDEs sind, zum einen über den konvektiven Anteil des Wärmeflusses, zum anderen über die Temperaturabhängigkeit der Dichte gekoppelt. Ich werde im Rahmen dieses Praktikums allerdings nur entkoppelt rechnen.

2.2 Wahl der Randbedingungen

Es gibt verschiedene Kombinationen von Dirichlet- und Neumannrändern, die das Problem sinnvoll beschreiben. Ich habe mich für folgende Variante entschieden:

An den seitlichen Rändern des Gebiets sollen Dirichlet-Randbedingunen für Druck und Temperatur anliegen. Dies entspricht der Annahme, dass das Gebiet so groß ist, dass die Auswirkungen der Geschehnisse im Inneren auf den Rändern vernachlässigbar sind.

Die konkreten Funktionen für diese Dirichleträndern erhält man über die durch den Erdwäremegradienten verursachte Temperaturverteilung und die hydrostatische Druckverteilung. Siehe hierzu Kapitel 5.

Am oberen und unteren Rand des Gebiets sollen jeweils Neumann-0-Randbedingunen vorliegen.

Das Ein- und Auspumpen von Wasser ist nicht über Randbedinungen, sondern über Quell- und Senkterme realisiert. Klassen, zur Abfrage welchem Typ Randbedingung eine Koordinate entspricht, werden in Kapitel 4.4 vorgestellt.

2.3 Bestimmung der Entzugsleistung

Die momentane Entzugsleistung $W(t)$ ergibt sich direkt aus der Differenz zwischen Vor- und Rücklauftemperatur. Betrachte hierzu die W^+ und W^-, also zu- und abgeführte Energie pro Zeit:

$$\begin{aligned} W^+(t) &= c_w \rho_w(T_R) V^+ T^+ \\ W^-(t) &= c_w \rho_w(T_V) V^- T^- \end{aligned}$$

Dabei bezeichnen V^+ und V^- Volumenströme; T_R und T_V bezeihcnen Vor- bzw. Rücklauftemperatur. V^+ ist dabei als Eigenschaft der Anlage bekannt. V^- muss durch folgende Überlegung gewonnen werden. Auch wenn Quel- und Senkterme in Volumen angegeben sind muss die Masse des ein- und ausgepumpten Wassers gleich sind. Also muss gelten:

$$\rho_w(T_R) V^+ = \rho_w(T_V) V^- \Rightarrow V^- = \frac{\rho_w(T_R)}{\rho_w(T_V)} \cdot V^+$$

Es gilt also:

$$\begin{aligned} W(t) &= W^-(t) - W^+(t) \\ &= c_w \rho_w(T_R) V^+ \left(T_V - T_R\right) \end{aligned}$$

3 Numerisches Schema

Die Modellgleichungen für Strömung und Wärmetransport sind wie oben beschrieben folgende:

$$\begin{aligned} \nabla \cdot u &= f & u &= -\frac{K}{\mu}(\nabla p - \rho_w(T(x,t))G) \\ \frac{\partial(c_e \rho_e T)}{\partial t} + \nabla \cdot q + g^- T &= g^+ & q &= c_w \rho_w u T - \lambda \nabla T \end{aligned}$$

Die Gleichungen werden nun mit einem Finite-Volumen-Verfahren mit Upwind auf einem achsenparallelen Hexaedergitter diskretisiert. Im Folgenden wird folgende Notation verwendet werden:

- E_h^0 - Menge der Zellen
- E_h^1 - Menge der Kodimension-1-Entitäten im Inneren von Ω
- B_h^1 - Menge der Kodimension-1-Entitäten auf $\partial\Omega$
- x_e - Mittelpunkt der Zelle $\Omega_e, e \in E_h^0$
- x_f - Mittelpunkt der Kodimension-1-Entität $f \in E_h^1 \cup B_h^1$

- x_f^- - Mittelpunkt der Zelle im Inneren der Kodimension-1-Entität $f \in E_h^1 \cup B_h^1$
- x_f^+ - Mittelpunkt der Zelle im Äußeren der Kodimension-1-Entität $f \in E_h^1$
- $|\Omega_e|$ - Volumen der Entität Ω_e

Es sollen nun die notwendigen Residuen für die Finite-Volumen-Diskretisierung der beiden Gleichungen hergeleitet werden.

3.1 Strömungsgleichung

Wir starten mit der variationellen Formulierung der Strömungsgleichung:

$$\int_\Omega \left(-\frac{K}{\mu}\Delta p + \frac{K}{\mu}\nabla \cdot \rho_w(T)G) - f \right) v dx = 0$$

Unterteile Ω in Zellen und integriere partiell:

$$\sum_{e \in E_h^0} \int_{\Omega_e} -fv dx + \frac{K}{\mu} \sum_{f \in E_h^1 \cup B_h^1} \int_{\Omega_f} (-\nabla p \cdot n + \rho_w(T)G \cdot n) v ds = 0$$

Anwenden der Mittelpunktsregel und ersetzen des Gradienten durch einen Differenzenquotienten ergibt:

$$\begin{aligned} r_h(p_h, v) = &- \sum_{e \in E_h^0} f(x_e)|\Omega_e| v(x_e) \\ &- \sum_{f \in E_h^1} \frac{K_{avg}}{\mu_{avg}} \left(\frac{p(x_f^+) - p(x_f^-)}{\|x_f^+ - x_f^-\|} - \rho_w(T_{avg})G \cdot n_f \right) |\Omega_f| (v(x_f^-) - v(x_f^+)) \\ &- \sum_{f \in B_h^1; \Omega_f \subseteq \Gamma_D^F} \frac{K_{avg}}{\mu_{avg}} \left(\frac{\phi(x_f) - p(x_f^-)}{\|x_f - x_f^-\|} - \rho_w(T(x_f^-))G \cdot n_f \right) |\Omega_f| v(x_f^-) \\ &+ \sum_{f \in B_h^1; \Omega_f \subseteq \Gamma_N^F} U(x_f)|\Omega_f| v(x_f^-) \end{aligned}$$

Hierbei wurde $T_{avg} = \frac{T(x_f^+) + T(x_f^-)}{2}$ gesetzt. Dichte und dynamische Viskosität wurden basiernd auf diesem Temperaturmittelwert ausgewertet. Für die Permeabilität wurde das harmonische Mittel verwendet:

$$K_{avg} = \frac{2}{\frac{1}{K(x_f^+)} + \frac{1}{K(x_f^-)}}$$

Diese Mittelung ist notwendig, da die Permeabilität zwischen den verschiedenen Bereichen Sprünge mehrerer Größenordnungen macht, wir sie aber auf den Grenzflächen auswerten müssen. Eine arithmetische Mittelung würde hier keine sinnvollen Ergebnisse liefern.

3.2 Transportgleichung

Die Variationsformulierung der Transportgleichung lautet

$$\frac{d}{dt} \underbrace{\int_\Omega c_e \rho_e T v dx}_{\Rightarrow m_h(T_h, v)} + \underbrace{\int_\Omega (\nabla \cdot (c_w \rho_w u T - \lambda \nabla T) + g^- T - g^+) v dx}_{\Rightarrow s_h(T_h, v)} = 0$$

Das Residuum $m_h(T_h, v)$ ergibt sich leicht:

$$m_h(T_h, v) = \sum_{e \in E_h^0} c_e \rho_e T(x_e) |\Omega_e| v(x_e)$$

Für den Rest betrachten wir wiederum die Variationsformulierung

$$\int_\Omega \left(\nabla \cdot (c_w \rho_w u T - \lambda \nabla T) + g^- T - g^+ \right) v dx = 0$$

und wenden partielle Integration an und teilen das Gebiet in Zellen ein

$$\sum_{e\in E_h^0}\int_{\Omega_e}(g^-T-g^+)vdx+\sum_{f\in E_h^1\cup B_h^1}\int_{\Omega_f}(c_w\rho_w uT\cdot n-\lambda\nabla T\cdot n)vds=0$$

Anwenden von Mittelpunktsregel und Differenzenquotienten ergibt dann:

$$\begin{aligned}
s_h(T_h,v)=&\sum_{e\in E_h^0}(g^-(x_e)T_h(x_e)-g^+(x_e))|\Omega_e|v(x_e)\\
&+\sum_{f\in E_h^1}\left(-\lambda\frac{T_h(x_f^+)-T_h(x_f^-)}{\|x_f^+-x_f^-\|}+c_w\rho_w u_h(x_f)T_{upw}(x_f)\cdot n_f\right)|\Omega_f|(v(x_f^-)-v(x_f^+))\\
&+\sum_{f\in B_h^1;\Omega_f\subseteq\Gamma_D^H}\left(-\lambda\frac{\psi(x_f)-T_h(x_f^-)}{\|x_f-x_f^-\|}+c_w\rho_w u_h(x_f)T_{upw}(x_f^-)\cdot n_f\right)|\Omega_f|v(x_f^-)\\
&+\sum_{f\in B_h^1;\Omega_f\subseteq\Gamma_N^H}Q(x_f)|\Omega_f|v(x_f^-)
\end{aligned}$$

u_h muss hierbei aus p_h rekonstruiert werden. Dies erfolgt durch:

$$\begin{aligned}
u_h\cdot n_f&=-\frac{K}{\mu}\left((\nabla p-\rho_w(T)G)\cdot n_f\right.\\
&=-\frac{K}{\mu}\left(\frac{p(x_f^+)-p(x_f^-)}{\|x_f^+-x_f^-\|}-\rho_w(T)G\cdot n_f\right)
\end{aligned}$$

K,μ,T müssen dabei wieder durch geeignete Mittelung errechnet werden. Besonderer Behandlung bedarf hier der Fall von Dirichlet-Rändern, da eine Rekonstruktion von u_h hier in Ermangelung einer äußeren Zelle kein Wert bei x_f^+ zur Verfügung steht. Dies lässt sich durch Abfrage der Art von Randbedingung für den Druck umgehen:

$$u_h\cdot n_f=\begin{cases}-\frac{K}{\mu}\left(\frac{\phi(x_f)-p(x_f^-)}{\|x_f-x_f^-\|}-\rho_w(T)G\cdot n_f\right) & \Omega_f\subseteq\Gamma_D^F\\ U(x_f)\cdot n_f & \Omega_f\subseteq\Gamma_N^F\end{cases}$$

Abhängig von der Flussrichtung wird dann T_{upw} gewählt:

$$T_{upw}=\begin{cases}T_h(x_f^-) & u_h\cdot n_f\geq 0\\ T_h(x_f^+) & u_h\cdot n_f<0\end{cases}$$

Auf dem Rand wird anstatt $T_h(x_f^+)$ die entsprechende Dirichlet-Randbedingung verwendet.

3.3 Implementierung

Die Implementierung der lokalen Operatoren folgt 1:1 den oben hergeleiteten Residuen und befindet sich in den Dateien

- `./src/localop_decoupled_flow.hh`
- `./src/localop_decoupled_transport.hh`
- `./src/localop_time.hh`

4 Gittergenerierung

Ziel dieses Abschnitts ist es, die zur Diskretisierung der Modellgleichungen verwendeten Gitter und ihre Implementierung in **DUNE** vorzustellen. Als zu Grunde liegender Gittermanager wird hierbei `UG` verwendet.

4.1 Anforderungen des Modells an das Gitter

Wie bereits erwähnt, lösen wir die Modellgleichungen auf einem achsparallelen, konformen Hexaedergitter. Dies ist notwendig damit sich der Gradient $\nabla p \cdot n = \frac{\partial p}{\partial n}$ als einfacher Differenzenquotient $\frac{p(x_f^+) - p(x_f^-)}{\|x_f^+ - x_f^-\|}$ darstellen lässt (analog auch für $\nabla T \cdot n$).

Die Größe des Rechengebiets muss so gewählt sein, dass auf den Rändern des Gebiets die Auswirkungen der Vorgänge im Inneren vernachlässigbar sind. Ich habe

$$[-3000, 3000] \times [-2000, 2000] \times [-3900, -3000]$$

gewählt.

Das Bohrloch hat mit 28 cm Durchmesser eine verschwindende Größe im Vergleich zum Rest des Gebiets. Daher muss das Bohrloch selbst und der Bereich darum herum vom Gitter sehr gut aufgelöst werden. Dies führt - da man durch die verwendeten Löser eine Obergrenze für die Gesamtzahl der Zellen hat - zu stark anisotropen Gittern.

4.2 Implementierung der Gittergenerierung

Der Einfachheit halber werden die benötigten Gitter direkt im Code erzeugt. Dies ist durch Verwendung der `Dune::GridFactory` sehr einfach möglich. Informationen über die Lage der Knoten werden in Vektoren (einer pro Raumdimension) übergeben. Um das Füllen dieser Vektoren möglichst einfach zu gestalten, habe ich die Klasse `GridVector` geschrieben, welche durch öffentliche Vererbung aus `std::vector` gewonnen wird und Methoden bereitstellt, die genau den typischen Anforderungen der Gittergenerierung entsprechen und - bei ausschließlicher Verwendung - gleichzeitg sicherstellen, dass der Inhalt die benötigte Monotonie hat. Diese Methoden unterteilen sich in folgende Gruppen:

- äquidistante Methoden: $h_{i+1} = h_i$
- lineare Methoden: $h_{i+1} = h_i + \Delta h$
- geometrische Methoden: $h_{i+1} = q \cdot h_i$

Die zur Verfügung stehenden Methoden werden im Folgenden kurz erläutert.
Die äquidistanten Methoden:

- `void equidistant_n_h(int n, ctype h)` : fügt n Intervalle der Länge h an.
- `void equidistant_n_xend(int n, ctype xend)` : füllt den Bereich bis x_{end} mit n Intervallen. Dabei gilt $h = \frac{x_{end} - x_{start}}{n}$.
- `void equidistant_h_xend(ctype h, ctype xend)` : fügt solange Intervalle der Länge h an bis x_{end} überschritten ist. **Achtung:** Der letzte Wert muss nicht x_{end} entsprechen!

Die linearen Methoden:

- `void linear_n_h0_dh(int n, ctype h0, ctype dh)` : fügt n Intervalle an, wobei $h_{i+1} = h_i + \Delta h$
- `void linear_h0_dh_xend(ctype h0, ctype dh, ctype xend)` fügt solange Intervalle linear ansteigender Länge (Rate: Δh) an bis x_{end} überschritten ist. **Achtung:** Der letzte Wert muss nicht x_{end} entsprechen!
- `void linear_n_h0_xend(int n, ctype h0, ctype xend)` : fügt n Intervalle an, so dass der letzte eingefügte Knoten x_{end} ist. Dabei wird Δh durch folgende Gleichung gewonnen:

$$\sum_{i=0}^{n-1} (h_0 + i\Delta h) = x_{end} - x_{start} \;\Leftrightarrow\; \Delta h = \frac{2(x_{end} - x_{start} - nh_0)}{n(n-1)} \tag{1}$$

- `void linear_h0_hend_xend(ctype h0, ctype hend, ctype xend)` : füllt den Bereich bis x_{end} mit Intervallen linear ansteigender Länge, so dass die erste Intervalllänge h_0 und die letzte h_{end} entspricht. Die offensichtlichen Gleichungen hierzu sind:

$$\sum_{i=0}^{n-1} (h_0 + i\Delta h) = x_{end} - x_{start} \;\; und \;\; h_{end} = h_0 + (n-1)\Delta h \tag{2}$$

Die Lösung dieses Gleichungssystems (mit Unbekannten n und Δh) unter der Annahme, dass $n \in \mathbb{R}$ ist, lautet:

$$n = 2\frac{x_{end} - x_{start}}{h_{end} + h_0} \quad und \quad \Delta h = \frac{h_0^2 - h_{end}^2}{h_0 + h_{end} - 2(x_{end} - x_{start})} \tag{3}$$

Durch Abrunden der Lösung für n und Aufrufen der Methode `linear_n_h0_xend` erhalten wir nun eine Lösung unseres Ausgangsproblems. **Achtung:** Diese Methode ist sehr experimentell und kann unter Umständen signifikante Fehler erzeugen. Nicht ungeprüft verwenden.

Die geometrischen Methoden:

- `void geometric_n_h0_q(int n, ctype h0, ctype q)` : fügt n Intervalle an, wobei $h_{i+1} = qh_i$
- `void geometric_h0_q_xend(ctype h0, ctype q, ctype xend)` : fügt solange Intervalle an $(h_{i+1} = qh_i)$ bis x_{end} überschritten ist. **Achtung:** Der letzte Wert muss nicht x_{end} entsprechen!
- `void geometric_n_h0_xend(int n, ctype h0, ctype xend)` : füllt den Bereich bis x_{end} mit n Intervallen $(h_{i+1} = qh_i)$. q ist hierbei die zu berechnende Größe:

$$\sum_{i=0}^{n-1} h_0 q^i = x_{end} - x_{start} \quad \Leftrightarrow \quad f(q) := -q^n + \frac{x_{end} - x_{start}}{h_0} q - \frac{x_{end} - x_{start}}{h_0} + 1 \stackrel{!}{=} 0 \tag{4}$$

  Betrachtet man die Eigenschaften von $f(q)$, so sieht man, dass f auf jeden Fall $n_0 = 1$ als Nullstelle hat. Außerdem gibt es immer eine von n_0 verschiedene positive Nullstelle (sogar eine doppelte wenn n gerade). Dies ist die für uns relevante Lösung. Im Falle ungeraden n's existiert zu dem eine negative Nullstelle. q lässt sich per Newtonverfahren $q_{i+1} = q_i - \frac{f(q_i)}{f'(q_i)}$ lösen. Als q_0 bietet sich $x_{end} - x_{start}$ an, da es auf jeden Fall größer als das gesuchte q ist. Wenn die gefundene Nullstelle n_0 ist, starte das Verfahren neu mit $q_0 = 0$.

- `void geometric_n_hend_xend(int n, ctype hend, ctype xend)` : analog ist auch die Vorgabe der letzten Schrittweite möglich.
- `void geometric_n_q_xend(int n, ctype q, ctype xend)` : fügt n Intervalle mit $h_{i+1} = qh_i$ an, bis zum Wert x_{end}. Die Anfangsschrittweite wird dabei folgendermaßen berechnet:

$$\sum_{i=0}^{n-1} h_0 q^i = x_{end} - x_{start} \;\Leftrightarrow\; h_0 = \frac{(x_{end} - x_{start})(1-q)}{1 - q^n} \tag{5}$$

- `void geometric_h0_hend_xend(ctype h0, ctype hend, ctype xend)` : füllt den Bereich bis x_{end} mit Intervallen, so dass Anfangs- und Endschrittweite h_0 bzw. h_{end} entsprechen und $h_{i+1} = qh_i$ gilt. Das Gleichungssystem (mit n und q unbekannt) für diesen Vorgang lautet:

$$\sum_{i=0}^{n-1} h_0 q^i = x_{end} - x_{start} \quad und \quad h_{end} = h_0 q^{n-1} \tag{6}$$

  Löst man dieses wiederum für $n \in \mathbb{R}$ erhält man:

$$n = \frac{log(\frac{h_{end}}{h_0})}{log(q)} + 1 \quad und \quad q = \frac{x_{end} - x_{start} - h_0}{x_{end} - x_{start} - h_{end}} \tag{7}$$

  Analog zum obigen linearen Verfahren runden wir nun auf $n \in \mathbb{N}$ und korrigieren das gefundene q durch Aufruf der Methode `geometric_n_h0_xend` ! **Achtung:** Diese Methode ist sehr experimentell und kann unter Umständen signifikante Fehler erzeugen. Nicht ungeprüft verwenden.

Zudem besitzt die Klasse eine Methode `void start(ctype x)` die einen Startwert für den Vektor festsetzt. Wird diese Methode nicht als erstes aufgerufen, wird automatisch $x = 0$ gesetzt. Ein späterer Aufruf ist wirkungslos.

Zudem gibt es eine Methode `void setEntityChange()`, deren Bedeutung im Abschnitt 4.4 erklärt wird.

Die Implementierung der Klasse `GridVector` ist in der Datei `./src/gridvector.hh` zu finden.

Wie bereits erwähnt, wird nach Befüllen der Vektoren mit den nötigen Informationen das Tensorproduktgitter mittels der `Dune::GridFactory` erzeugt. Dies erfolgt durch die Factory-Klasse `GeoGridFactory<int dim>` (implementiert sind - wie in allen anderen Fällen auch - 2D und 3D), welche im Konstruktor die entsprechende Anzahl von Gittervektoren übergeben bekommt. Die Implementierung befindet sich in der Datei `./src/geogridfactory.hh`.

4.3 Standardgitter

Die Klasse `GeoGridFactory` hält neben den bisher beschriebenen weitere Konstruktoren bereit, um typische Gitter zu erzeugen. Dieser bekommt als Parameter einen Integer-Wert mit der Anzahl der Löcher in der Anlage und einen Integer-Wert, der die Feinheit des Gitters spezifiziert. Die Verwendung eines Grobgitters mit anschließender uniformer Verfeinerung ist unpraktikabel, da die gewünschten typischen Gitterweiten (z.B. geometrische Folgen) unter uniformer Verfeinerung in 'pseudoäquidistante' Gitter übergehen. Aufgrund einer Besonderheit des Gittermanagers `UG` (die Erzeugung eines Gitters mit mehr als 100000 Knoten wird sehr langsam, es sei denn man modifiziert den `UG`-Code geeignet[1]) bin ich allerdings lange davon ausgegangen, dass ich dies trotzdem so machen muss. Nachdem diese Hürde weggefallen ist, bin ich dazu übergegangen das Gitter direkt zu bauen und auf jedem Abschnitt eine linear von der angegebenen Gitterfeinheit abhängende Anzahl von Zellen einzufügen. Bei dieser Form der Implementierung ist die Klasse `GridVector` in ihrer Allgemeinheit gar nicht mehr notwendig, ich greife aber trotzdem noch auf sie zurück.

4.4 Physikalische Entitäten

Das Rechengebiet teilt sich auf in verschiedene physikalische Bereiche, wie z.B. Aquifer, Bohrloch etc. In der Implementierung der Parameterklasse muss dementsprechend häufig abgefragt werden welcher solcher physikalsichen Entität eine bestimmte Koordinate zuzuordnen ist. Da diese Abfrage leicht in eine endlose Verschachtelung von `if`-Anweisungen ausartet, sie aber doch stets den gleichen Mustern folgt, liegt eine Parametrisierung dieser Abfrage nahe. Ich habe versucht eine möglichst bequeme und generische Herangehensweise an dieses Problem zu finden.
Die physikalischen Entitäten werden wie folgt mit Werten vom Typ `integer` codiert:

0	Aquifer
1	Schicht oberhalb des Aquifers
2	isolierende Schicht unterhalb des Aquifers
3	Bohrloch
4	Quelle
5	Senke
6	Isolation zwischen Quelle und Senke (nur bei 1-Loch-Anlage)

Der Aufbau des Rechengebiets ist dabei immer gleich. Daher fußt meine Idee darauf, bereits bei Generierung der Gittervektoren, die notwendigen Informationen zu gewinnen. Hierzu ist die Methode `setEntityChange()` der Klasse `GridVector` gedacht: Wird sie aufgerufen, so merkt der Gittervektor sich den entsprechende Eintrag. Später kann dann eine `Selector`-Klasse erzeugt werden, die alle Gebeitsinformationen kennt und der Parameterklasse übergeben werden kann. Klarer Vorteil ist hierbei die leichte Handhabung von Ein- oder Zweilochanlagen - es müssen nur entsprechende `Selector`-Klassen implementiert sein.

Wichtig ist allerdings, dass der Benutzer die richtige Anzahl von Entitätswechseln in seinen Gittervektoren bereitstellt. Dies sind:

	2D	3D
Einlochanlage	x:1 y:6	x:1 y:1 z:6
Zweilochanlage	x:4 y:4	x:4 y:2 z:4

Die folgenden Graphiken sollten verdeutlichen, wie dies gemeint ist und wie die Entitätswechsel gewählt werden müssen:
Die Implementierungen der `Selector`-Klassen

[1] Hochsetzen des Wertes für `NDELEM_BLKS_MAX` in `my-ug-directory/gm/gm.h`

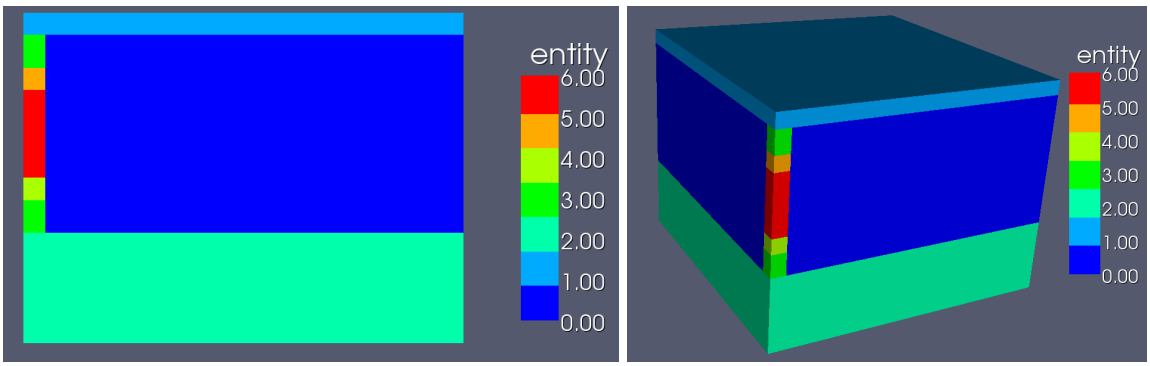


Abbildung 1: Standardentitätsverteilung einer Einlochanlage in 2D bzw. 3D

- `OneHoleSelector<int dim>`
- `TwoHoleSelector<int dim>`

befindet sich in `./src/physicalentityselector.hh`

Die Abfrage erfolgt über die Methode `int entity(Dune::FieldVector<dim> x)`

Die Klassen stellen aber auch noch weitere nützliche Methoden zur Verfügung. So kann z.B. über die Vektoren `min` und `max` direkt auf die Abmessungen des Gitters in allen Raumdimensionen zugegriffen werden.

Ausserdem liefern die Methoden

- `int boundary_pressure(Dune::FieldVector<dim> x)`
- `int boundary_temperature(Dune::FieldVector<dim> x)`

den Randbedingungstyp an den entsprechenden Koordinaten. Dies entspricht der Wahl von Randbedingungen, die ich in Kapitel 2.2 beschrieben habe.

Der Typ der Selectorklasse wird dem Parameterinterface über einen Template-Parameter mitgeteilt, eine Instanz muss im Konstruktor übergeben werden.

5 Parameter für die Modellgleichungen

Ziel dieses Abschnitts ist es die in den Modellgleichungen auftretenden Parameter und Parameterfunktionen durch einen Satz von Parametern zu ersetzen, der in naheliegender Art und Weise eine Geothermieanlage beschreibt. Dazu werden 3 Kategorien von Parametern unterschieden:

- Physikalische Eigenschaften von Wasser
- Geologische Eigenschaften des Gebiets
- Eigenschaften der verwendeten Anlage

Die Implementierung kapselt diese Kategorien in folgender Art und Weise: Das in `./src/parameterinterface.hh` implementierte Parameterinterface muss nicht variiert werden. Hier werden alle Parameterfunktionen der Modellgleichungen auf die Parameter der Geothermieanlage zurückgeführt. Für die oben genannten Kategorien von Parametern stehen in den Verzeichnissen `./src/parameter`,`./src/geology` und `./src/engine` eine Reihe von Parameterklassen zur Verfügung. Das Parameterinterface wird durch Mehrfach-Vererbung aus diesen erzeugt. Das ermöglicht ein einfaches Baukastenprinzip beim sukzessiven Erweitern der Simulation.

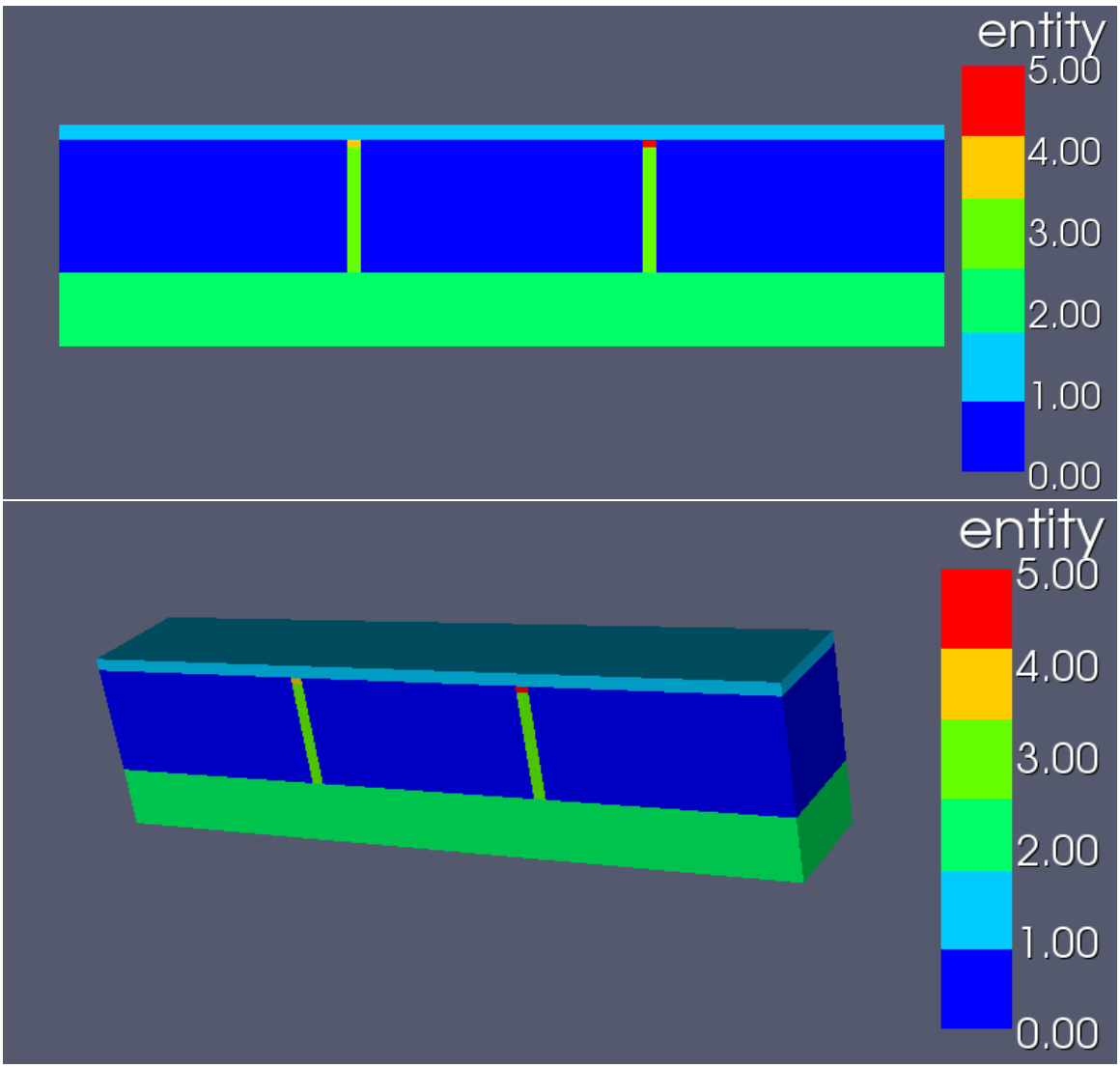


Abbildung 2: Standardentitätsverteilung einer Zweilochanlage in 2D bzw. 3D

5.1 Physikalische Eigenschaften von Wasser

Die auftretenden physikalischen Eigenschaften von Wasser sind in der Regel temperatur- und/oder druckabhängig. Inwiefern diese Unterschiede für die Ergebnisse der Simulation entscheidend sind ist zunächst unklar. Es werden daher für jede Eigenschaft verschiedene Implementierungen angeboten. Diese sind jeweils im angegeben File gesammelt.

Die Klassennamen der einzelnen Implementierungen tragen jeweils den Namen der physikalischen Eigenschaft und eine Endung, die die Implementierung beschreibt. Folgende Endungen sind häufig vertreten:

- `_const`: Konstante, welche über den Konstruktor gesetzt werden kann. Wird dies nicht getan, steht auch ein Default-Wert zur Verfügung. Diese Endung steht für alle Eigenschaften zur Verfügung.
- `_Linear`: lineare Implementierung, deren Parameter ebenfalls optional im Konstruktor gesetzt werden können. Die Defaultwerte wurden so gewählt, dass sie die IAPWS Formulierung im relevanten Bereich möglichst gut approximieren.
- `_IAPWS`: Implementierung der von der International Association for the Properties of Water and Steam[2] (IAPWS) herausgegebenen Formulierungen der entsprechenden Größe. Diese sind in der Regel sehr teuer auszuwerten.

[2] `www.iapws.org`

- `_PolygonalChain35`: implementiert einen Polygonzug mit 35 Stützstellen im Intervall $[280K, 450K]$. Die Stützwerte sind die Werte der IAPWS Formulierung. Diese Implementierung ist - wenn vorhanden - die optimale Mischung aus Approximationsgüte und geringen Auswertungskosten.

Die Plots der verschiedenen Implementierungen können nach Modifizierung mit `make all` im Verzeichnis `./src/parameter` aktualisiert werden.

5.1.1 Dichte von Wasser

Für die Dichte von Wasser müssen die Implementierungen nicht nur den Wert $\rho(T)$ bereitstellen, sondern auch deren Integral. Dies wird bei der Berechnung der hydrostatischen Druckverteilung in Abschnitt 5.4.4 benötigt.

Der Default-Wert für `Density_const` ist $1000kg/m^3$.

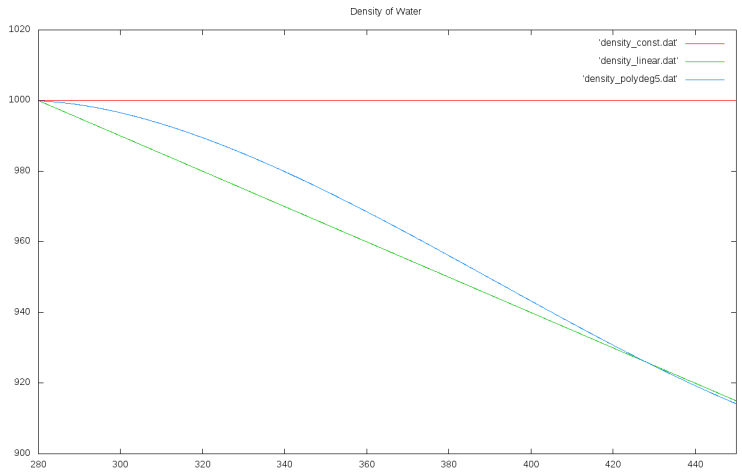
Als lineare Approximation der Dichte wurde $\rho(T) = 1140 - 0.5 \cdot T$ gewählt. Diese kann über den Kostruktor verändert werden.

Für die Dichte steht keine IAPWS-Implementierung zur Verfügung, da es eine weitere gute Formulierung gibt[3]:

$$\rho(T) = \frac{a_0 + a_1 T + a_2 T^2 + a_3 T^3 + a_4 T^4 + a_5 T^5}{1 + bT}$$

Diese Implementierung trägt die Endung `Polydeg5`, die genauen Koeffizienten finden sich im Code.

Abbildung 3: Plot der verschiedenen Implementierungen der Dichte



Die Implementierungen der Dichte finden sich in `./src/parameter/density.hh`

5.1.2 Dynamische Viskosität von Wasser

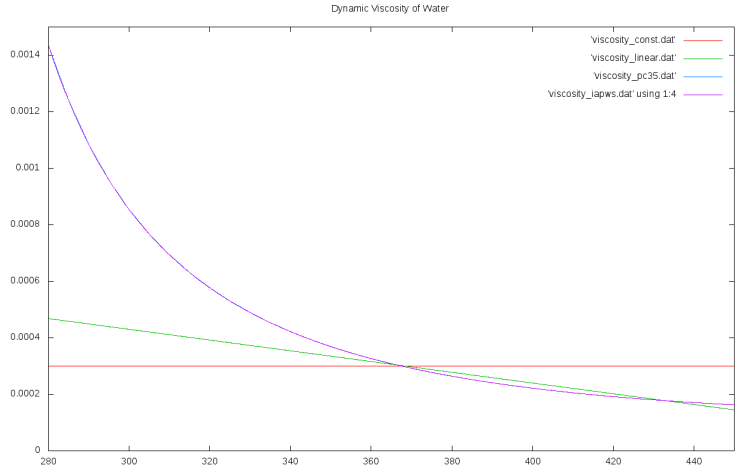
Die IAPWS Formulierung der dynamischen Viskosität[4]von flüssigem Wasser lautet:

$$\mu(\overline{T}) = \frac{100\sqrt{\overline{T}}}{\sum_{i=0}^{3} \frac{H_i}{\overline{T}^i}} \cdot exp\left(\overline{\rho} \sum_{i=0}^{5} \left(\frac{1}{\overline{T}} - 1 \right)^i \sum_{j=0}^{6} H_{ij} \left(\overline{\rho} - 1 \right)^j \right)$$

Auch hier finden sich die Koeffizienten im Code. Die IAPWS Forumlierung ist in diesem Fall sehr teuer auszuwerten, daher wurde ein Polygonzug mit 35 Stützstellen implementiert (Endung `PolygonalChain35`). Für einfachere Rechnungen stehen die konstante Implementierung mit einem Wert von $3 \cdot 10^{-4}\ Pa \cdot s$ und die lineare mit $\mu(T) = 10^{-3} - 1.9 \cdot 10^{-6} \cdot T$ zur Verfügung.
Die Implementierungen der Viskosität finden sich in
`./src/parameter/viscosity.hh`

[3] `http://de.wikipedia.org/wiki/Dichteanomalie`
[4] siehe `http://www.iapws.org/relguide/visc.pdf`

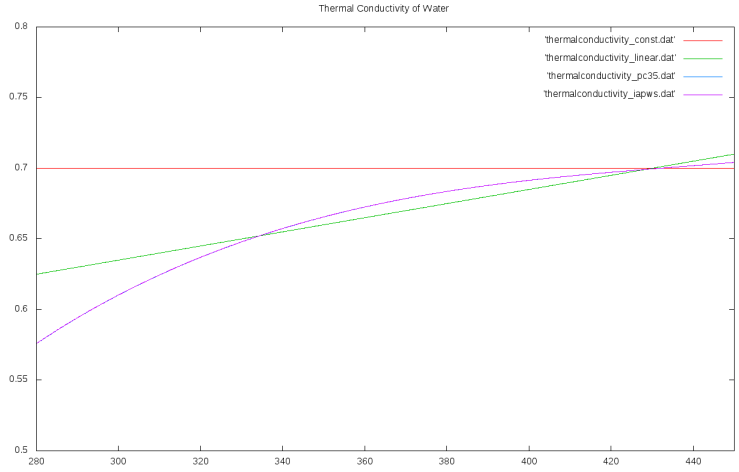
Abbildung 4: Plot der verschiedenen Implementierungen der dynamischen Viskosität



5.1.3 Spezifische Wärmeleitfähigkeit von Wasser

Die IAPWS Formulierung findet sich unter `http://iapws.org/relguide/thcond.pdf`. Wie auch bei der dynamischen Viskosität ist diese sehr teuer auszuwerten und wird daher ersetzt durch einen Polygonzug, die konstante Implementierung mit $0.7\frac{J}{s \cdot K}$ und die lineare mit $\lambda(T) = 0.485 - 5 \cdot 10^{-4} \cdot T$.

Abbildung 5: Plot der verschiedenen Implementierungen der spezifischen Wärmeleitfähigkeit



Die Implementierungen der spezifischen Wärmeleitfähigkeit finden sich in
`./src/parameter/thermalconductivity.hh`

5.1.4 Spezifische Wärmekapazität von Wasser

Wegen der geringen Temperaturabhängigkeit steht hier nur die konstante Implementierung mit Wert $4192\frac{J}{kg \cdot K}$ zur Verfügung.

Die Implementierungen der spezifischen Wärmekapazität finden sich in
`./src/parameter/heatcapacity.hh`

5.2 Geologische Eigenschaften

Parameter der Geologie sind:

- absolute Permeabilität $K(x)$
- Porösität $\eta(x)$
- Wärmekapazität der Gesteinsmatrix c_s

- Dichte der Gesteinsmatrix ρ_s
- der Erdwärmegradient κ

All diese Parameter werden in einer, die Geologie beschreibenden, Klasse zusammengefasst. In `./src/geology` befindet sich bisher nur die Klasse `ModelGeology` deren entsprechende Parameterwerte unten zu finden sind.

Aus c_s und ρ_s wird mittels η die effektive Wärmekapazität bzw. Dichte ausgerechnet:

$$c_e(x) = \eta(x) \cdot c_s + (1 - \eta(x)) \cdot c_w$$
$$\rho_e(x) = \eta(x) \cdot \rho_s + (1 - \eta(x)) \cdot \rho_w$$

Es gibt verschiedene Größen, die in der Literatur im geologischen Zusammenhang als Permeabilität bezeichnet werden. Die korrekte Größe hat die Einheit m^2. Abbildung 5.2 gibt die Werte in den Bereichen des Rechengebiets an.

	Bereich	Wert
0	Aquifer	10^{-13}
1	Schicht oberhalb des Aquifers	10^{-15}
2	isolierende Schicht unterhalb des Aquifers	10^{-15}
3	Bohrloch	10^{-8}
4	Quelle	10^{-8}
5	Senke	10^{-8}
6	Isolation zwischen Quelle und Senke (nur bei 1-Loch-Anlage)	10^{-15}

Abbildung 6: Werte der absoluten Permeabilität in den entsprechenden Bereichen

Die Geologieklasse hat eine Methode `permeability(int entity)` zur Abfrage der Werte in diesen Bereichen. Zur Implementierung komplizierterer Geologien als der `ModelGeology` ist dies allerdings nicht ausreichend. Daher existiert auch eine Methode, die eine globale Koordinate als Parameter bekommt. Das Parameter-Interface ruft zuerst diese Methode auf. Sollte sie 0 zurückgeben, wird die Methode mit Parameter `int` aufgerufen. Auf diesem Wege lassen sich einfach geologische Besonderheiten modellieren.
Für die anderen Parameter habe ich nur Konstanten implementiert. Diese sind

- $\eta = 0.3$
- $c_s = 1300\ J/kg \cdot K$
- $\rho_s = 6000\ kg/m^3$
- $\kappa = -0.035\ K/m$

5.3 Eigenschaften der Anlage

Die wesentlichen Eigenschaften der Anlage sind

- die Pumprate r_P in m^3/h
- die Rücklauftemperatur T_R

Eine Kapselung dieser beiden Parameter scheint übertrieben. Ich habe es jedoch trotzdem getan, um zum Beispiel Experimente, die eine Abschaltung der Anlage miteinbeziehen, leicht realisieren zu können. Die Anlagenklasse sammelt zudem alle Leistungsinformationen in einer `std::map` und kennt die globale Zeit, so dass eine betriebsverlaufsabhängige Abschaltung leicht implementierbar ist.

Die einfachstmögliche Engineklasse findet sich in `./src/engine/trivialengine.hh`. Die Pumprate beträgt 160 m^3/h, die Rücklauftemperatur 343 K. Die ermittelte momentane Entzugsleistung, wird am Ende der Rechnung - in GNUPlot-geeigneter Form - in die Datei `power.dat` geschrieben.

5.4 Die Parameterfunktionen der Modellgleichungen

Die hier aufgelisteten Beschreibungen der Parameterfunktionen sind in `./src/paramterinterface.hh` implementiert.

5.4.1 Wasserquellen und -senken: $f(x,t)$

Der Quellterm ergibt sich direkt aus der Pumprate der Anlage r_P:

$$f(x,t)|_{\Omega_{source}} = \frac{r_P}{3600 \cdot |\Omega_{source}|}$$

Der Faktor 3600 stammt aus der Umrechnung von m^3/h auf m^3/s. Es muss zudem durch die Größe der Quelle geteilt werden, um das eingepumpte Volumen unabhängig von der im Gitter definierten Quellengröße zu machen.

Bei der Berechnung des Senkterms muss folgende Überlegung angestellt werden:

Die angegebene Pumprate bezieht sich auf das Volumen. Da im Allgemeinen allerdings die Dichte des entnommenen und des zugeführten Wassers verschieden ist, muss, um das Entstehen eines Sees an der Oberfläche oder die Notwendigkeit zusätzliches Wasser in das System einzupumpen zu verhindern, der Volumenstrom durch die Senke angepasst werden. Aufgrund der geforderten Massengleichheit gilt

$$\rho(T_R)V^+ = \rho(T_V)V^-$$

mit Volumenströmen V^+, V^-. Für den Senkterm ergibt dies:

$$f(x,t)|_{\Omega_{sink}} = -\frac{\rho(T_R)}{\rho(T_V)} \cdot \frac{r_P}{3600 \cdot |\Omega_{sink}|}$$

In $\Omega \backslash (\Omega_{source} \cup \Omega_{sink})$ gilt selbstverständlich $f(x,t) = 0$.

5.4.2 Wärmequellen und -senken: $g^+(x,t)$ und $g^-(x,t)$

Die Wärmequellen und -senken berechnen sich durch die durch f angegebenen Volumenströme:

$$g^+(x,t)|_{\Omega_{source}} = f(x,t)T_R c_w(T_R)\rho(T_R)$$

Bei der Wärmesenke ist zu beachten, dass die Temperatur erst in den Gleichungen selbst berücksichtigt:

$$\begin{aligned} g^-(x,t)|_{\Omega_{sink}} &= -f(x,t)\rho(T_V)c_w(T_V) \\ &= \rho(T_R)c_w(T_V) \cdot \frac{r_P}{3600 \cdot |\Omega_{sink}|} \end{aligned}$$

5.4.3 Dirichletrand für die Temperatur: $\psi(x,t)$

Das Rechengebiet soll so groß sein, dass an den Rändern die Geschehnisse im Inneren keine Bedeutung mehr haben. Demenstprechend soll dort zu allen Zeitpunkten die Temperaturverteilung des Erdwärmegradienten vorliegen:

$$\psi(x) = 283.15 + \kappa x_3$$

283.15 ist hier die Temperatur an der Oberfläche - in wenigen Metern Tiefe ganzjährig ein guter Näherungswert!

5.4.4 Dirichletrand für den Druck: $\phi(x,t)$

Ränder mit Dirichlet-Randbedingungen müssen der hydrostatischen Druckverteilung entsprechen. Diese kann bei konstanter Dichte einfach als

$$\phi(x) = \rho|g|x_3$$

berechnet werden. Bei temperaturabhängiger Dichte geht dies über in

$$\phi(x) = \int_0^{x_3} \rho(T(h))|g|dh$$

Durch Substitution mit der in Kapitel 5.4.3 beschriebenen Temperaturverteilung erhält man dann

$$\phi(x) = \frac{|g|}{\kappa} \int_{T(0)}^{T(x_3)} \rho(T)dT$$

Dieses Integral wird (wie bereits beschrieben) von den verschiedenen Implementierungen der Dichte von Wasser bereitgestellt werden. Der Vollständigkeit halber wird zudem der Atmospährendruck von $101325Pa$ addiert.

5.4.5 Anfangstemperatur im Gebiet: $T_a(x)$

Zu Beginn entspricht die Temperaturverteilung genau der Verteilung, die aus dem Erdwärmegradient resultiert:

$$T_a(x) = \psi(x)$$

5.4.6 Neumannränder: $U(x,t)$ und $Q(x,t)$

Die Wahl der Randbedingungen, die ich getroffen habe, hat nur Neumann-Ränder über die es keinen Fluss geben soll: $U(x,t) = 0$ bzw. $Q(x,t) = 0$.

6 Verwendete Löser

Zu Testzwecken habe ich lange Zeit mit `SuperLU` gearbeitet. Ab circa 10^5 Zellen ist dies allerdings durch den immensen Speicheraufwand nicht mehr möglich.

Prinzipielle Schwierigkeiten für die Löser sind zum einen die notwendigerweise sehr große Anzahl von Zellen, deren Anisotropie, die aus der Fortpflanzung von kleinen Gitterweiten in die gesamte Raumrichtung resultiert, und die massiven Koeffizientensprünge bei der Permeabilität. Ich habe daher mit vielen in **DUNE** implementierten Lösern experimentiert. Als beste Wahl hat sich erwiesen:

Zum Lösen des Druckproblems kommt ein CG-Verfahren mit ILU(1)-Vorkonditionierer zum Einsatz. Selbst sehr große Systeme werden in akzeptabler Zeit gelöst, vergleiche hierzu Abbildung 7.

Level	Zellen	Zeit
5	291456	1:09 min
6	448448	2:18 min
7	653312	4:15 min
8	912384	7:10 min

Abbildung 7: Performance von CG mit ILU(1)-Vorkonditionierer

Zur Lösung des - weit weniger aufwendigen - Transportproblems kommt der BiCGSTAB zum Einsatz. Die typischen Laufzeiten finden sich in in Abbildung 8.

Level	Zellen	Zeit
5	291456	7.92 sec
6	448448	15.49 sec
7	653312	25.95 sec
8	912384	44.25 sec

Abbildung 8: Performance von BICGSTAB

Die angegebenen Rechenzeiten wurden auf dem Rechner an meinem Arbeitsplatz - Intel Core-2 Duo E8400 Prozessor (3 GHz) mit 8 GB Arbeitsspeicher - gemessen.

7 Ergebnisse

Für alle in diesem Abschnitt genannten Rechnungen gilt, dass eine Zweilochanlage simuliert wird. Die Dichte wird dabei als konstant angesehen, während Viskosität und spezifische Wärmeleitfähigkeit durch die in Kapitel 5.1 beschriebenen Polygonzüge approximiert werden. Geologie und Anlage sind die beschriebenen trivialen Klassen.

7.1 Abschätzung des Diskretisierungsfehlers

Das Abschätzen des Diskretisierungsfehlers auf Basis mehrerer Rechnungen auf verschiedenen Leveln hat mir einige Probleme bereitet. Ich habe versucht die Lösungen quantitativ zu vergleichen, indem ich die Differenz zweier solcher Lösungen in Paraview betrachte. Ein solcher Datensatz ist allerdings schwierig zu erstellen, da die verschiedenen Lösungen auf verschiedenen - nicht ineinander enthaltenen - Gittern leben. Es war mir trotz ausgiebiger Versuche nicht möglich aus dem **DUNE**-code heraus eine VTK-Datei, welche die verschiedenen Lösungen als Datensätze enthält (Ich hatte dazu ein großes

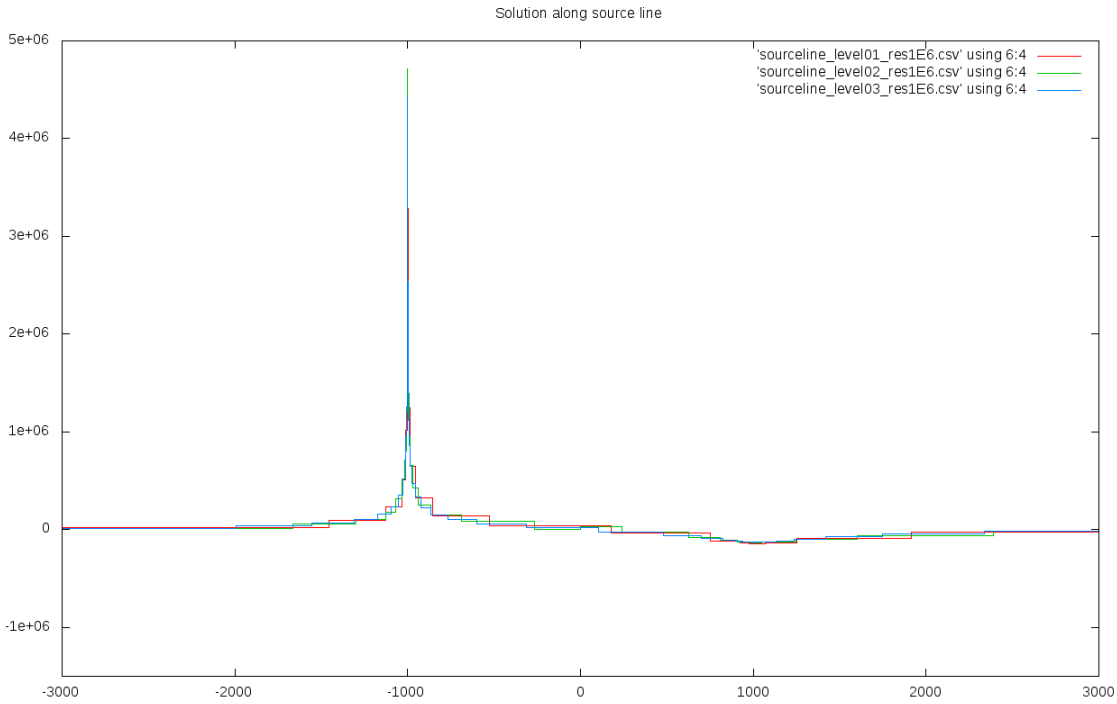


Abbildung 9: Lösungen entlang der Quelllinie

Gitter erstellt, das alle anderen Gitter enthält). Also müssen die Datensätze direkt in Paraview verglichen werden. Auch hier habe ich versucht durch den Calculator eine Differenz zu erstellen. Durch die verschiedenen Gitter kam es allerdings zu fehlerhaften Rechnungen in Paraview.

Die Lösungen werden nun entlang zweier Linien mit dem Paraview-Filter 'Plot Over Line' verglichen. Diese sind die parallel zur x-Achse verlaufenden Linien durch Quelle und Senke, also

$$l_{source}(t) = \begin{pmatrix} 0 \\ 0 \\ \text{-3100} \end{pmatrix} + t \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \qquad t \in [-3000, 3000]$$

bzw.

$$l_{sink}(t) = \begin{pmatrix} 0 \\ 0 \\ \text{-3400} \end{pmatrix} + t \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \qquad t \in [-3000, 3000]$$

Für das Druckproblem wurde dabei nur die Differenz zur hydrostatischen Druckverteilung betrachtet. Die Ergebnisse der Rechnungen zu Level 1-3 (unter 100000 Zellen) finden sich in den Abbildungen 9 und 10. Auf größeren Gittern war auch ein Vergleich mittels dieser Methode nicht möglich, da Paraview ab einer gewissen Gittergröße keine Ergebnisse mehr produziert hat.
Für das Transportproblem finden sich die Ergebnisse - wiederum als Differenz zur geothermischen Verteilung - in den Abbildungen 11 und 12.

7.2 Rechenergebnisse

Für die simulierten Begebenheiten sind keine besonderen Vorkommnisse zu erwarten: Rund um die Quelle bildet sich eine Kaltwasserblase. So lange diese die Senke nicht erreicht, hat die Anlage eine nahezu konstante Leistung von knapp über 10 MW. Im betrachteten Zeitraum von 30 Jahren ist dies nicht der Fall.
Abbildung 13 zeigt die Kaltwasserblase nach 30 jahren. Im Verzeichnis `./results/` befindet das Video `hole.avi` welches die Entstehung zeigt. Dabei wurden 200 Zeitschritte à $5 \cdot 10^6$ s gerechnet.

8 Fazit und Ausblick

Das Modul `dune-geothermics` stellt auf dem aktuellen Stand eine gute Basis für die numerische Simulation hydrothermaler Geothermieanlagen dar. Die gelieferten Interfaces erlauben eine einfache

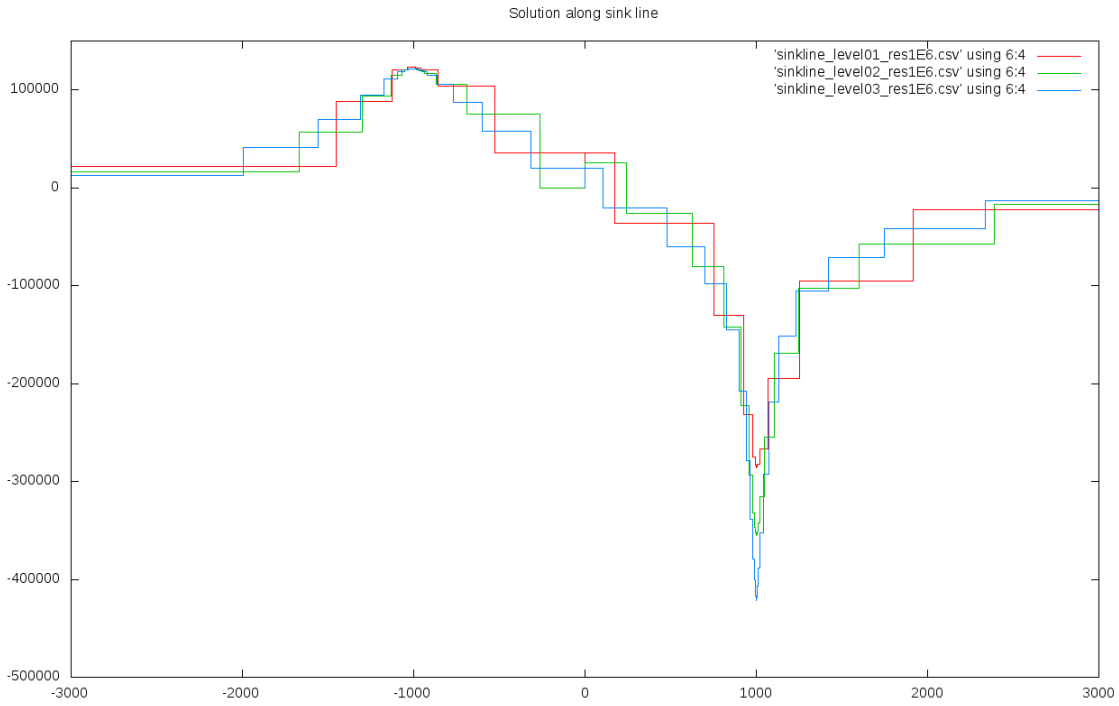


Abbildung 10: Lösungen entlang der Senkenlinie

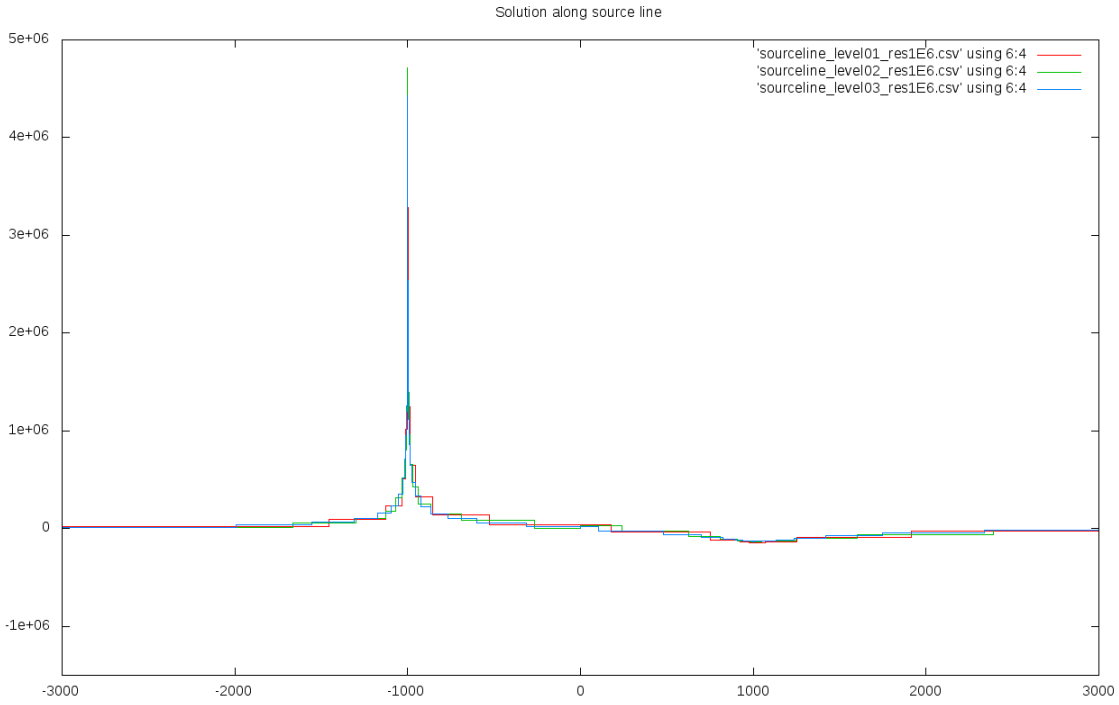


Abbildung 11: Lösungen entlang der Quelllinie

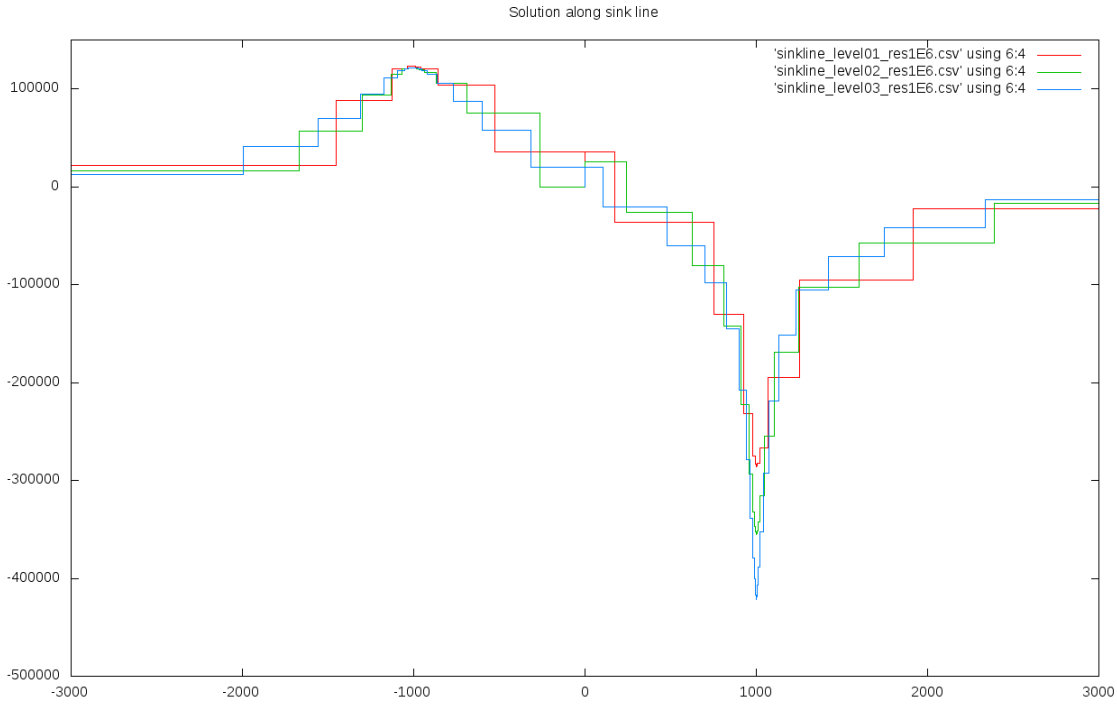


Abbildung 12: Lösungen entlang der Senkenlinie

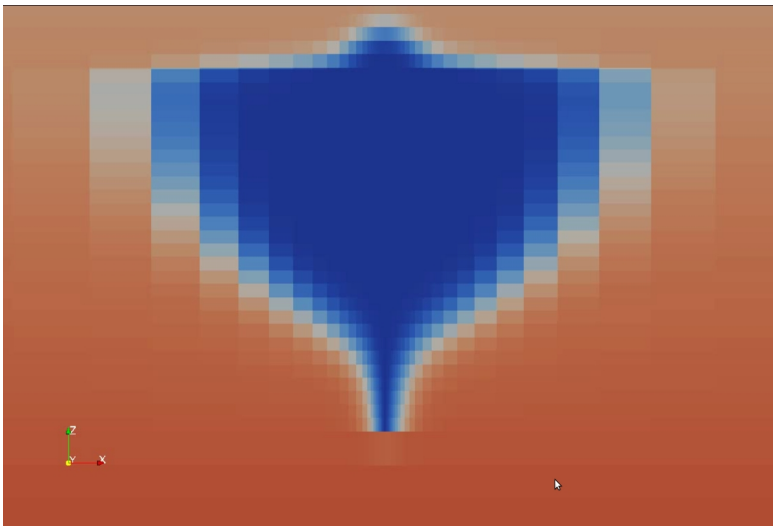


Abbildung 13: Kaltwasserblase rund um die Senke nach 30 Jahren

Anpassung an kompliziertere geologische Szenarien.

Die Arbeit an diesem Praktikum ging allerdings sehr schleppend voran - die Einarbeitung in **DUNE**, die Fehlersuche, das zu Beginn fehlende Konzept im Software-Design und die mangelnde Erfahrung mit Software-Projekten dieser Art im Allgemeinen haben mich lange aufgehalten. Ich konnte daher nur den trivialen Fall der Modellgeologie bei einer Zweilochanlage rechnen.

Anknüpfungspunkte für eine weiterführende Arbeit wären z.B.

- die eingehende Untersuchung von Einlochanlagen
- das Untersuchen von worst-case-Geologien
- eine betriebsverlaufsabhängige Anlagensteuerung

A Subversion

Der komplette Code dieses Praktikums wurde mit dem **DUNE** Trunk vom 8.3.2011 erfolgreich kompiliert. Alle vorgestellten Files finden sich als **DUNE** Modul in meinem Repository unteren

`https://conan.iwr.uni-heidelberg.de/svn/dune-geothermics`

Das Repository enthält die folgenden Dateien:

- Makefile.am
- README
- configure.ac
- doc/
- doc/Makefile.am
- doc/doxygen/
- doc/doxygen/Doxylocal
- doc/doxygen/Makefile.am
- doc/images/
- doc/images/pes1H2D.png
- doc/images/pes1H3D.png
- doc/images/pes2H2D.png
- doc/images/pes2H3D.png
- doc/images/res.jpg
- doc/images/sinkline.png
- doc/images/sourceline.png
- doc/report.tex
- dune/
- dune/Makefile.am
- dune/geothermics/
- dune/geothermics/Makefile.am
- dune/geothermics/geothermics.hh
- dune-geothermics.pc.in
- dune.module
- m4/
- m4/Makefile.am
- m4/dune-geothermics.m4
- results/
- results/hole.avi
- src/
- src/Makefile.am
- src/ccfv.hh
- src/dune_geothermics.cc
- src/engine/
- src/engine/trivialengine.hh
- src/geogridfactory.hh
- src/geology/
- src/geology/model_geology.hh
- src/gridvector.hh
- src/gridvisualization.hh
- src/localop_decoupled_flow.hh
- src/localop_decoupled_transport.hh
- src/localop_time.hh
- src/parameter/
- src/parameter/Makefile
- src/parameter/density.hh
- src/parameter/density_plot.cc
- src/parameter/heatcapacity.hh
- src/parameter/thermalconductivity.hh
- src/parameter/thermalconductivity_plot.cc
- src/parameter/viscosity.hh
- src/parameter/viscosity_plot.cc
- src/parameterinterface.hh
- src/parametertraits.hh
- src/physicalentityselector.hh
- src/t0adapter.hh
- stamp-vc