Technische Universität Berlin
Institut für angewandte Geowissenschaften

Diplomarbeit

Konzeption eines gekoppelten Strömungs- und Stofftransportmodells in GIS GRASS

Sören Gebbert

24. Juli 2007

Betreuer: PD Dr. Traugott Scheytt

Hiermit möchte ich Dipl. Mathematiker Dmitri Naumov für seine Unterstützung und für die vielen konstruktiven Anregungen danken.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Symbolverzeichnis

Lateinische Buchstaben

Symbol	Einheit	Beschreibung
b	$[m]$	Höhe der Grundwassersohle
c	$[kg/m^3]$	Konzentration
c_i	$[kg/m^3]$	Konzentration der Zelle i
c_j	$[kg/m^3]$	Konzentration der Zelle j
$c_{(in)}$	$[kg/m^3]$	Konzentration
D_mol	$[m^2/s]$	molekularer Diffusionskoeffizient
$D^{kl}_{(F)ij}$	$[m^2/s]$	harmonisch gemittelte Komponenten des Diffusionstensors
$D^{kl}_{(S)ij}$	$[m^2/s]$	harmonisch gemittelte Komponenten des Dispersionstensors
$\mathbf{D}_{mol}$	$[m^2/s]$	Tensor der molekularen Diffusion
$\mathbf{D}^*$	$[m^2/s]$	Diffusions-Dispersions-Tensor
D_L	$[m^2/s]$	longitudinaler Diffusionskoeffizient
D_T	$[m^2/s]$	transversaler Diffusionskoeffizient
g	$[m/s^2]$	Erdbeschleunigung
h	$[m]$	piezometrische Druckhöhe
h_i	$[m]$	piezometrische Druckhöhe der Zelle i
h_j	$[m]$	piezometrische Druckhöhe der Zelle j
j_{adv}	$[kg/(m^2s)]$	advektiver Stofffluss
j_{diff}	$[kg/(m^2s)]$	diffusiver Stofffluss
j_{disp}	$[kg/(m^2s)]$	gesamter dispersiver Stofffluss
j_L	$[kg/(m^2s)]$	longitudinaler dispersiver Stofffluss
j_T	$[kg/(m^2s)]$	transversaler dispersiver Stofffluss
K	$[D]$	Permeabilität
k_f	$[m/s]$	Durchlässigkeitsbeiwert
k^{kl}	$[m/s]$	Tensorkomponenten des Durchlässigkeitsbeiwerts
k^{kl}_{ij}	$[m/s]$	harmonisch gemittelte Tensorkomponenten des Durchlässigkeitsbeiwerts
$\mathbf{K}_f$	$[m/s]$	Tensor der Durchlässigkeitsbeiwerte
m	$[m]$	Wassererfüllte Mächtigkeit des Grundwasserleiters
m_i	$[m^3]$	Volumen der Zelle i
m_{ij}	$[m^2]$	Flächengröße des Randes ij
n	$[-]$	Porosität des Grundwasserleiters

n_f	$[-]$	effektive Porosität
p	$[bar]$	Messdruck
p	$[bar]$	Druck des Fluids
p_0	$[bar]$	Luftdruck
p_{ij}	$[-]$	Wichtungsparameter
Pe_g	$[-]$	Gitter-PECLET-Zahl
Q	$[m/s]$	Quellterm
q, q_i	$[m/s]$	Quellterme
R	$[-]$	Retardationsfaktor
S	$[-]$	Speicherkoeffizient
S_s	$[m^{-1}]$	spezifischer Speicherkoeffizient
t	$[s]$	Zeitpunkt
t_0	$[s]$	Anfangszeitpunkt
T	$[m^2/s]$	Transmissivität
$\mathbf{T}$	$[m^2/s]$	Transmissivitätstensor
u	$[m/s]$	Abstandsgeschwindigkeit
$\mathbf{u}$	$[m/s]$	Vektor der Abstandsgeschwindigkeit
$\mathbf{u}_{ij}$	$[m/s]$	Vektor der Abstandsgeschwindigkeit auf dem Rand
u_{ij}^k	$[m/s]$	Komponente der Abstandsgeschwindigkeit auf dem Rand
v	$[m/s]$	Fließgeschwindigkeit
v_f	$[m/s]$	Filtergeschwindigkeit
$\mathbf{v_f}$	$[m/s]$	Vektor der Filtergeschwindigkeit
V_x	$[m/s]$	Fluss über den Rand x
V_y	$[m/s]$	Fluss über den Rand y
V_z	$[m/s]$	Fluss über den Rand z
x	$[m]$	Abstand in x-Richtung
x_i	$[m]$	Abstand in i'te-Richtung
z	$[m]$	Ortshöhe

Griechische Buchstaben

Symbol	Einheit	Beschreibung
α_L	$[m]$	longitudinale Dispersivitätslänge
α_N	$[m]$	numerische Dispersion
α_T	$[m]$	transversale Dispersivitätslänge
β_s	$[Pa^{-1}]$	Kompressibilität des Korngerüsts
β_w	$[Pa^{-1}]$	Kompressibilität des Wassers
δ	$[-]$	Kronecker Delta
$\overline{\delta}$	$[-]$	Negatives Kronecker Delta
Δx	$[m]$	Abstand in x-Richtung
Δy	$[m]$	Abstand in y-Richtung
Δz	$[m]$	Abstand in z-Richtung

Δt	$[s]$	Zeitschritt
η	$[Pa \cdot s]$	Viskosität des Fluids
Γ_{ij}	$[-]$	gemeinsame Fläche der Ränder benachbarter Kontrollvolumen
$\overline{\Lambda}$	$[-]$	Indexmenge aller Kontrollvolumen
Λ_i	$[-]$	Menge der Indizes benachbarter Kontrollvolumen
$\boldsymbol{\nu}$	$[m/s]$	Normalenvektor
$\boldsymbol{\nu}_{ij}$	$[m/s]$	Normalenvektor auf dem Rand
ν_{ij}^k	$[m/s]$	k-te Komponente des Normalenvektors
Ω	$[-]$	Gesamtgebiet
Ω_i	$[-]$	Zelle mit dem Index i
Φ	$[m^2 \cdot kg/s^2]$	Potential
ρ	$[Pa \cdot s]$	Dichte des Fluids
σ	$[kg/s]$	Stofftransportquellterm

Indizes

Indize	Beschreibung
i	Index aller Raster- oder Volumenzellen
j	Index aller Nachbarn der Zelle i
k, l	Laufvariablen der kartesischen Raumkomponenten
m	Index der Randschwerpunkte
n	Index der Randschwerpunkte

Abkürzungen

Abkürzung	Beschreibung
BiCGStab	Stabilisiertes bikonjugierte Gradienten Verfahren
CG	Verfahren der konjugierten Gradienten
CFL	Courant-Fridrich-Lewy-Kriterium
GIS	Geoinformationssystem
gpde	GRASS partial differential equation library
GRASS	***G****eographical* ***R****esources* ***A****nalysis and* ***S****upport* ***S****ystem*

Einleitung

In der Hydrogeologie werden zur Bestimmung und Validierung von Quantität und Qualität des Grundwassers numerische Strömungs- und Stofftransportmodelle eingesetzt. [KR95] So haben sich Strömungsmodelle bei der Bestimmung von Grundwasserbilanzen, bei der Ermittlung von Schutzzonen und Einzugsgebieten zur Trinkwasserversorgung oder bei der Vorhersage von Grundwasserabsenkungen bewährt. Strömungsmodelle dienen des weiteren als Grundlage für Transportmodelle. Diese werden bei der Vorhersage der Ausbreitung einer Verschmutzung im Grundwasser oder bei der Planung von hydraulischen Abwehr- und Sanierungsmaßnahmen eingesetzt.

Die für die Modellierung benötigten Daten müssen im Rahmen von hydrogeologischen Erkundungen, aus dem Datenbestand von Landesämtern sowie verfügbaren analogen sowie digitalen Kartenmaterial zusammengetragen werden. Für die Verwaltung und Aufbereitung der erhobenen hydrogeologischen Parameter haben sich seit einiger Zeit Geoinformationssysteme (GIS) bewährt. Sollen die aufbereiteten Daten in ein Grundwassermodell einfließen, müssen sie in ein für das Modellierungsprogramm lesbare Format exportiert werden.

Diese Vorgehensweise ist sehr weit verbreitet, bringt aber eine Reihe von Nachteilen mit sich. So kann der Transfer der Modelldaten von einem GIS in ein Modellierungsprogramm mit Datenverlust verbunden sein. Ebenso der Rücktransfer der numerischen Ergebnisse aus dem Modellierungsprogramm. Häufig steht man vor dem Problem der Dateninkompatibilität, was unter anderem die weitere Bearbeitung und Betrachtung der Modellierungsergebnisse in einem GIS verhindert.

Der in dieser Arbeit vorgestellte Ansatz versucht diese Nachteile auszuräumen, indem die Strömung und der Stofftransport direkt in einem Geoinformationsystem modelliert wird. Neben der Vermeidung von Datenverlust und Inkompatibilitäten beim Modelldatenaustausch, steht die gesamte GIS Funktionalität zur Datenanalyse, Auswertung und Visualisierung der Modellparameter sowie der Ergebnisse zur Verfügung. Allerdings ist die Wahl des Geoinformationssystems für diesen Ansatz ausschlaggebend. Es sollte sich gut für die Bearbeitung hydrogeologischer Aufgaben eignen und die Möglichkeit zur Realisierung neuer Konzepte und Erweiterungen bieten. Für die Programmierung der Erweiterungen muß der Quellcode des GIS zugänglich sein und möglichst einer offenen Lizenz unterliegen. Auch sollten grundlegende numerische Methoden als Funktionen vorhanden sein, um die Implementierung des Strömungs- und Transportmodells zu erleichtern. Es stellt sich also die Frage, welches GIS diese Anforderungen erfüllt und ob

mit den dort zu implementierenden Erweiterungen, hydrogeologische Modelle erstellt werden können. Weitere Anforderungen betreffen das im GIS implementierte Datenmodell, welches direkten Einfluss auf die Wahl der numerischen Methoden hat. Ist das GIS in der Lage unstrukturierten Gittern oder strukturierten Gittern zu verwalten? Sind damit Modelle in zwei und drei Dimensionen möglich?

Nach eingehender Recherche wurde das Open Source GIS GRASS (*G*eographic *R*esources *A*nalysis *S*upport *S*ystem) als Ausgangspunkt für das hier vorgestellte Konzept gewählt. Als Aufgabenstellung soll deshalb, basierend auf dem Geoinformationssystem GRASS, ein numerisches Modell zur Lösung von Strömungs- und Stofftransportproblemen konzipiert und implementiert werden. Desweiteren wird die Eignung des GIS GRASS für die gewählte Konzeption untersucht.

1 Grundlagen der Strömungs- und Stofftransportmodellierung

Die im Grundwasser ablaufenden Strömungs- und Transportprozesse können durch Strömungs- und Stofftransportmodelle beschrieben werden. Voraussetzung für die Modellierung solcher Prozesse ist das Wissen über die relevanten physikalische Parameter und Gesetze. Ein Großteil der verfügbaren Modellierungssoftware sowie die in dieser Arbeit entwickelten Programme setzten dieses Wissen voraus, um physikalisch korrekte Modelle zu erstellen. Aus diesem Grund sollen folgende grundlegende physikalische Parameter und Gesetze der Strömungs- und Stofftransportmodellierung wie Permeabilität, Durchlässigkeitsbeiwert, Transmissivität, Speicherkoeffizient, effektive Porosität, das Gesetz von Darcy, Advektion, Molekulare Diffusion und Dispersion, Retardation und Stoffabbau eingeführt werden.

Die Grundlage für eine Transportmodellierung bildet meist ein Strömungsmodell. In Folge dessen wird zuerst das Strömungsmodell hergeleitet, welches in den Advektionterm und den Dispersionstensor der Transportgleichung einfließt.

1.1 Physikalische Parameter der Strömungsgleichung

Das in dieser Arbeit hergeleitete Strömungsmodell, bezieht sich ausschließlich auf Porengrundwasserleiter. Dessen physikalischen Parameter sollen im Folgenden erläutert werden.

Permeabilität K

Die Permeabilität beschreibt die Durchlässigkeit eines Gesteins gegenüber beliebigen Fluiden. Diese Größe ist gesteinspezifisch und nicht von den Fluideigenschaften abhängig. Die Permeabilität K wird in Darcy angegeben $K = 1$ Darcy.

Effektive Porosität n_f

Poröse Medien wie Sedimente und Böden bestehen aus einer Matrix fester Mineralkörper, getrennt und umgeben von Lücken, Poren und Hohlräumen [MU83]. Diese Hohlräume werden im Lockergestein üblicherweise als Porosität bezeichnet. Der Anteil des Hohlraumes, in dem sich Grundwasser bewegen kann, abzüglich des adhäsiven und kapillar gebundenen Wassers, wird als effektiv Porosität n_f bezeichnet. Die effektive Porosität ist einheitslos.

Durchlässigkeitsbeiwert k_f

Der Durchlässigkeitsbeiwert k_f ist ein in der Hydrogeologie gebräuchlicher Parameter, um die Durchlässigkeit eines Grundwasserleiters zu beschreiben. Der k_f Wert kombiniert die physikalische Eigenschaft des Gesteins (Permeabilität) mit den physikalischen Eigenschaften des Fluids. Es fließen die Viskosität, die Temperatur, die Dichte und die Reynoldsche Zahl in die Größe ein. Der k_f Wert ist gültig, wenn die Reynoldsche Zahl kleiner eins ist und somit laminares fließen vorherrscht. Turbuntes fließen beginnt ab einer Reynoldschne Zahl zwischen eins und zehn. Der k_f Wert wird meist direkt durch Siebanalysen des Grundwasserleitersediments oder Pumpversuchen bestimmt. Nach [MU83] kann er folgendermaßen berechnet werden:

$$k_f = \frac{\rho}{\eta} g K$$

k_f = Durchlässigkeitsbeiwert $[\frac{m}{s}]$

ρ = Dichte des Fluids $[\frac{kg}{m^3}]$,

η = Viskosität des Fluids $[Pa \cdot s]$,

g = Erdbeschleunigung $[\frac{m}{s^2}]$,

K = Permeabilität des Gesteins $[D]$.

Wird von einem anisotropen, inhomogenen Grundwasserleiter ausgegangen, kann der Durchlässigkeitsbeiwert k_f durch einen symmetrischen Tensor zweiter Ordnung $\mathbf{K}_f$ beschrieben werden. Die einzelnen Tensoreinträge k^{kl} beschreiben den spezifischen Durchflussbetrag.

Der Durchlässigkeitstensor ist in zwei Dimensionen wie folgt definiert:

$$\mathbf{K}_f = \begin{bmatrix} k^{xx} & k^{xy} \\ k^{yx} & k^{yy} \end{bmatrix}.$$

In drei Dimensionen hat er folgende Gestalt:

$$\mathbf{K}_f = \begin{bmatrix} k^{xx} & k^{xy} & k^{xz} \\ k^{yx} & k^{yy} & k^{yz} \\ k^{zx} & k^{zy} & k^{zz} \end{bmatrix}.$$

Transmissivität T

Die Transmissivität T ist eine Kombination aus dem Durchlässigkeitsbeiwert und der wassererfüllten Mächtigkeit m des Grundwasserleiters.

Für gespannte Grundwasserleiter berechnet sich die Transmissivität wie folgt

$$T = k_f \cdot m$$

T = Transmissivität $[\frac{m^2}{s}]$

m = Wassererfüllte Mächtigkeit des Grundwasserleiters $[m]$.

Für ungespannte Grundwasserleiter wird die wassererfüllte Mächtigkeit m aus der Differenz der Höhe der Grundwassersohle b und der piezometrischen Druckhöhe der Wasseroberfläche h berechnet:

$$T = k_f \cdot (h - b)$$

b = Höhe der Grundwassersohle $[m]$,

h = piezometrischen Druckhöhe der Wasseroberfläche $[m]$.

Die Transmissivität wird in einem anisotropen, inhomogenen Grundwasserleiter durch einen Tensor zweiter Ordnung $\mathbf{T}$ beschrieben [KR95].

$$\mathbf{T} = m\, \mathbf{K}_f$$

Spezifische Speicherkoeffizient S_s

Der spezifische Speicherkoeffizient S_s beschreibt die spezifische Speichermenge $[\frac{1}{m}]$, also die gespeicherte Wassermenge $[m^3]$, die aus einer Volumeneinheit Grundwasserleiter $[m^3]$ je Einheit Höhenabsenkung $[m]$ frei wird [KR95]:

$$S_s = \rho g[(1-n)\beta_s + n\beta_w] \tag{1.1}$$

β_s = Kompressibilität des Korngerüsts $[\frac{1}{Pa}]$,

β_w = Kompressibilität des Wassers $[\frac{1}{Pa}]$,

n = Porosität des Grundwasserleiters [-].

In einem gespannten Grundwasserleiter der Mächtigkeit m ergibt sich durch Integration über die Mächtigkeit der dimensionslosen Speicherkoeffizient S durch:

$$S = m\, S_s$$

1.2 Grundwasserbewegung

Das Grundwasser bewegt sich auf Grund von Energiepotentialdifferenzen durch den Grundwasserleiter. Es fließt immer vom höheren zum niedrigeren Potential. Das Gesamtpotential einer Einheitsmasse an einem gegebenen Punkt ist nach aus [MU83]:

$$\Phi = gz + \int_{p_0}^{p} \frac{dp}{\rho g} + \frac{v^2}{2}, \tag{1.2}$$

Φ = Gesamtenergie einer Einheitsmasse $[\frac{m^2 kg}{s^2}]$,

p = Messdruck $[bar]$,

p_0 = Luftdruck $[bar]$,

ρ = Dichte des Fluids $[\frac{kg}{m^3}]$,

z = Ortshöhe $[m]$,

v = Fließgeschwindigkeit $[\frac{m}{s}]$.

Das Quadrat der Fließgeschwindigkeit ist im Allgemeinen gegenüber den anderen Parametern so klein, dass es hier vernachlässigt werden kann. Somit fällt der letzte Term weg. Nimmt man außerdem an, dass das Fluid inkompressibel ist, so vereinfacht sich die Gleichung (1.2) nach [MU83] S.156 zu

$$\Phi = gz + \frac{p - p_0}{\rho g}.$$

Piezometrische Druckhöhe h

In der Grundwassermodellierung rechnet man meist mit der piezometrischen Druckhöhe. Diese ist direkt proportional zum Strömungspotential $\Phi = gh$ und nach [MU83] S.156 folgendermaßen definiert:

$$h = \frac{p}{\rho g} + z,$$

h = Piezometrische Druckhöhe $[m]$,

p = Druck des Fluids $[bar]$,

ρ = Dichte des Fluids $[\frac{kg}{m^3}]$.

Für jeden Punkt in einem Grundwasserleiter kann somit die piezometrische Druckhöhe berechnet werden. Die Druckhöhe gibt die Höhe der Wassersäule über dem Bezugspunkt z an, welche mit dem atmosphärischen Druck im Gleichgewicht steht. An der Grundwasseroberfläche ist $h = z$. Dies bedeutet, dass es keine überlagernde Wassersäule gibt und damit der Druck p Null ist. Die Druckhöhe h ist auch deshalb sehr praktisch, da man diese direkt in Brunnen oder Grundwassermeßstellen mißt.

Das Gesetz von Darcy

Grundlage der Strömungsberechnungen von Grundwasser und vielen anderen Fluiden bildet das Darcysche Gesetz. Es besagt, dass Wasser dann fließt, wenn es ein Potentialgefälle gibt und sich das Fluid durch das Gestein bewegen kann. Im eindimensionalen Fall ist es folgendermaßen definiert:

$$v_f = -k_f \cdot \frac{\partial h}{\partial x},$$

x = Abstand in x-Richtung $[m]$,

v_f = Filtergeschwindigkeit $[\frac{m}{s}]$.

Die Filtergeschwindigkeit v_f ergibt sich also aus dem hydraulischen Gradienten $\frac{\partial h}{\partial x}$ und dem Durchlässigkeitsbeiwert. Das Gesetz von Darcy gilt nur, wenn die Strömung des Fluids in den Zwischenräumen des Grundwasserleiters laminar ist. Laminare Strömung ist in Abbildung 1.1 dargestellt. Dieses Gesetz bildet die Grundlage der Strömungsdifferentialgleichung. Die Filtergeschwindigkeit ist nicht auf eine Dimension beschränkt, im mehrdimensionalen Fall wird der resultierende Geschwindigkeitsvektor $\mathbf{v}_f$ folgendermaßen berechnet

$$\mathbf{v}_f = -k_f \cdot \frac{\partial h}{\partial x_i}, \tag{1.3}$$

k_f = Durchlässigkeitsbeiwert $[\frac{m}{s}]$,

$\mathbf{v}_f$ = Filtergeschwindigkeitsvektor $[\frac{m}{s}]$,

x_i = Abstand in die jeweilige Raumrichtung $[m]$.

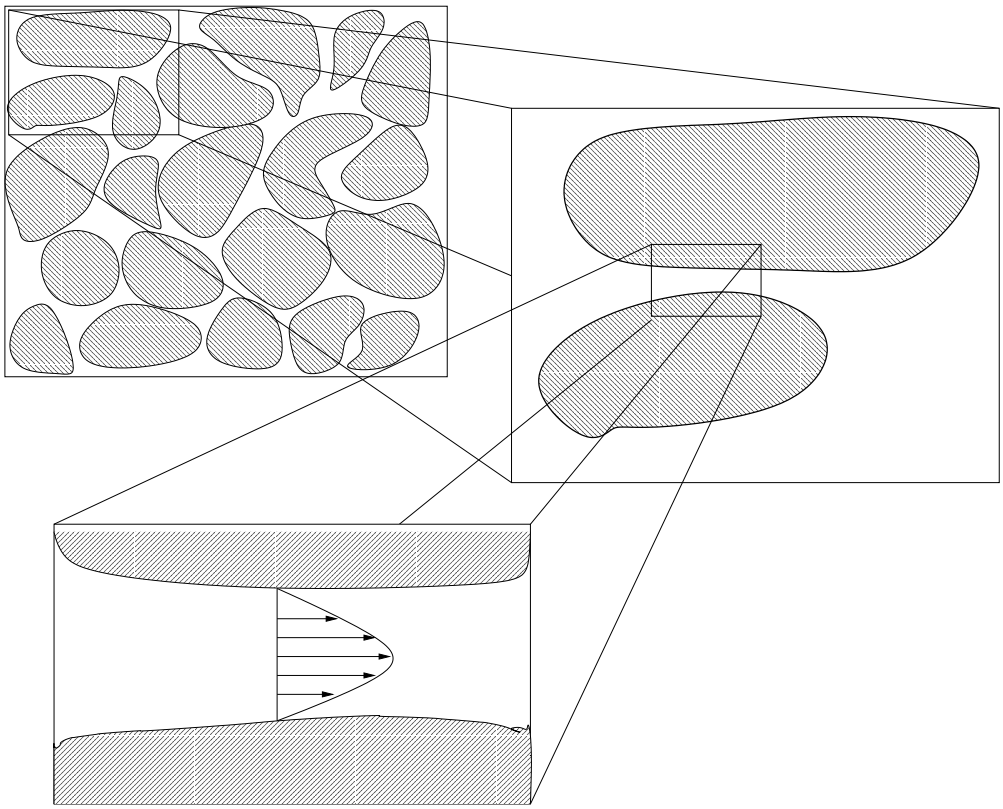


Abbildung 1.1: Parabolisches Geschwindigkeitsprofil der laminaren Strömung eines Fluids im Porengrundwasserleiter.

Für einen inhomogenen, anisotropen Grundwasserleiter, läßt sich die Berechnung der Filtergeschwindigkeit wie folgt ausdrücken

$$\mathbf{v}_f = -\mathbf{K}_f \cdot \frac{\partial h}{\partial x_i} = -\mathbf{K}_f \cdot \nabla h. \tag{1.4}$$

In das Transportmodell fließt die Abstandsgeschwindigkeit $\mathbf{u}$ ein. Diese berechnet sich aus der Division der Filtergeschwindigkeit mit der effektiven Porosität n_f:

$$\mathbf{u} = \frac{\mathbf{v}_f}{n_f}$$

1.3 Die Strömungsdifferentialgleichung

Aufbauend auf den im vorherigen Abschnitt vorgestellten hydrogeologischen Größen und Parameter soll nun die allgemeine Strömungsdifferentialgleichung hergeleitet werden. Diese beschreibt die Grundwasserbewegung in Porengrundwasserleitern und bildet die Grundlage der Strömungsmodellierung. Diese Gleichung gilt für gespannte und ungespannte, (an)isotrope sowie (in)homogene Grundwasserleiter.

1.3.1 Herleitung

Die Grundlage bildet das Gesetz von Darcy, siehe Gleichung (1.4). Es wird vorausgesetzt, dass das Wasser laminar durch den Grundwasserleiter strömt, (siehe Abbildung 1.1). Die Strömungsdifferentialgleichung hängt noch von weiteren Parametern ab, die der Einfachheit halber hier nicht näher erläutert werden. Dies sind die Dichte des Fluids, die Viskosität und die Temperatur. Um diese Parameter nicht zusätzlich zu beachten, müssen bestimmte Annahmen getroffen werden. Deshalb sind für die hier betrachteten Problemstellungen Dichte-, Viskositäts- und Temperaturveränderungen auszuschließen.

Um zu einer Differentialgleichung zu gelangen die den Grundwasserfluss beschreibt, wird eine Bilanzgleichung für ein infinitesimales rechteckiges Kontrollvolumen herangezogen [KR95]. Über ein bestimmtes Zeitschritt Δt wird der Zu- und Abfluss über die Oberfläche des Volumens betrachtet. Dabei wird auch eine innere Quelle Q berücksichtigt sowie der spezifische Speicherkoeffizient S_s. Das Kontrollvolumen ist in Abbildung 1.2 dargestellt. Die Summe aller Zu- und Abflüsse entspricht der Änderung des Wasservolumens für

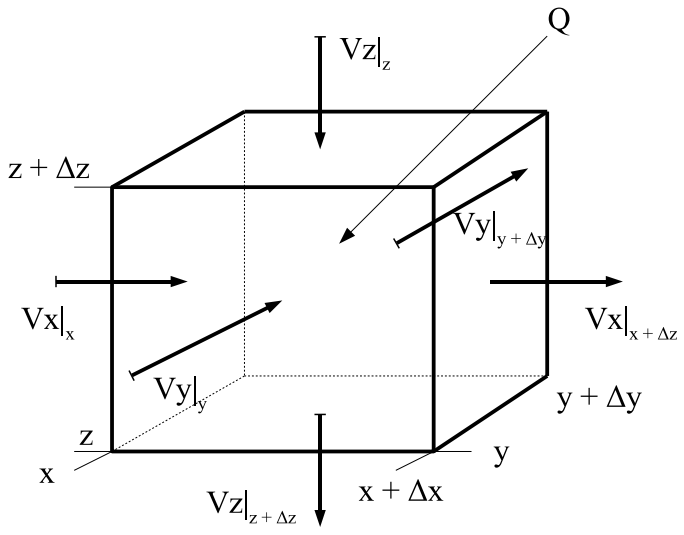


Abbildung 1.2: Rechteckiges Kontrollvolumen zur Herleitung der Strömungsdifferentialgleichung.

einen Zeitschritt Δt.

$$\Delta t[(\Delta z Vx|_x)\Delta y - (\Delta z Vx|_{x+\Delta x})\Delta y + (\Delta z Vy|_x)\Delta x - (\Delta z Vy|_{y+\Delta y})\Delta x + Vz|_z \Delta x \Delta y - Vz|_{z+\Delta z})\Delta x \Delta y + Q] = S_s \Delta x \Delta y \Delta z (h(t+\Delta t) - h(t)) \quad (1.5)$$

Teilt man nun durch $\Delta x \Delta y \Delta z$ und Δt entstehen folgende Differenzenquotienten

$$\frac{Vx|_x - Vx|_{x+\Delta x}}{\Delta x} + \frac{Vy|_y - Vy|_{y+\Delta y}}{\Delta y} + \frac{Vz|_z - Vz|_{z+\Delta z}}{\Delta z} + q = \frac{S_s(h(t+\Delta t) - h(t))}{\Delta t}. \quad (1.6)$$

Führt man einen Grenzübergang $\Delta x \to 0$, $\Delta y \to 0$, $\Delta z \to 0$, $\Delta t \to 0$ durch, ergibt sich mit

$$\mathbf{v}_f = (Vx, Vy, Vz) \text{ und } \boldsymbol{\nabla} = (\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}).$$

folgende Gleichung

$$-\boldsymbol{\nabla} \cdot \mathbf{v}_f + q = S_s \frac{\partial h}{\partial t}. \quad (1.7)$$

Da

$$\mathbf{v}_f = -\mathbf{K}_f \cdot \boldsymbol{\nabla} h,$$

erhält man durch Einsetzen die Differentialgleichung [KR95]

$$\boldsymbol{\nabla} \cdot \mathbf{K}_f \cdot \boldsymbol{\nabla} h + q = S_s \frac{\partial h}{\partial t}. \quad (1.8)$$

Diese Gleichung bildet die Grundlage der instationären Stömungsmodellierung. Sie hat die Einheit $[s^{-1}]$. Falls stationäre Bedingungen untersucht werden, fällt der Speicherterm $S_s \frac{\partial h}{\partial t}$ weg und die Gleichung vereinfacht sich zu

$$\boldsymbol{\nabla} \cdot \mathbf{K}_f \cdot \boldsymbol{\nabla} h + q = 0. \quad (1.9)$$

Für die zweidimensionale Grundwassermodellierung wird oft die Transmissivität an Stelle des Durchlässigkeitsbeiwertes verwendet. Die Differentialgleichung für einen gespannten Grundwasserleiter verändert sich zu

$$\boldsymbol{\nabla} \cdot \mathbf{T} \cdot \boldsymbol{\nabla} h + q = S \frac{\partial h}{\partial t}. \quad (1.10)$$

wobei der spezifische Speicherkoeffizient S_s durch den über die grundwassererfüllte Mächtigkeit integrierte Speicherkoeffizient S ersetzt wird.

Im ungespannten Grundwasserleiter wird die Grundwassermächtigkeit m aus der piezometrischen Druckhöhe h und der Höhe der Sohle des Grundwasserleiters b, vergleiche Gleichung (1.1), berechnet. Da in diesem Fall

$$\mathbf{T} = m\mathbf{K}_f = (h-b)\mathbf{K}_f$$

ist, wird die Strömungsdifferentialgleichung nichtlinear in h.

$$\boldsymbol{\nabla} \cdot (h-b)\mathbf{K}_f \cdot \boldsymbol{\nabla} h + q = S \frac{\partial h}{\partial t} \quad (1.11)$$

1.3.2 Anfangsbedingungen

Die Anfangsbedingungen stellen die Piezometerhöhenverteilungen zu einem Anfangszeitpunkt $t = t_0$ dar.

1.3.3 Randbedingungen

Im Bezug auf die numerische Lösung von Differentialgleichungen werden meist drei Randbedingungen für gespannte und ungespannte Grundwasserleiter verwendet, dies sind: DIRICHLET, NEUMANN und CAUCHY Randbedingungen.

1. Art DIRICHLET Bedingungen

Die Randbedingungen der ersten Art schreiben Piezometerhöhen am Rand vor. Die Lösung der Differentialgleichung ist an diesen Stellen vorgegeben, $h = f(t)$. Ein Spezialfall der ersten Randbedingungen ist der so genannte Festpotentialrand. Hier ist die Piezometerhöhenverteilung am Rand konstant $h = \text{const}$. Typische Anwendungsfälle sind Seen oder Flüsse mit vollständigem hydraulischem Kontakt zum Grundwasserleiter.

2. Art NEUMANN Bedingungen

Durch diese Randbedingungen wird der Zufluss oder Abfluss über den Rand in Richtung der Normalen beschrieben. In einem isotropen Grundwasserleiter entspricht das einem Gradienten von h senkrecht zum Rand ($\frac{\partial h}{\partial n} = \text{const}$). Ein sehr häufig verwendeter Spezialfall dieser Randbedingungen ist der undurchlässige Rand ($\frac{\partial h}{\partial n} = 0$). Man spricht hier von natürlichen NEUMANN Randbedingungen. Wasserscheiden und Randstromlinien werden z.B. als undurchlässige Ränder modelliert. Auch die Grundwasserneubildung, Brunnen sowie Randzuflüsse können durch diese Randbedingungen modelliert werden.

3. Art CAUCHY Bedingungen

Die CAUCHY Randbedingungen stellen eine Mischung aus den ersten und zweiten Randbedingungen dar. Sie schreiben eine Linearkombination

$$const = \alpha h + \beta \frac{\partial h}{\partial n}$$

auf dem Rand vor. Dabei wird das äußere Potential durch einen Widerstand abgeschwächt im Grundwasserleiter wirksam. Sie werden benutzt, wenn Flüsse oder Seen

mit konstanten Wasserstand influente oder effluente Verhältnisse erzeugen und über eine weniger durchlässige Schicht an den Grundwasserleiter angeschlossen sind. Dieser so genannte Leakage Effekt tritt auch zwischen einzelnen Grundwasserleitern auf.

1.4 Physikalische Parameter und Gesetzmäßigkeiten des Stofftransportes

Meist bildet ein Strömungsmodell die Grundlage der Transportmodellierung [RSW02]. Die aus dem Strömungsmodell berechnete Abstandsgeschwindigkeit fließt als Geschwindigkeitskomponente in das Transportmodell ein. Es wird im weiteren davon ausgegangen, dass die Konzentration der im Wasser gelösten Stoffe so niedrig ist, dass die Dichte und Viskosität des Grundwassers nicht verändert wird. Die für den Stofftransport wesentlichen Prozesse sollen im Folgenden vorgestellt werden.

Advektion

Unter Advektion versteht man den Transport des gelösten Stoffes mit der Abstandsgeschwindigkeit des Grundwassers. Der advektive Stoffffluss j_{adv} über die Einheitsfläche eines repräsentativen Kontrollvolumens ist gegeben durch [RSW02]:

$$j_{adv} = u \cdot n_f \cdot c$$

u = Abstandsgeschwindigkeit $[\frac{m}{s}]$,

n_f = durchflusswirksame Porosität [-],

c = Konzentration des Stoffes im Grundwasser $[\frac{kg}{m^3}]$

Molekulare Diffusion

Die molekulare Diffusion wird durch die Brown'sche Molekularbewegung verursacht und läßt sich makroskopisch durch das Fick'sche Gesetz beschreiben. Der diffuse Massenfluss j_{diff} ist dabei proportional zum Konzentrationsgradienten. Der Proportionalitätsfaktor ist der molekulare Diffusionskoeffizient [LKW96]:

$$j_{diff} = -n_f D_m ol \cdot \frac{\partial c}{\partial x}$$

j_{diff} = diffusiver Stoffffluss $[\frac{kg}{m^2 s}]$,

D_{mol} = molekularer Diffusionskoeffizient $[\frac{m^2}{s}]$,

$\frac{\partial c}{\partial x}$ = Konzentrationsgradient in x-Richtung $[\frac{kg}{m^3 m}]$.

In einem anisotropen Medium, in welchem sich die Komponenten der molekularen Diffusion an den Achsen der kartesischen Raumkoordinaten orientieren, bildet sich der molekulare Diffusionstensor folgendermaßen:

$$\mathbf{D}_{mol} = \begin{matrix} D^{xx}_{(F)} & 0 & 0 \\ 0 & D^{yy}_{(F)} & 0 \\ 0 & 0 & D^{zz}_{(F)} \end{matrix} .$$

Dispersion

Einen größeren Einfluss als die molekulare Diffusion besitzt die korngerüstbedingte Dispersion. Sie ist skalenabhängig und wird durch die unterschiedlichen Geschwindigkeiten des Grundwassers auf Korn- und Sedimentebene verursacht. Hier wird nur die so genannte Makrodispersion betrachtet, da der Aufbau des Sedimentkörpers meist nur recht ungenau erfasst werden kann. Die Makrodispersion ist richtungsabhängig und wird nach [RSW02] wie folgt beschrieben:

$$j_L = -n_f D_L \frac{\partial c}{\partial s_L}$$
$$j_T = -n_f D_T \frac{\partial c}{\partial s_T}$$
$$j_{disp} = j_L + j_T$$

j_L = dispersiver Stofffluss longitudinal (entlang) der Strömungsrichtung $[\frac{kg}{m^2 s}]$,

j_T = dispersiver Stofffluss transversal zur Strömungsrichtung $[\frac{kg}{m^2 s}]$,

j_{disp} = gesamter dispersiver Stofffluss $[\frac{kg}{m^2 s}]$,

D_L = longitudinaler Dispersionskoeffizient $[\frac{m^2}{s}]$,

D_T = transversaler Dispersionskoeffizient $[\frac{m^2}{s}]$,

$\frac{\partial c}{\partial s_L}$ = Konzentrationsgradient in longitudinaler Richtung $[\frac{kg}{m^3 m}]$,

$\frac{\partial c}{\partial s_T}$ = Konzentrationsgradient in transversaler Richtung $[\frac{kg}{m^3 m}]$.

Die Dispersionskoeffizienten sind vom Strömungsfeld der Grundwasserströmung abhängig. Longitudinale Dispersion findet nur in Strömungsrichtung statt, transversale in der Senkrechten dazu. Deswegen lassen sich die Dispersionskoeffizienten in einem ungleichförmigen Strömungsfeld, annähernd durch den Strömungsvektor $\mathbf{u}$ und die longitudinalen und transversalen Dispersivitätslängen α_L und α_T errechnen.

Zusammen ergeben diese den symmetrischen Dispersivitätstensor $\mathbf{D}_{disp}$ zweiter Ordnung für ein ungleichförmiges Strömungsfeld

$$\mathbf{D}_{disp} = \begin{matrix} D^{xx}_{(S)} & D^{xy}_{(S)} & D^{xz}_{(S)} \\ D^{yx}_{(S)} & D^{yy}_{(S)} & D^{yz}_{(S)} \\ D^{zx}_{(S)} & D^{zy}_{(S)} & D^{zz}_{(S)} \end{matrix} .$$

Dieser setzt sich aus den Dispersivitätslängen α_L und α_T sowie dem Strömungsvektor $\mathbf{u} = (u^x, u^y, u^z)$ zusammen.

$$\begin{aligned}
D^{xx}_{(S)} &= \alpha_L \frac{(u^x)^2}{|\mathbf{u}|} + \alpha_T \frac{(u^y)^2}{|\mathbf{u}|} + \alpha_T \frac{(u^z)^2}{|\mathbf{u}|} \\
D^{yy}_{(S)} &= \alpha_T \frac{(u^x)^2}{|\mathbf{u}|} + \alpha_L \frac{(u^y)^2}{|\mathbf{u}|} + \alpha_T \frac{(u^z)^2}{|\mathbf{u}|} \\
D^{zz}_{(S)} &= \alpha_T \frac{(u^x)^2}{|\mathbf{u}|} + \alpha_T \frac{(u^y)^2}{|\mathbf{u}|} + \alpha_L \frac{(u^z)^2}{|\mathbf{u}|} \\
D^{yx}_{(S)} &= D^{xy}_{(S)} = (\alpha_L - \alpha_T) \frac{u^x u^y}{|\mathbf{u}|} \\
D^{zx}_{(S)} &= D^{xz}_{(S)} = (\alpha_L - \alpha_T) \frac{u^x u^z}{|\mathbf{u}|} \\
D^{zy}_{(S)} &= D^{yz}_{(S)} = (\alpha_L - \alpha_T) \frac{u^z u^y}{|\mathbf{u}|}
\end{aligned}$$

Im zweidimensionalen Fall fallen die Terme $D^{xz}_{(S)}$, $D^{yz}_{(S)}$, $D^{zz}_{(S)}$,$D^{zy}_{(S)}$ und $D^{zx}_{(S)}$ weg.

Der Diffusions-Dispersions Tensor $\mathbf{D}^*$ ergibt sich aus der Addition

$$\mathbf{D}^* = \mathbf{D}_{mol} + \mathbf{D}_{disp}.$$

Reaktion

Die im weiteren betrachteten Reaktionsprozesse sind Sorptionsprozesse der gelösten Schadstoffe mit dem Aquifermaterial. Die Sorptionsprozesse sind abhängig vom Aquifermaterial sowie den physikalisch-chemischen Eigenschaften des Schadstoffes.

In der verwendeten mathematischen Formulierung der Transportgleichung wird auf Formen der Sorption mittels der Retardierung eines Wasserinhaltstoffes eingegangen. Diese

ist meist eine Funktion der gelösten Schadstoffkonzentration. Der Retardationsfaktor R wird wie folgt definiert:

$$R = (1 + \frac{\rho_{mat} n_{mat}}{n_f} \frac{\partial f}{\partial c}) \qquad \text{allgemeine Beschreibung}$$

$$R = (1 + \frac{\rho_{mat} n_{mat} k_d}{n_f}) \qquad \text{lineare Isotherme}$$

$$R = (1 + \frac{\rho_{mat} n_{mat}}{n_f} n k c^{n-1}) \qquad \text{Freundlich Isotherme}$$

$$R = (1 + \frac{\rho_{mat} n_{mat}}{n_f} \frac{c_{a,max} k_L}{(1 + k_L c)^2}) \qquad \text{Langmuir Isotherme}$$

n_{mat} = Anteil des Volumens des Aquifermaterials am Gesamtvolumen [-],

ρ_{mat} = Dichte des Aquifermaterials $[\frac{kg}{m^3}]$.

Als Vereinfachung wird nur die lineare Isotherme im numerischen Modell berücksichtigt, da die Transportgleichung sonst nichtlinear in der Konzentration wird.

1.5 Die Transportdifferentialgleichung

Die Bewegung von gelösten Stoffen im Grundwasser ist außerordentlich komplex [LKW96]. In die Transportgleichung, die diese Bewegung beschreibt, fließen die im vorherigen Abschnitt eingeführten Prozesse: *Diffusion*, *Dispersion*, *Advektion* und *Retardation* ein.

1.5.1 Herleitung

Die Herleitung der Transportgleichung für einen nicht reaktiven Wasserinhaltsstoff erfolgt durch die Massenbilanz über ein repräsentatives Kontrollvolumen [RSW02]. Über die Oberfläche des Volumens werden die im vorherigen Kapitel eingeführten advektiven, diffusiven und dispersiven Flüsse j_{adv}, j_{diff} und j_{disp} aufsummiert. Dabei gewährleistet das Prinzip der Massenerhaltung, dass die Änderung der Flüsse über die Oberfläche des Kontrollvolumens zu einer Speicherung im Kontrollvolumen führen, oder durch Quellen und Senken ausgeglichen wird. Die Terme der Gleichung sind auf die effektive Porosität bezogen. Die Retardierung fließt als Steuerterm der Speicherung in die Zeitableitung mit ein.

Unter Berücksichtigung der Zeitabhängigkeit der Konzentrationsänderung mit Retardierung, dem sogenannten Speicherterm

$$n_f \frac{\partial c}{\partial t} R \tag{1.12}$$

sowie der Quell- und Senkterme für den Stoffeintrag

$$n_f \sigma \tag{1.13}$$

ergibt sich folgende Formulierung:

$$n_f \frac{\partial c}{\partial t} R = -\boldsymbol{\nabla} \cdot j_{adv} + \boldsymbol{\nabla} \cdot j_{diff} + \boldsymbol{\nabla} \cdot j_{disp} + n_f \sigma.$$

Nach dem Einsetzen der Massenflüsse erhält man:

$$n_f \frac{\partial c}{\partial t} R = -\boldsymbol{\nabla} \cdot (\mathbf{u} c n_f) + \boldsymbol{\nabla}(\mathbf{D}_{mol} n_f \boldsymbol{\nabla} c) + \boldsymbol{\nabla}(\mathbf{D}_{disp} n_f \boldsymbol{\nabla} c) + \sigma n_f.$$

Aufgrund der Produktregel ergibt sich aus

$$\boldsymbol{\nabla} \cdot (\mathbf{u} c n_f) = c \boldsymbol{\nabla} \cdot (\mathbf{u} n_f) + \mathbf{u} \cdot \boldsymbol{\nabla}(c n_f),$$

der Stoffeintrag durch Wasserzustrom. Die Ableitung der Abstandsgeschwindigkeit beschreibt eine Änderung des advektiven Massenflusses und kann somit als zusätzlicher Quellterm aufgefasst werden:

$$c \boldsymbol{\nabla} \cdot (\mathbf{u} n_f) = q(c - c_{(in)}) \tag{1.14}$$

q = flächenbezogener Abfluss (meist der Quellterm des Strömungsmodells) $[\frac{1}{s}]$,

$c_{(in)}$ = Stoffkonzentration im zugegebenen Wasser $[\frac{kg}{m^3}]$.

Nach Vereinfachung und der Division durch die effektive Porosität ergibt sich:

$$\frac{\partial c}{\partial t} \cdot R = -\mathbf{u} \cdot \boldsymbol{\nabla} c + \boldsymbol{\nabla}(\mathbf{D}^* \boldsymbol{\nabla} c) + \sigma - \frac{q}{n_f}(c - c_{(in)}). \tag{1.15}$$

Die Gleichung (7.1) bildet die Grundlage der numerischen Berechnung des Stofftransportes in zwei und drei Dimensionen.

1.5.2 Merkmale der Transportgleichung

Die Transportgleichung (7.1) ist eine partielle Differentialgleichung zweiter Ordnung. Sie setzt sich zusammen aus einer Diffusionsgleichung vom parabolischem Typ

$$\frac{\partial c}{\partial t} \cdot R = \boldsymbol{\nabla}(\mathbf{D}^* \boldsymbol{\nabla} c) + \sigma, \tag{1.16}$$

die ähnlich der Strömungsgleichung ist, und einer Advektionsgleichung vom hyperbolischem Typ

$$\frac{\partial c}{\partial t} \cdot R = -\mathbf{u} \cdot \boldsymbol{\nabla} c + \sigma - \frac{q}{n_f}(c - c_{(in)}). \tag{1.17}$$

Je nachdem welcher Term überwiegt, zeigt die Transportgleichung ihren parabolischen oder hyperbolischen Charakter. Der hyperbolische Term macht die Lösung dieser Gleichung schwierig und wesentlich aufwendiger als die der Strömungsgleichung. Ein Maß für die Charakteristik der Transportgleichung ist die *Peclet Zahl*. Sie gibt das Verhältnis zwischen advektiven und dispersiven Transportanteil wider, vergleiche [RSW02] S.19.

1.5.3 Lösungsvoraussetzungen der Transportgleichung

Zur Lösung der Transportgleichung benötigt man in der Regel drei Gruppen von Daten. Dies sind

- Anfangsbedingungen
  Anfangskonzentrationen an gelösten Stoffen im Grundwasser.
- Randbedingungen
  Angaben von Förder-/Infiltrationsbrunnen, Konzentrationszuflüssen usw..
- Physikalische Parameter der Differentialgleichung, Konstanten und Variablen. Zum Beispiel die Dispersivitätslängen, die effektive Porosität usw. .

Um die Lösbarkeit zu gewährleisten, muss man darauf achten, physikalisch korrekte Parameter sowie sinnvolle Anfangsbedingungen und Randbedingungen anzugeben. Desweiteren spielen eine Reihe von Stabilitätskriterien ein wichtige Rolle, auf die später eingegangen wird.

Randbedingungen

DIRICHLET **Randbedingungen** Es wird eine feste Stoffkonzentration auf dem Rand vorgegeben. Dies ist sinnvoll an Zustromrändern, um einen Schadstoffeintrag nachzubilden. Wenn man den Stoffzustrom auf Null setzt, strömt unkontaminiertes Wasser in das Modellgebiet. An einem Zustromrand sollte die Konzentration immer angegeben werden.

NEUMANN **Randbedingungen** Diese geben den diffusiv/dispersiven Stofffluss in Normalenrichtung zum Rand an. Mit dieser Randbedingung kann ein kontinuierlicher Stofffluss über den Rand in das Modellgebiet definiert werden. Falls diese Randbedingung im Modell nicht explizit angegeben werden, sind diese am Rand Null. Das bedeutet, dass es keinen Stofftransport über den Rand hinaus gibt. Solche undurchlässigen Ränder werden auch als natürliche NEUMANN Randbedingungen bezeichnet.

CAUCHY **Randbedingungen** Sie spezifizieren eine Linearkombination aus der Konzentration und der Normalenableitung auf dem Gebietsrand. Aufgrund der hohen Komplexität dieser Randbedingung wurde sie nicht implementiert.

Transmissionsrandbedingungen Diese Randbedingungen werden eingesetzt, wenn es einen Konzentrationsstrom aus dem Gebiet über den Rand gibt. Sie sind besonders an Abstromrändern von Bedeutung und können als variable DIRICHLET Randbedingungen mit advektivem Term betrachtet werden.

2 Numerische Grundlagen

Für die Grundwasser- und Transportmodellierung haben sich verschiedene numerische Verfahren bewährt. Dies sind die Methode der Finiten Differenzen (FDM), die Methode der Finiten Elemente (FEM), vergleiche [LKW96] S.153ff, und die Methode der Finiten Volumen (FVM) [KA00].

2.1 Gebietsdiskretisierung

Grundlage der oben genannten Diskretisierungsverfahren ist die Zerlegung des Grundgebietes in diskrete Teilgebiete. Je nach gewähltem numerischen Verfahren stehen unterschiedliche Diskretisierungsmethoden zur Verfügung. Das Verfahren der Finiten Differenzen läßt sich nur auf strukturierten Gittern anwenden. Das Untersuchungsgebiet wird in rechteckige Gitter zerlegt, in denen die Nachbarschaften zwischen den Gitterpunkten klar definiert sind. Diese Diskretisierung ist am leichtesten zu implementieren. Die Finite Volumen und Finite Elemente Methoden funktionieren auf strukturierten und unstrukturierten Gittern. Dabei stellt das Finite Volumen Verfahen höhere Anforderungen an die Gittergenerierung als die FEM. Typische Elementformen von unstrukturierten Gittern der FVM und FEM sind Dreiecke, Vierecke, Tetraeder, Hexaeder, Prismen, Pyramiden teilweise auch mit gekrümmten Rändern [KA00]. Dabei ist zu beachten, dass die Theorie der FVM bisher nur auf Simplizialen[1] Gittern vollständig ist [Bey93]. Zur Gittererzeugung stehen verschiedene Methoden zu Verfügung wie: Belegungsmethoden z.B. die *Quadtree* und *Octtree* Zerlegungen, die Delaunay-Triangulierung, die Methoden der sukzessiven Gebietsreduktion und weitere, vergleiche [KA00].

2.2 Zeitdiskretisierung

Für die Diskretisierung der Zeit gibt es verschiedene Ansätze. Zum einen das explizite und implizite Euler-Verfahren sowie die Kombination aus beiden, das so genannte Crank-Nicolson-Verfahren. Das explizite Euler-Verfahren ist sehr einfach zu implementieren, da hier, im Gegensatz zum impliziten Euler-Verfahren, keine lineares

[1] Simplizis sind Punkte, Linien, Dreiecke und Tetraeder im 0,1,2 und 3 dimensionalen Raum

Gleichungssystem aufgestellt werden muß. Der Nachteil ist, dass eine Reihe von Stabilitätskriterien beachtet werden und die Zeitschritte meist sehr klein gewählt werden müssen, um unphysikalische[2] Ergebnisse zu vermeiden. Das implizite EULER-Verfahren hat diese Nachteile nicht, ist aber komplizierter zu implementieren. Beide Verfahren besitzen die Konvergenzordung 1. Das CRANK-NICOLSON-Verfahren kombiniert beide Methoden und erreicht die Konvergenzordnung 2, vergleiche [Hin03].

2.3 Grundlagen der Finiten Volumen Methode

Aufgrund ihrer besonderen Eigenschaften wurde die Methode der Finiten Volumen zur Diskretisierung der Strömungs- und Stofftransportgleichung gewählt. Sie vereint Vorteile der Finiten Differenzen und Finiten Elemente Verfahren. Mit ihr lassen sich Erhaltungsgleichungen, wie die Grundwasserströmungsgleichung, oder Transportgleichungen mit parabolischen und hyperbolischen Eigenschaften auf unstrukturierten Gittern lösen. Die Implementierung ist einfacher und physikalisch ersichtlicher als die der FEM. Die FVM wird meist als integrierte Finiten Differenzen Methode auf unstrukturierten Gittern verstanden [Hin03]. Die so genannte Zellen zentrierte Methode ist ein solcher Ansatz und wird auch hier für die Lösung der Strömungs- und Transportdifferentialgleichung verwendet.

Das *Zellen zentrierte* Finite Volumen Verfahren

Die Diskretisierung der Strömungs- und Transportgleichung basiert auf dem *Zellen zentrierten* Finite Volumen Verfahren. An einem Beispiel soll die theoretische Grundlage des Verfahrens erklärt werden.

Gegeben sei eine Differentialgleichung zweiter Ordnung, die einen Diffusionsprozess beschreibt, der Form

$$-\boldsymbol{\nabla} \cdot (k \boldsymbol{\nabla} h) = f \tag{2.1}$$

mit

$$k : \Omega \to \mathbb{R}^d, f : \Omega \to \mathbb{R}, d \in [1, 2, 3].$$

Mit $\mathbf{q}(h) = (k\boldsymbol{\nabla} h)$ erhält (2.1) folgende Form

$$\boldsymbol{\nabla} \cdot \mathbf{q}(h) = f. \tag{2.2}$$

Die Grundlage der FVM besteht in der Diskretisierung der Untersuchungsgebietes Ω in M Teilgebiete. Nach [KA00] muß dabei folgendes beachtet werden:

[2] Wie negative Konzentrationen oder unsinnige Gradienten.

1. jedes Ω_i ist offen, einfach zusammenhängend und polygonal berandet. Allerdings sind Schlitze auszuschließen.
2. $\Omega_i \cap \Omega_j = \emptyset$ für $i \neq j$
3. $\bigcup_{i=0}^{M} \Omega_i = \Omega$

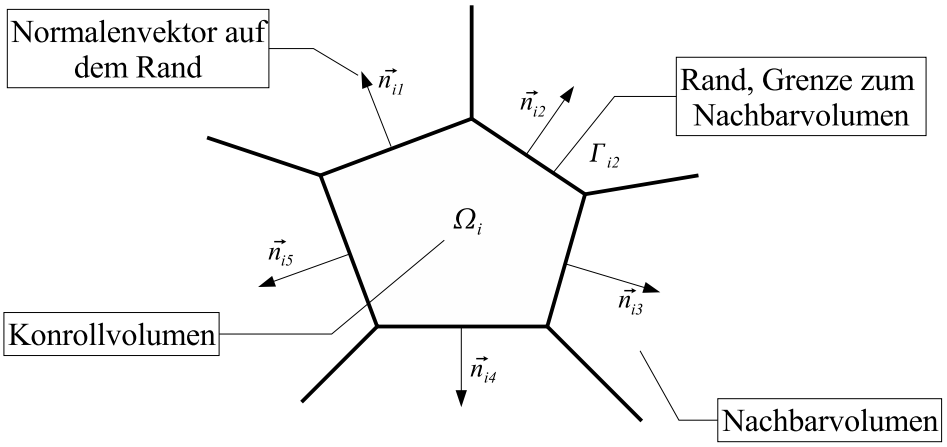


Abbildung 2.1: Kontrollvolumen mit Rand.

Diese Teilgebiete Ω_i werden Kontrollvolumen oder Kontrollgebiete genannt. Der nächste Schritt besteht darin, Gleichung (2.2) über jedes Kontrollvolumen zu integrieren und dann den Gaußschen Integralsatz anzuwenden. Dies ergibt

$$\int_{\partial\Omega_i} \boldsymbol{\nu} \cdot \mathbf{q}(h) d\sigma = \int_{\Omega_i} f dx, i \in \{1, ..., M\},$$

wobei $\boldsymbol{\nu}$ die äußere Einheitsnormale zu Ω_i ist. Da wegen der ersten Eigenschaft der Partition der Rand $\partial\Omega_i$ aus Geradenstücken $\Gamma_{ij} (j = 1, ..., n_i)$ besteht, entlang derer die Normale $\boldsymbol{\nu}|_{\Gamma_{ij}} =: \boldsymbol{\nu}_{ij}$ konstant ist, kann das Randintegral in eine entsprechende Summe zerlegt werden

$$\sum_{j=1}^{n_i} \int_{\Gamma_{ij}} \boldsymbol{\nu}_{ij} \cdot \mathbf{q}(h) d\sigma = \int_{\Omega_i} f dx \tag{2.3}$$

Die in (2.3) auftretenden Integrale können auf verschiedene Weise approximiert werden. Dies führt zu unterschiedlichen Ansätzen und somit zu unterschiedlichen Finiten Volumen Methoden. Die hier verwendete Methode ist die *Zellen zentrierte* Finite Volumen Methode, wobei sich der Funktionswert im Schwerpunkt der Partition Ω_i befindet.

Durch die Rücksubstitution $\mathbf{q}(h) = k\boldsymbol{\nabla} h$ ergibt sich aus (2.3) folgende Gleichung

$$\sum_{j=1}^{n_i} \int_{\Gamma_{ij}} \boldsymbol{\nu}_{ij} \cdot (k\boldsymbol{\nabla} h) d\sigma = \int_{\Omega_i} f dx \tag{2.4}$$

Im nächsten Schritt werden die Koeffizienten $k := \mu_{ij}$ auf den Rändern Γ_{ij} approximiert.

Die Normalenableitung $\boldsymbol{\nu}_{ij}\boldsymbol{\nabla}h$ wird nach [KA00] durch

$$\frac{h_i - h_j}{d_{ij}}$$

ersetzt. Die Integrale $\int_{\Gamma_{ij}}$ und $\int_{\Omega_i}$ werden durch durch m_{ij} und m_i abgeschätzt. Der Abstand zwischen zwei Schwerpunkten wird durch d_{ij} abgeschätzt, wobei zu beachten ist, dass die Berechnung von d_{ij} eine Diskretisierung des Untersuchungsgebietes durch Voronoi Polygone voraussetzt. Somit sind die Schwerpunkte zweier Polygone durch eine gerade Linie verbunden die senkrecht auf dem Rand steht. Daraus ergibt sich folgendes lineares Gleichungssystem

$$\sum_{j=1}^{n_i} m_{ij}\mu_{ij}\frac{h_i - h_j}{d_{ij}} = f_i m_i, \tag{2.5}$$

bzw. umgeschrieben

$$\sum_{j=1}^{n_i} \mu_{ij}(h_i - h_j)\frac{m_{ij}}{d_{ij}} = f_i m_i. \tag{2.6}$$

3 Anwendung der numerischen Verfahren

Die im vorherigen Kapitel vorgestellten numerischen Verfahren, werden auf die im Kapitel *Grundlagen der Strömungs- und Stofftransportmodellierung* hergeleiteten Differentialgleichungen angewendet. Die in diesem Kapitel erarbeitete Finite Volumen Diskretisierung, bildet die Grundlage für die Implementierung des numerischen Strömungs- und Transportmodells in GRASS und kann auf zwei und drei dimensionale Probleme angewendet werden.

3.1 Gebietsdiskretisierung

Es wird für die Berechnung der Strömungs- und Transportdifferentialgleichung eine Diskretisierung durch strukturierte Gitter gewählt, welche das Kontrollgebiet in rechtwinklige Vierecke, im zweidimensionalen Fall, oder rechtwinklige Hexaeder im dreidimensionalen Fall, zerlegt. Grundlage für die Diskretisierung bildet das GRASS Raster und Volumen Datenbankformat, welches Daten im zweidimensionalen und dreidimensionalen (Voxel) Pixelformat verwaltet. Bei gespannten Grundwasserverhältnissen sind die Raster und Voxeldaten aus GRASS ausreichend, um eine dreidimensionale Gebietsgeometrie zu beschreiben. Im Falle zweidimensionaler ungespannter Grundwasserverhältnisse, werden die Höheninformationen für jeden Pixel über die piezometrische Druckhöhe oder eine obere Grenze ermittelt, sodass rechtwinklige Hexaeder entstehen. Dieses Vorgehen bricht mit den Kriterien der Gebietsdiskretisierung. Die Flächen in X und Y Richtung der Hexaeder fallen nicht mehr exakt aufeinander. Der dabei entstehende numrische Fehler wird hier vernachlässigt, da das gekoppelte Strömungs- und Transportmodell in dieser Arbeit, nur auf gespannte Grundwasserleiter mit konformer Gebietszerlegeung angewendet wird.

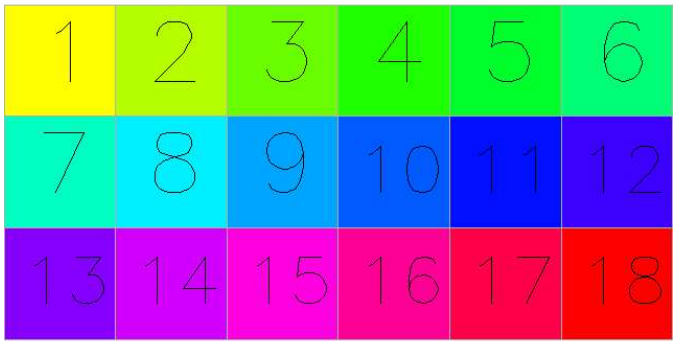
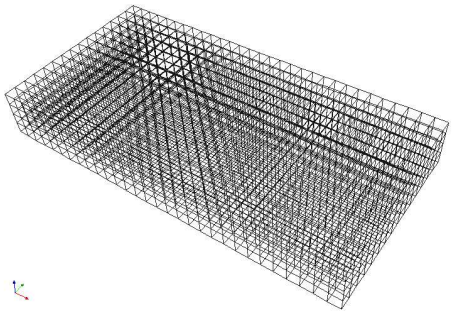
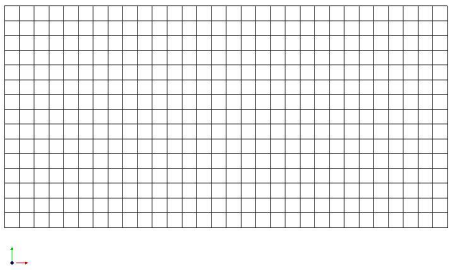


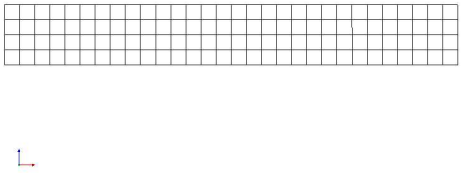
Abbildung 3.1: Das GRASS Rasterformat mit Nummerierung.



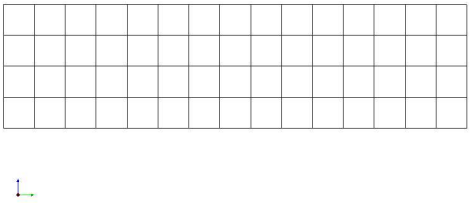
(a) perspektivische Ansicht



(b) Ansicht xy-Ebene



(c) Ansicht xz-Ebene



(d) Ansicht yz-Ebene

Abbildung 3.2: Gebietsdiskretisierung in rechteckige Hexaeder

Durch die Zerlegung des Untersuchungsgebietes $\Omega \subset \mathbb{R}^3$ in rechtwinklige Hexaeder, den Kontrollvolumen Ω_i, lassen sich die so erzeugten Volumen durch ihre Länge Δx, Breite Δy und Höhe Δz beschreiben. In Abbildung 3.2 ist eine Beispieldiskretisierung aus GRASS mit $30 \times 15 \times 4 = 1800$ Volumen dargestellt.

Weiterhin sei $i \in \overline{\Lambda}$ die Indexmenge aller Kontrollvolumen. Wir bezeichnen mit $\partial\Omega_i$ die Ränder eines Volumens Ω_i. Weiterhin seien definiert:

$\Lambda_i := \{j \in \overline{\Lambda} : \partial\Omega_i \cap \partial\Omega_j \neq \emptyset\}$ für die Menge der Indizes benachbarter Kontrollvolumen

$\Gamma_{ij} := \partial\Omega_i \cap \partial\Omega_j, j \in \Lambda_i$
für die gemeinsamen Fläche der Ränder benachbarter Kontrollvolumen Ω_i und Ω_j
.

3.2 Zeitdiskretisierung

Für die Diskretisierung der Zeit wird ein implizites Euler Verfahren gewählt, da dieses Zeitschritte beliebiger Größe zuläßt. Dieses Verfahren ist von erster Fehlerordnung. Deshalb wird mit beliebig großen Zeitschritten auch der dadurch entstehende Fehler beliebig groß. Eine bessere zeitliche Diskretisierung stellt das Crank-Nicolson-Verfahren dar, welches eine quadratische Fehlerordnung hat. Aufgrund des höheren numerischen Aufwands wird es hier nicht weiter betrachtet.

3.2.1 Zeitliche Diskretisierung der Strömungs- und Transportgleichung

Die Zeitableitung in der Gleichung (1.8) wird bei einer impliziten Euler Zeitdiskretisierung durch einen Vorwärts-Differenzenquotient approximiert

$$\frac{\partial h}{\partial t} = \frac{h^{(t+\Delta t)} - h^{(t)}}{\Delta t}. \tag{3.1}$$

Gesucht ist die piezometrische Druckhöhe h zum neuen Zeitpunkt $(t + \Delta t)$.

Die zeitliche Diskretisierung der Transportgleichung ist

$$\frac{\partial c}{\partial t} = \frac{c^{(t+\Delta t)} - c^{(t)}}{\Delta t}. \tag{3.2}$$

Gesucht ist die Konzentration c zum neuen Zeitpunkt $(t + \Delta t)$.

3.3 Diskretisierung der Strömungsgleichung

3.3.1 Finite Volumen Formulierung

Im Fall der *Zellen zentrierten* FVM liegt die Problemvariable im Schwerpunkt des Kontrollvolumens. Dieser Wert ist repräsentativ für das gesamte Kontrollvolumen. Zur Approximation der partiellen Ableitungen werden Differenzenquotienten benutzt. Die gewählte FVM Formulierung ist exakt auf homogenen, linearen, rechtwinkligen Vierecks- und Hexaedergittern. Homogen bedeutet, daß die Abstände in der jeweiligen Raumrichtung gleich sind. Dies entspricht der Gebietsdiskretisierung von GRASS. Aufgrund dieser Formulierung können Programme entwickelt werden, die die Strömungsgleichung in zwei oder drei Raumdimensionen lösen.

Mit der impliziten Zeitdiskretisierung läßt sich Gleichung (1.8) wie folgt formulieren

$$\boldsymbol{\nabla} \cdot \mathbf{K}_f \cdot \boldsymbol{\nabla} h^{(t+\Delta t)} + q = S_s \frac{h^{(t+\Delta t)} - h^{(t)}}{\Delta t}, \tag{3.3}$$

mit

$$b = S_s \frac{h^{(t)}}{\Delta t} - q$$

und

$$r = \frac{S_s}{\Delta t}$$

ergibt sich

$$\boldsymbol{\nabla} \cdot \mathbf{K}_f \cdot \boldsymbol{\nabla} h^{(t+\Delta t)} - r h^{(t+\Delta t)} = b. \tag{3.4}$$

Desweiteren sei β die Anzahl der zu betrachtenen kartesischen Raumkomponenten. Für zwei Dimensionen ergibt sich $\beta \in x, y$, für drei Dimensionen $\beta \in x, y, z$.

Nun wird Gleichung (3.4) über jedes Kontrollvolumen integriert.

$$\int_{\Omega_i} \boldsymbol{\nabla} \cdot \mathbf{K}_f \cdot \boldsymbol{\nabla} h^{(t+\Delta t)} - r h^{(t+\Delta t)} dx = \int_{\Omega_i} b dx. \tag{3.5}$$

Durch der Anwendung des Gaußschen Integralsatzes ergibt sich

$$\int_{\partial\Omega_i} \boldsymbol{\nu}^T \cdot \mathbf{K}_f \cdot \boldsymbol{\nabla} h^{(t+\Delta t)} d\sigma - \int_{\Omega_i} r h^{(t+\Delta t)} dx = \int_{\Omega_i} b dx, \tag{3.6}$$

wobei $\boldsymbol{\nu}^T$ den äußeren Einheitsnormalenvektor zu $\partial\Omega_i$ bezeichnet. Der Rand $\partial\Omega_i$ besteht aus ebenen Flächenstücken $\Gamma_{ij}, j \in \Lambda_i$. Auf diesen Flächen ist die Normale $\boldsymbol{\nu}_{ij}^T$ konstant, siehe Abbildung 3.3. Folgende Normalenvektoren treten auf:

$$\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} -1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ -1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix}. \tag{3.7}$$

Somit kann das Randintegral in eine Summe zerlegt werden:

$$\sum_{j\in\Lambda_i}\int_{\Gamma_{ij}} \boldsymbol{\nu}_{ij}^T\cdot\mathbf{K}_f\cdot\boldsymbol{\nabla}h^{(t+\Delta t)}d\sigma+\int_{\Omega_i} rh^{(t+\Delta t)}dx=\int_{\Omega_i} bdx. \tag{3.8}$$

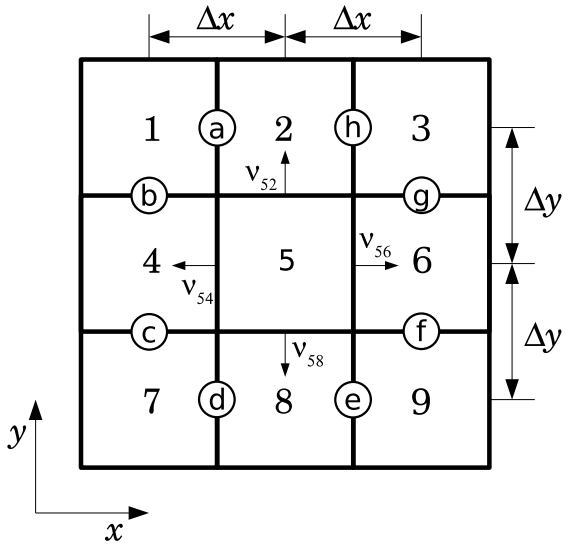


Abbildung 3.3: Neun-Punkt-Stern Diskretisierung mit Flächennormalen und Indizierung. Die Nummern geben die gesuchten piezometrischen Druckhöhen im Schwerpunkt der jeweiligen Zelle an. Die Buchstaben stellen die Druckhöhen auf den Mittelpunkten der Seitenflächen dar.

Allgemeine Formulierung

In einem inhomogenen, anisotropen Grundwasserleiter ist der Durchlässigkeitstensor $\mathbf{K}_f$ voll besetzt. In kartesischen Koordinaten ergibt sich

$$\begin{aligned}\boldsymbol{\nu}_{ij}^T\cdot\mathbf{K}_f\cdot\boldsymbol{\nabla}h &= \begin{pmatrix}\nu_{ij}^x\\ \nu_{ij}^y\\ \nu_{ij}^z\end{pmatrix}^T\begin{pmatrix}k^{xx}&k^{xy}&k^{xz}\\ k^{yx}&k^{yy}&k^{yz}\\ k^{zx}&k^{zy}&k^{zz}\end{pmatrix}\begin{pmatrix}\frac{\partial h}{\partial x}\\ \frac{\partial h}{\partial y}\\ \frac{\partial h}{\partial z}\end{pmatrix}\\ &= \nu_{ij}^x(k^{xx}\frac{\partial h}{\partial x}+k^{xy}\frac{\partial h}{\partial y}+k^{xz}\frac{\partial h}{\partial z})+\nu_{ij}^y(k^{yx}\frac{\partial h}{\partial x}+k^{yy}\frac{\partial h}{\partial y}+k^{yz}\frac{\partial h}{\partial z})+\\ &\qquad \nu_{ij}^z(k^{zx}\frac{\partial h}{\partial x}+k^{zy}\frac{\partial h}{\partial y}+k^{zz}\frac{\partial h}{\partial z}).\end{aligned} \tag{3.9}$$

Die partiellen Ableitungen werden durch Differenzenquotienten angenähert. Im folgenden sollen, basierend auf Abbildung 3.3, die Differenzenquotienten in zwei Dimensionen hergeleitet werden.

In x-Richtung ergeben sich zwei Differenzenquotienten:

$$\nu_{54}^{x}\left(k^{xx}\frac{\partial h}{\partial x}+k^{xy}\frac{\partial h}{\partial y}\right)=\nu_{54}^{x}k^{xx}\frac{(h_5^{(t+\Delta t)}-h_4^{(t+\Delta t)})}{\Delta x}+\nu_{54}^{x}k^{xy}\frac{(h_d^{(t+\Delta t)}-h_a^{(t+\Delta t)})}{2\Delta y} \quad (3.10)$$

und

$$\nu_{56}^{x}\left(k^{xx}\frac{\partial h}{\partial x}+k^{xy}\frac{\partial h}{\partial y}\right)=\nu_{56}^{x}k^{xx}\frac{(h_6^{(t+\Delta t)}-h_5^{(t+\Delta t)})}{\Delta x}+\nu_{56}^{x}k^{xy}\frac{(h_e^{(t+\Delta t)}-h_h^{(t+\Delta t)})}{2\Delta y}\ . \quad (3.11)$$

In y-Richtung ebenfalls:

$$\nu_{52}^{y}\left(k^{yx}\frac{\partial h}{\partial x}+k^{yy}\frac{\partial h}{\partial y}\right)=\nu_{52}^{y}k^{yx}\frac{(h_g^{(t+\Delta t)}-h_b^{(t+\Delta t)})}{2\Delta x}+\nu_{52}^{y}k^{yy}\frac{(h_2^{(t+\Delta t)}-h_5^{(t+\Delta t)})}{\Delta y} \quad (3.12)$$

und

$$\nu_{58}^{y}\left(k^{yx}\frac{\partial h}{\partial x}+k^{yy}\frac{\partial h}{\partial y}\right)=\nu_{58}^{y}k^{yx}\frac{(h_e^{(t+\Delta t)}-h_d^{(t+\Delta t)})}{2\Delta x}+\nu_{58}^{y}k^{yy}\frac{(h_5^{(t+\Delta t)}-h_8^{(t+\Delta t)})}{\Delta y}\ . \quad (3.13)$$

Zur Approximation der Tensoreinträge auf dem jeweiligen Rand wird zwischen benachbarten Zellen das harmonische Mittel verwendet. Das harmonische Mittel gewährleistet, daß bei Tensoreinträgen von Null kein Fluß über den Rand erfolgt. Auch werden Grenzen zwischen gut und schlecht durchlässigen Schichten besser approximiert als durch das arithmetische Mittel. Die Berechnung des harmonischen Mittels von Tensorteinträgen wird in Abschnitt 3.4.1 näher beschrieben.

Sei $i \in \{5\}$ der Index des zentralen Elements und $j \in \{2,4,6,8\}$ der Index aller umliegenden Elemente. Weiterhin sei $m_{ij}^{l} \in \{a,f,g,h\}$ und $n_{ij}^{l} \in \{b,c,d,e\}$ die Indizes der piezometrischen Druckhöhe auf dem Rand transversal zur Strecke ij. Die Indizes m_{ij}^{l} und n_{ij}^{l} sind von i und j sowie der Raumrichtung/Orientierung l abhängig.

Weiterhin sei

$$\overline{\delta_{kl}}=\begin{cases}0 & \text{falls } k=l\\ 1 & \text{falls } k\neq l\end{cases}\ .$$

Somit ergibt sich, unter Berücksichtigung des Vorzeichens der Komponenten der Flächennormalen, folgende Formulierung:

$$\boldsymbol{\nu}_{ij}^{T}\cdot\mathbf{K}_f\cdot\boldsymbol{\nabla}h=(h_j^{(t+\Delta t)}-h_i^{(t+\Delta t)})\left(\sum_{k\in\beta}|\nu_{ij}^{k}|\frac{k_{ij}^{kk}}{d_{ij}}\right)+\left(\sum_{k\in\beta}|\nu_{ij}^{k}|\sum_{l\in\beta}\frac{k_{ij}^{kl}}{d_{m_{ij}^{l}n_{ij}^{l}}}\overline{\delta_{kl}}(h_{m_{ij}^{l}}^{(t+\Delta t)}-h_{n_{ij}^{l}}^{(t+\Delta t)})\right)\ . \quad (3.14)$$

wobei k und l die drei kartesischen Raumrichtungen x,y und z repräsentieren, d_{ij} ist der Abstand der Schwerpunkte von $h_i^{(t+\Delta t)}$ und $h_j^{(t+\Delta t)}$. Der Abstand der Seitenmittelpunkte von $h_{m_{ij}^{l}}^{(t+\Delta t)}$ und $h_{n_{ij}^{l}}^{(t+\Delta t)}$ ist durch $d_{m_{ij}^{l}n_{ij}^{l}}$ gegeben. Die Komponenten k_{ij}^{kl} sind die harmonisch gemittelten Tensoreinträge auf dem Rand Γ_{ij}. Der Normalenvektor $\boldsymbol{\nu}_{ij}=(\nu_{ij}^{x},\nu_{ij}^{y},\nu_{ij}^{z})^{T}$ wird durch seine Raumkomponenten ν_{ij}^{k} beschrieben.

Die Berechnung von $h_{m_{ij}^l}^{(t+\Delta t)}$ und $h_{n_{ij}^l}^{(t+\Delta t)}$ auf dem Rand kann durch das arithmetische Mittel der anliegenden Zellen erfolgen. Eingesetzt in (3.8) ergibt sich

$$\sum_{j\in\Lambda_i}\int_{\Gamma_{ij}}(h_j^{(t+\Delta t)}-h_i^{(t+\Delta t)})\left(\sum_{k\in\beta}|\nu_{ij}^k|\frac{k_{ij}^{kk}}{d_{ij}}\right)+\left(\sum_{k\in\beta}|\nu_{ij}^k|\sum_{l\in\beta}\frac{k_{ij}^{kl}}{d_{m_{ij}^l n_{ij}^l}}\overline{\delta_{kl}}(h_{m_{ij}^l}^{(t+\Delta t)}-h_{n_{ij}^l}^{(t+\Delta t)})\right)d\sigma + \int_{\Omega_i} r_i h_i^{(t+\Delta t)}dx = \int_{\Omega_i} b_i dx. \quad (3.15)$$

Aus Gleichung (3.4) ergibt sich also mit der FVM

$$\sum_{j\in\Lambda_i}(h_j^{(t+\Delta t)}-h_i^{(t+\Delta t)})\left(\sum_{k\in\beta}|\nu_{ij}^k|\frac{k_{ij}^{kk}}{d_{ij}}\right)+\left(\sum_{k\in\beta}|\nu_{ij}^k|\sum_{l\in\beta}\frac{k_{ij}^{kl}}{d_{m_{ij}^l n_{ij}^l}}\overline{\delta_{kl}}(h_{m_{ij}^l}^{(t+\Delta t)}-h_{n_{ij}^l}^{(t+\Delta t)})\right)m_{ij} + m_i r_i h_i^{(t+\Delta t)} = m_i b_i \quad (3.16)$$

wobei die Flächeninhalte m_{ij} und das Volumen m_i entsprechend der verwendeten Kontrollvolumen errechnet werden müssen.

Vereinfachungen

Es wird im Weiteren davon ausgegangen, dass der Durchlässigkeitstensor $\mathbf{K}_f$ nur auf der Diagonalen besetzt ist und aufgrund dessen die Anisotropien im Grundwasserleiter am kartesischen Koordinatensystem ausgerichtet sind.

$$\boldsymbol{\nu}_{ij}^T\cdot\mathbf{K}_f\cdot\boldsymbol{\nabla}h = \begin{pmatrix}\nu_{ij}^x\\ \nu_{ij}^y\\ \nu_{ij}^z\end{pmatrix}\begin{pmatrix}k^{xx}&0&0\\0&k^{yy}&0\\0&0&k^{zz}\end{pmatrix}\begin{pmatrix}\frac{\partial h}{\partial x}\\ \frac{\partial h}{\partial y}\\ \frac{\partial h}{\partial z}\end{pmatrix} = \nu_{ij}^x k^{xx}\frac{\partial h}{\partial x}+\nu_{ij}^y k^{yy}\frac{\partial h}{\partial y}+\nu_{ij}^z k^{zz}\frac{\partial h}{\partial z}. \quad (3.17)$$

Folgende Differenzenquotienten ergeben sich basierend auf Abbildung 3.4 in zwei Dimensionen:

In x-Richtung ergeben sich zwei Differenzenquotienten:

$$\nu_{02}^x k^{xx}\frac{\partial h}{\partial x} = \nu_{02}^x k^{xx}\frac{(h_0^{(t+\Delta t)}-h_2^{(t+\Delta t)})}{\Delta x} \quad (3.18)$$

und

$$\nu_{04}^x k^{xx}\frac{\partial h}{\partial x} = \nu_{04}^x k^{xx}\frac{(h_4^{(t+\Delta t)}-h_0^{(t+\Delta t)})}{\Delta x}\ . \quad (3.19)$$

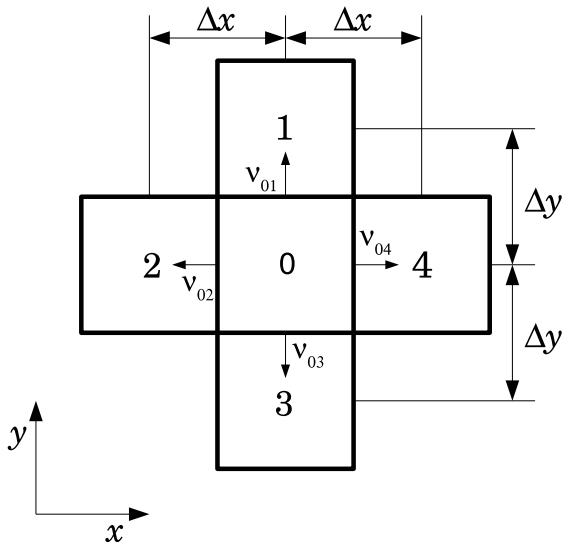


Abbildung 3.4: Fünf-Punkt-Stern Diskretisierung mit Flächennormalen und Indizierung. Die Nummern geben die gesuchten piezometrischen Druckhöhen im Schwerpunkt der jeweiligen Zelle an.

In y-Richtung ebenfalls:

$$\nu_{01}^{y} k^{yy} \frac{\partial h}{\partial y} = \nu_{01}^{x} k^{yy} \frac{(h_1^{(t+\Delta t)} - h_0^{(t+\Delta t)})}{\Delta y} \tag{3.20}$$

und

$$\nu_{03}^{y} k^{yy} \frac{\partial h}{\partial y} = \nu_{03}^{x} k^{yy} \frac{(h_0^{(t+\Delta t)} - h_3^{(t+\Delta t)})}{\Delta y} \; . \tag{3.21}$$

Sei $i \in \{0\}$ der Index des zentralen Elements und $j \in \{1, 2, 3, 4\}$ der Index aller umliegenden Elemente dann ergibt sich, unter Berücksichtigung des Vorzeichens der Komponenten der Flächennormalen, folgende Formulierung:

$$\boldsymbol{\nu}_{ij}^{T} \cdot \mathbf{K}_f \cdot \boldsymbol{\nabla} h = (h_j^{(t+\Delta t)} - h_i^{(t+\Delta t)}) \left(\sum_{k \in \beta} |\nu_{ij}^{k}| \frac{k^{kk}}{d_{ij}} \right) . \tag{3.22}$$

Eingesetzt in die allgemeine Finite Volumen Formulieren (3.16) und Berücksichtigung der erfolgten Approximationen ergibt

$$\sum_{j \in \Lambda_i} \int_{\Gamma_{ij}} (h_j^{(t+\Delta t)} - h_i^{(t+\Delta t)}) \left(\sum_{k \in \beta} |\nu_{ij}^{k}| \frac{k_{ij}^{kk}}{d_{ij}} \right) d\sigma + \int_{\Omega_i} r_i h_i^{(t+\Delta t)} dx = \int_{\Omega_i} b_i dx. \tag{3.23}$$

Für die Berechnung der Flächen und Volumenintegrale wird folgender Ansatz gemacht: Da die Kontrollvolumen aus rechtwinkligen Hexaedern bestehen, kann das Volumen m_i eines einzelnen Kontrollvolumens Ω_i durch $\Delta x_i \Delta y_i \Delta z_i$ exakt berechnet werden.

Für die Flächenintegrale können die Flächen m_{ij} abhängig von der Orientierung durch $0,5(\Delta y_i \Delta z_i + \Delta y_j \Delta z_j)$, $0,5(\Delta x_i \Delta z_i + \Delta x_j \Delta z_j)$ oder $0,5(\Delta x_i \Delta y_i + \Delta x_j \Delta y_j)$ errechnet werden. Durch die Vereinfachung ergeben sich folgende Beziehungen zwischen den Vektoren der Kontrollvolumenflächen und den Normalenvektoren der 6 Raumrichtungen.

$$m_{ij} = 0,5 \left| \begin{pmatrix} (\Delta y_i \Delta z_i + \Delta y_j \Delta z_j) \\ (\Delta x_i \Delta z_i + \Delta x_j \Delta z_j) \\ (\Delta x_i \Delta y_i + \Delta x_j \Delta y_j) \end{pmatrix} \cdot \begin{pmatrix} \nu_{ij}^x \\ \nu_{ij}^y \\ \nu_{ij}^z \end{pmatrix} \right|. \tag{3.24}$$

Eingesetzt ergibt sich das Gleichungssystem

$$\sum_{j \in \Lambda_i} (h_j^{(t+\Delta t)} - h_i^{(t+\Delta t)}) \left(\sum_{k \in \beta} |\nu_{ij}^k| \frac{k_{ij}^{kk}}{d_{ij}} \right) m_{ij} + m_i r_i h_i^{(t+\Delta t)} = m_i b_i. \tag{3.25}$$

3.3.2 Aufstellung der Steifigkeitsmatrix und Massenmatrix

Für die Aufstellung des Gleichungssystems werden die Terme r_i und b_i aus Gleichung (3.4) zurück substituiert:

$$\sum_{j \in \Lambda_i} (h_j^{(t+\Delta t)} - h_i^{(t+\Delta t)}) \left(\sum_{k \in \beta} |\nu_{ij}^k| \frac{k_{ij}^{kk}}{d_{ij}} \right) m_{ij} + m_i S_i \frac{h_i^{(t+\Delta t)}}{\Delta t} = m_i S_i \frac{h_i^{(t)}}{\Delta t} + m_i q_i \tag{3.26}$$

oder

$$\begin{aligned} &\sum_{j \in \Lambda_i} h_j^{(t+\Delta t)} \left(\sum_{k \in \beta} |\nu_{ij}^k| \frac{k_{ij}^{kk}}{d_{ij}} \right) m_{ij} \\ &\qquad - \left(\sum_{j \in \Lambda_i} \left(\sum_{k \in \beta} |\nu_{ij}^k| \frac{k_{ij}^{kk}}{d_{ij}} \right) m_{ij} + m_i \frac{S_i}{\Delta t} \right) h_i^{(t+\Delta t)} = m_i S_i \frac{h_i(t)}{\Delta t} + m_i q_i. \end{aligned} \tag{3.27}$$

Die Steifigkeitsmatrix $\mathbf{S}$ bildet sich aus

$$m_{ij} \left(\sum_{k \in \beta} |\nu_{ij}^k| \frac{k_{ij}^{kk}}{d_{ij}} \right) \tag{3.28}$$

auf den Nebendiagonalen und

$$- \sum_{j \in \Lambda_i} \left(\sum_{k \in \beta} |\nu_{ij}^k| \sum_{l \in \beta} \frac{k_{ij}^{kk}}{d_{ij}} \right) m_{ij} \tag{3.29}$$

für auf der Diagonalen.

Die Einträge der Massenmatrix $\mathbf{M}$ werden durch

$$\mathbf{M} = -m_i \frac{S_i}{\Delta t} \tag{3.30}$$

gebildet. Die Einträge in den Lastvektor $\mathbf{b}$ ergeben sich durch

$$\mathbf{b} = m_i S_i \frac{h_i^{(t)}}{\Delta t} + m_i q_i. \tag{3.31}$$

Das lineare Gleichungssystem lautet somit

$$(\mathbf{M} + \mathbf{S})\mathbf{x} = \mathbf{b}. \tag{3.32}$$

Die Steifigkeitsmatrix $\mathbf{S}$ und die Massenmatrix $\mathbf{M}$ ergeben zusammen die Systemmatrix $\mathbf{A}$ des Gleichungssystems $\mathbf{Ax} = \mathbf{b}$. Es ist schwach besetzt, symmetrisch und positiv definit.

3.3.3 Einbau der Randbedingungen

Dirichlet Bedingungen

Diese Randbedingungen geben die Lösung der Differentialgleichung auf dem Rand vor. Die Dirichlet Randbedingungen können auf zwei unterschiedliche Arten eingebaut werden.

Die erste Möglichkeit ist, die Randbedingungen in die Massenmatrix, Steifigkeitsmatrix und den Lastvektor zu integrieren. Dabei werden die Zeilen und Spalten der Matrix $\mathbf{A} = \mathbf{M} + \mathbf{S}$, in denen das Diagonalelement eine Dirichletsche Randbedingung enthält, ausgenommen dem Diagonalwert, auf Null gesetzt. Der Diagonalwert wird auf eins gesetzt. Dann müssen die entsprechenden Dirichletwerte, multipliziert mit dem entsprechenden Spaltenelement, vom Vektor $\mathbf{b}$ abgezogen werden und der Eintrag im Vektor, der dem Diagonalelement eins entspricht, auf den entsprechenden Wert der piezometrischen Druckhöhe gesetzt werden.

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & x & x & 0 & x & 0 & 0 & 0 & 0 \\ 0 & x & x & 0 & 0 & x & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & x & 0 & 0 & x & x & 0 & x & 0 \\ 0 & 0 & x & 0 & x & x & 0 & 0 & x \\ 0 & 0 & 0 & 0 & 0 & 0 & x & x & 0 \\ 0 & 0 & 0 & 0 & x & 0 & x & x & x \\ 0 & 0 & 0 & 0 & 0 & x & 0 & x & x \end{pmatrix} \cdot \begin{pmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \\ h_9 \end{pmatrix} = \begin{pmatrix} h_{D1} \\ b - (x \cdot h_{D1}) - (x \cdot h_{D4}) \\ b - (x \cdot h_{D1}) - (x \cdot h_{D4}) \\ h_4 D \\ b - (x \cdot h_{D1}) - (x \cdot h_{D4}) \\ b - (x \cdot h_{D1}) - (x \cdot h_{D4}) \\ b - (x \cdot h_{D1}) - (x \cdot h_{D4}) \\ b - (x \cdot h_{D1}) - (x \cdot h_{D4}) \\ b - (x \cdot h_{D1}) - (x \cdot h_{D4}) \end{pmatrix} \tag{3.33}$$

Im Lastvektor treten nur die Terme auf, für die x an der entsprechenden Stelle ungleich Null ist.

Die zweite Möglichkeit besteht darin, nur die Einträge im Lastvektor $\mathbf{b}$ zu berücksichtigen und die Zeilen und Spalten, welche auf Null gesetzt werden, nicht in die Matrix zu integrieren. Die Ordnung der Matrix verringert sich somit um die Anzahl der Zellen, in welchen DIRICHLET Randbedingungen definiert sind.

$$\begin{pmatrix} x & x & x & 0 & 0 & 0 & 0 \\ x & x & 0 & x & 0 & 0 & 0 \\ x & 0 & x & x & 0 & x & 0 \\ 0 & x & x & x & 0 & 0 & x \\ 0 & 0 & 0 & 0 & x & x & 0 \\ 0 & 0 & x & 0 & x & x & x \\ 0 & 0 & 0 & x & 0 & x & x \end{pmatrix} \cdot \begin{pmatrix} h_2 \\ h_3 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \\ h_9 \end{pmatrix} = \begin{pmatrix} b-(x \cdot h_{D1})-(x \cdot h_{D4}) \\ b-(x \cdot h_{D1})-(x \cdot h_{D4}) \\ b-(x \cdot h_{D1})-(x \cdot h_{D4}) \\ b-(x \cdot h_{D1})-(x \cdot h_{D4}) \\ b-(x \cdot h_{D1})-(x \cdot h_{D4}) \\ b-(x \cdot h_{D1})-(x \cdot h_{D4}) \\ b-(x \cdot h_{D1})-(x \cdot h_{D4}) \end{pmatrix} \tag{3.34}$$

Der Vektor $\mathbf{x}$ der Unbekannten und der DIRICHLET Bedingungen, wird in zwei Vektoren aufgeteilt:

$$\mathbf{x} = \mathbf{x}^{\mathbf{unbekannt}} + \mathbf{x}^{\mathbf{bekannt}}.$$

Daraus folgt:

$$\mathbf{A} \cdot \left(\mathbf{x}^{\mathbf{unbekannt}} + \mathbf{x}^{\mathbf{bekannt}}\right) = \mathbf{b}, \tag{3.35}$$

nach dem Auflösen der Summe und Verschiebung auf die rechte Seite ergibt sich

$$\mathbf{A} \cdot \mathbf{x}^{\mathbf{unbekannt}} = \mathbf{b} - \mathbf{A} \cdot \mathbf{x}^{\mathbf{bekannt}} \tag{3.36}$$

Diese Vorgehensweise ist für Strömungs- und Stofftransport identisch.

NEUMANN **Bedingungen**

NEUMANN Randbedingungen schreiben den Gradienten $\frac{\partial h}{\partial \mathbf{n}} = q^*_{(n)}$ senkrecht auf dem Rand vor. Falls auf einer der linearen Randkomponenten eine NEUMANN Randbedingung vorgegeben ist, so wird das Integral durch diese Randbedingung ersetzt.

$$\int_{\Gamma_{ij}} \boldsymbol{\nu}_{ij}^T \cdot \mathbf{K}_f \cdot \boldsymbol{\nabla} h^{(t+\Delta t)} d\sigma = q^*_{(n)ij}. \tag{3.37}$$

Bei natürlichen NEUMANN Randbedingungen werden die entsprechenden Randintegrale auf Null gesetzt. Es findet also kein Massenfluß über den Rand in oder aus dem Kontrollvolumen statt. Die Randbedingungen $q^*_{(n)ij}$ werden anschließend in den Lastvektor integriert. Sie haben keinen Einfluss auf die Steifigkeits- oder Massenmatrix.

3.4 Diskretisierung der Transportgleichung

3.4.1 Finite Volumen Formulierung

Die Diskretisierung der Transportgleichung entspricht im wesentlichen der der Strömungsgleichung, siehe Abschnitt 3.3.1 auf Seite 40.

Mit der impliziten Zeitdiskretisierung aus Gleichung (3.2) läßt sich Gleichung (7.1) wie folgt formulieren

$$\frac{c^{(t+\Delta t)} - c^{(t)}}{\Delta t} R = \boldsymbol{\nabla} \cdot (\mathbf{D}^* \cdot \boldsymbol{\nabla} c^{(t+\Delta t)}) - \mathbf{u} \cdot \boldsymbol{\nabla} c^{(t+\Delta t)} + \sigma - \frac{q}{n_f}(c^{(t+\Delta t)} - c_{(in)}), \quad (3.38)$$

mit

$$b = \frac{q}{n_f} c_{(in)} - \frac{c^{(t)}}{\Delta t} R - \sigma$$

und

$$r = -\frac{q}{n_f} - \frac{R}{\Delta t}$$

ergibt sich

$$\boldsymbol{\nabla} \cdot (\mathbf{D}^* \cdot \boldsymbol{\nabla} c^{(t+\Delta t)}) - \mathbf{u} \cdot \boldsymbol{\nabla} c^{(t+\Delta t)} + r c^{(t+\Delta t)} = b. \quad (3.39)$$

Bevor die FVM auf die Transportgleichung angewendet werden kann, muss diese in die konservative Form überführt werden. Dazu wird der zusätzliche Term $c^{(t+\Delta t)} \boldsymbol{\nabla} \mathbf{u} = 0$ eingeführt. Dieser Term ist Null in Quell- und Senkfreien Gebieten. Die Transportgleichnung verändert sich zu

$$\boldsymbol{\nabla} \cdot (\mathbf{D}^* \cdot \boldsymbol{\nabla} c^{(t+\Delta t)}) - \mathbf{u} \cdot \boldsymbol{\nabla} c^{(t+\Delta t)} + c^{(t+\Delta t)} \boldsymbol{\nabla} \mathbf{u} + r c^{(t+\Delta t)} = b \quad (3.40)$$

und läßt sich wie folgt umformen

$$\boldsymbol{\nabla} \cdot (\mathbf{D}^* \cdot \boldsymbol{\nabla} c^{(t+\Delta t)} - \mathbf{u} c^{(t+\Delta t)}) + r c^{(t+\Delta t)} = b \quad (3.41)$$

Nun wird Gleichung (3.41) über jedes Kontrollvolumen integriert.

$$\int_{\Omega_i} \boldsymbol{\nabla} \cdot (\mathbf{D}^* \cdot \boldsymbol{\nabla} c^{(t+\Delta t)} - \mathbf{u} \cdot c^{(t+\Delta t)}) + r c^{(t+\Delta t)} dx = \int_{\Omega_i} b dx. \quad (3.42)$$

Unter Anwendung des Gaußschen Integralsatzes ergibt sich

$$\int_{\partial\Omega_i} \boldsymbol{\nu}^T \cdot (\mathbf{D}^* \cdot \boldsymbol{\nabla} c^{(t+\Delta t)} - \mathbf{u} c^{(t+\Delta t)}) d\sigma + \int_{\Omega_i} r c^{(t+\Delta t)} dx = \int_{\Omega_i} b dx \; . \quad (3.43)$$

Anschließend wird das Randintegral in eine Summe zerlegt:

$$\sum_{j \in \Lambda_i} \int_{\Gamma_{ij}} \boldsymbol{\nu}_{ij}^T \cdot (\mathbf{D}^* \cdot \boldsymbol{\nabla} c^{(t+\Delta t)} - \mathbf{u} c^{(t+\Delta t)}) d\sigma + \int_{\Omega_i} r c^{(t+\Delta t)} dx = \int_{\Omega_i} b dx. \quad (3.44)$$

Diskretisierung des Diffusionsterms

Zunächst soll der Diffusionsterm

$$\sum_{j\in\Lambda_i}\int_{\Gamma_{ij}} \boldsymbol{\nu}_{ij}^T\cdot\mathbf{D}^*\cdot\boldsymbol{\nabla}c^{(t+\Delta t)}d\sigma$$

betrachtet werden. Neben dem Normalenvektor auf Γ_{ij} setzt sich dieser Term aus $\mathbf{D}^*$ und der partiellen Ableitung von $c^{(t+\Delta t)}$ zusammen. $\mathbf{D}^*$ wird aus dem Tensor der molekularen Diffusion $\mathbf{D}_{diff}$ und dem Tensor der korngerüstbedingten Dispersion $\mathbf{D}_{disp}$ gebildet und ist voll besetzt. In Komponentenschreibweise ergibt sich:

$$\begin{aligned}&\boldsymbol{\nu}_{ij}^T\cdot(\mathbf{D}_{diff}+\mathbf{D}_{disp})\cdot\boldsymbol{\nabla}c^{(t+\Delta t)} = \\ &\begin{pmatrix}\nu_{ij}^x\\ \nu_{ij}^y\\ \nu_{ij}^z\end{pmatrix}^T\begin{pmatrix}D^{xx}_{(F)ij} & 0 & 0\\ 0 & D^{yy}_{(F)ij} & 0\\ 0 & 0 & D^{zz}_{(F)ij}\end{pmatrix}+\begin{pmatrix}D^{xx}_{(S)ij} & D^{xy}_{(S)ij} & D^{xz}_{(S)ij}\\ D^{yx}_{(S)ij} & D^{yy}_{(S)ij} & D^{yz}_{(S)ij}\\ D^{zx}_{(S)ij} & D^{zy}_{(S)ij} & D^{zz}_{(S)ij}\end{pmatrix}\begin{pmatrix}\frac{\partial c^{(t+\Delta t)}}{\partial x}\\ \frac{\partial c^{(t+\Delta t)}}{\partial y}\\ \frac{\partial c^{(t+\Delta t)}}{\partial z}\end{pmatrix}.\end{aligned}\tag{3.45}$$

Aufgrund der numerischen Komplexität eines vollbesetzten Dispersionstensors, sollen hier nur die Einträge auf der Diagonalen betrachtet werden. Zwar bringt dies eine numerische Ungenauigkeit mit sich, jedoch ist die Diskretisierung der Transportgleichung mittels der FVM wesentlich einfacher. Anderenfalls müßten zusätzliche Einträge im linearen Gleichungssystem erzeugt werden.

Mit dieser Annahme vereinfacht sich Gleichung (3.45) zu

$$\begin{aligned}&\boldsymbol{\nu}_{ij}^T\cdot(\mathbf{D}_{diff}+\mathbf{D}_{disp})\cdot\boldsymbol{\nabla}c^{(t+\Delta t)} = \\ &\begin{pmatrix}\nu_{ij}^x\\ \nu_{ij}^y\\ \nu_{ij}^z\end{pmatrix}^T\begin{pmatrix}D^{xx}_{(F)ij} & 0 & 0\\ 0 & D^{yy}_{(F)ij} & 0\\ 0 & 0 & D^{zz}_{(F)ij}\end{pmatrix}+\begin{pmatrix}D^{xx}_{(S)ij} & 0 & 0\\ 0 & D^{yy}_{(S)ij} & 0\\ 0 & 0 & D^{zz}_{(S)ij}\end{pmatrix}\begin{pmatrix}\frac{\partial c^{(t+\Delta t)}}{\partial x}\\ \frac{\partial c^{(t+\Delta t)}}{\partial y}\\ \frac{\partial c^{(t+\Delta t)}}{\partial z}\end{pmatrix}.\end{aligned}\tag{3.46}$$

Die partiellen Ableitungen werden wie in Abschnitt 3.3.1 beschrieben durch Differenzenquotienten angenähert. Zur Approximation der Tensoreinträge auf dem jeweiligen Rand wird zwischen benachbarten Zellen i und j das harmonische Mittel gebildet:

$$D^{kl}_{(F)ij}=\frac{2D^{kl}_{(F)i}D^{kl}_{(F)j}}{D^{kl}_{(F)i}+D^{kl}_{(F)j}}$$

und

$$D^{kl}_{(S)ij}=\frac{2D^{kl}_{(S)i}D^{kl}_{(S)j}}{D^{kl}_{(S)i}+D^{kl}_{(S)j}}.$$

Eingesetzt in (3.46) ergibt sich

$$\boldsymbol{\nu}_{ij}^T\cdot(\mathbf{D}_{diff}+\mathbf{D}_{disp})\cdot\boldsymbol{\nabla}c^{(t+\Delta t)}=(c_j^{(t+\Delta t)}-c_i^{(t+\Delta t)})\sum_{k\in\beta}|\nu_{ij}^k|\sum_{l\in\beta}\frac{1}{d_{ij}}(D^{kl}_{(F)ij}\delta_{kl}+D^{kl}_{(S)ij}\delta_{kl}).\tag{3.47}$$

Diese Gleichung kann folgendermaßen vereinfacht werden

$$\boldsymbol{\nu}_{ij} \cdot (\mathbf{D}_{diff} + \mathbf{D}_{disp}) \cdot \boldsymbol{\nabla} c^{(t+\Delta t)} = (c_j^{(t+\Delta t)} - c_i^{(t+\Delta t)}) \sum_{k \in \beta} |\nu_{ij}^k| \frac{1}{d_{ij}} (D_{(F)ij}^{kk} + D_{(S)ij}^{kk}). \quad (3.48)$$

Die Komponenten $D_{(F)ij}^{kl}$ und $D_{(S)ij}^{kl}$ sind die harmonisch gemittelten Tensoreinträge auf dem Rand Γ_{ij}.

Diskretisierung des Advektionsterms

Für den Advektionsterm aus (3.44)

$$-\sum_{j \in \Lambda_i} \int_{\Gamma_{ij}} \boldsymbol{\nu}_{ij}^T \cdot \mathbf{u} c^{(t+\Delta t)} d\sigma$$

wird folgender Ansatz gemacht. Der Geschwindigkeitsvektor $\mathbf{u}$ wird auf dem Rand Γ_{ij} zwischen den Punkten i und j durch $\mathbf{u}_{ij}$ approximiert. Der Vektor $\mathbf{u}_{ij}$ berechnet sich aus den Mittelwerten der Geschwindigkeitsvektoren in den Punkten i und j oder wird direkt auf dem Rand, basierend auf dem Gesetz von Darcy, aus den piezometrischen Druckhöhen h_i, h_j, der Tensorkomponente k_{ij}^{kk} und der effektiven Porosität berechnet. Die Diskretisierung des Integrals $c|_{\Gamma_{ij}}$ erfolgt nach [KA00] durch eine Konvexkombination von $c^{(t+\Delta t)}$ in den Knoten i und j:

$$c|_{\Gamma_{ij}} = p_{ij} c_i^{(t+\Delta t)} + (1 - p_{ij}) c_j^{(t+\Delta t)},$$

wobei p_{ij} ein Wichtungsparameter ist, welcher später beschrieben wird. Somit ergibt sich für den Advektionsterm folgende Formulierung:

$$-\sum_{j \in \Lambda_i} \int_{\Gamma_{ij}} \sum\nolimits_{k \in \beta} |\nu_{ij}^k| u_{ij}^k (p_{ij} c_i^{(t+\Delta t)} + (1 - p_{ij}) c_j^{(t+\Delta t)}) d\sigma. \quad (3.49)$$

Gesamtbild

Das Einsetzen der Diskretisierungen der Diffusions- und Advektionsterme in (3.44) ergibt

$$\sum_{j \in \Lambda_i} \int_{\Gamma_{ij}} [(c_j^{(t+\Delta t)} - c_i^{(t+\Delta t)}) \sum_{k \in \beta} |\nu_{ij}^k| \frac{1}{d_{ij}} (D_{(F)ij}^{kk} + D_{(S)ij}^{kk}) - \sum_{k \in \beta} |\nu_{ij}^k| u_{ij}^k (p_{ij} c_i^{(t+\Delta t)} + (1 - p_{ij}) c_j^{(t+\Delta t)})] d\sigma + \int_{\Omega_i} r_i h_i^{(t+\Delta t)} dx = \int_{\Omega_i} b_i dx. \quad (3.50)$$

Die Integrale $\int_{\Gamma_{ij}}$ und $\int_{\Omega_i}$ werden durch die Konstanten m_{ij} und m_i approximiert. Die Berechnung der Konstanten entspricht exakt der der Strömungsdiskretisierung, siehe Gleichungen (3.24) und (3.7).

Aus Gleichung (3.39) ergibt sich also folgende Finite Volumen Diskretisierung:

$$\sum_{j\in\Lambda_i}[(c_j^{(t+\Delta t)}-c_i^{(t+\Delta t)})\sum_{k\in\beta}|\nu_{ij}^k|\frac{1}{d_{ij}}(D_{(F)ij}^{kk}+D_{(S)ij}^{kk})- \sum_{k\in\beta}|\nu_{ij}^k|u_{ij}^k(p_{ij}c_i^{(t+\Delta t)}+(1-p_{ij})c_j^{(t+\Delta t)})]m_{ij}+m_i r_i c_i^{(t+\Delta t)}=m_i b_i. \quad (3.51)$$

Wichtungsparameter und Stabilisierung

Der bei der Diskretisierung des Advektionsterms eingeführte Wichtungsparameter p_{ij} spielt für die Stabilisierung der Lösung eine wichtige Rolle. Nach [KA00] gilt folgende Darstellung

$$p_{ij}=p(\frac{\sum_{k\in\beta}|\nu_{ij}^k|u_{ij}^k\cdot\sum_{k\in\beta}|\nu_{ij}^k|d_{ij}}{\sum_{k\in\beta}|\nu_{ij}^k|\sum_{l\in\beta}(D_{(F)ij}^{kl}\delta_{kl}+D_{(S)ij}^{kl}\delta_{kl})})$$

Das Argument für p_{ij} ist die *lokale Peclet-Zahl*. Implementiert sind folgende Funktionen: das so genannte *full upwinding*

$$p(z)=\frac{1}{2}[sig(z)+1]$$

und das *exponential upwinding*

$$p(z)=1-\frac{1}{z}(1-\frac{z}{e^z-1}).$$

Zur Stabilisierung der Lösung der Transportgleichung gibt es eine Reihe von Kriterien. Diese Kriterien beziehen sich meist auf eine explizite Zeitdiskretisierung. Da hier eine implizite Diskretisierung gewählt wurde und damit die Kriterien nicht unbedingt eingehalten werden müssen, werden diese nur kurz vorgestellt. Es ist allerdings von numerischen Vorteil die im weiteren aufgeführten Kriterien bei einer impliziten Euler Diskretisierung einzuhalten.

Courant-Kriterium

Das Courant-Kriterium ist nach [RSW02] für eine eindimensionale Diskretisierung wie folgt definiert:

$$\left|\frac{u^x\Delta t}{\Delta x}\right|\leq 1$$

und besagt, daß das Verhältnis der Abstandsgeschwindigkeit u über die Zeit Δt zur Maschenweite Δx kleiner gleich eins sein sollte. Dieses Kriterium muß bei einer expliziten Euler Diskretisierung eingehalten werden, da es sonst zu Instabilität oder unphysikalischen Konzentrationen kommen kann.

Neumann-Kriterium

Nach [RSW02] besagt dieses Kriterium, daß ein Konzentrationsgradient allein durch dispersiven Transport nicht umgekehrt werden darf. Es muß bei einer expliziten Euler Diskretisierung eingehalten werden.

Brunnen-Kriterium

Das Brunnen-Kriterium besagt, daß pro Zeitschritt nicht mehr Masse entnommen werden darf, als zu Anfang des Zeitschritts vorhanden war. Es halbiert das Courant-Kriterium auf den Wert $0,5$ und spielt nur bei der expliziten Euler Diskretisierun eine Rolle. [RSW02]

Numerische Dispersion

Das hier gewählte numerische Verfahren, sowie die FDM und FEM Verfahren, haben den Nachteil, daß sie sich weniger gut zur Lösung von hyperbolischen Gleichungen eignen. Die Transportgleichung besitzt jedoch einen parabolischen und einen hyperbolischen Term. Dieser Umstand führt bei der Diskretisierung durch die oben erwähnten Verfahren zu dem Effekt der numerischen Dispersion. Das bedeutet, daß sich scharfe Konzentrationsfronten aufgrund der räumlichen Diskretisierung aufweiten. Eine Abschätzung für die implizite Euler Diskretisierung erfolgt durch

$$\alpha_N = \frac{1}{2}(\Delta x + u\Delta t).$$

Der Effekt der numerischen Dispersion kann durch eine sehr feine räumliche und zeitliche Auflösung vermindert werden, was allerdings zu einem wesentlich höherem Rechenaufwand führt. Eine Maßzahl, die den Charakter der Transportgleichung angibt, ist die Gitter-Peclet-Zahl Pe_g

$$Pe_g = \frac{\Delta x}{\alpha_L}.$$

Die Gitter-Peclet-Zahl sollte immer kleiner 2 sein, um den parabolischen Charakter der Transportgleichung hervorzuheben. Die numerische Dispersion wird dann klein gegen die physikalische Dispersion [RSW02].

3.4.2 Aufstellung der Steifigkeits- und Massenmatrix

Für die Aufstellung des Gleichungssystems werden die Terme r_i und b_i aus Gleichung (3.41) zurück substituiert.

$$\sum_{j\in\Lambda_i}[\left(c_j^{(t+\Delta t)}-c_i^{(t+\Delta t)}\right)\sum_{k\in\beta}|\nu_{ij}^k|\frac{1}{d_{ij}}\left(D_{(F)ij}^{kk}+D_{(S)ij}^{kk}\right)-$$
$$\sum_{k\in\beta}|\nu_{ij}^k|u_{ij}^k\left(p_{ij}c_i^{(t+\Delta t)}+(1-p_{ij})c_j^{(t+\Delta t)}\right)]m_{ij}+m_i\left(-\frac{q_i}{nf_i}-\frac{R_i}{\Delta t}\right)c_i^{(t+\Delta t)}=$$
$$m_i\left(\frac{q_i}{nf_i}c_{(in)i}-\frac{c_i^{(t)}}{\Delta t}R_i-\sigma_i\right) \quad (3.52)$$

oder

$$\sum_{j\in\Lambda_i}c_j^{(t+\Delta t)}\left[\sum_{k\in\beta}|\nu_{ij}^k|\frac{1}{d_{ij}}(D_{(F)ij}^{kk}+D_{(S)ij}^{kk})-\sum_{k\in\beta}|\nu_{ij}^k|u_{ij}^k(1-p_{ij})\right]m_{ij}+$$
$$\left[-\sum_{j\in\Lambda_i}\left(\sum_{k\in\beta}|\nu_{ij}^k|\frac{1}{d_{ij}}(D_{(F)ij}^{kk}+D_{(S)ij}^{kk})-\sum_{k\in\beta}|\nu_{ij}^k|u_{ij}^k p_{ij}\right)m_{ij}+m_i(-\frac{q_i}{nf_i}-\frac{R_i}{\Delta t})\right]c_i^{(t+\Delta t)}=$$
$$m_i(\frac{q_i}{nf_i}c_{(in)i}-\frac{c_i^{(t)}}{\Delta t}R_i-\sigma_i) \quad (3.53)$$

Die Steifigkeitsmatrix $\mathbf{S}$ bildet sich aus

$$\left[\sum_{k\in\beta}|\nu_{ij}^k|\frac{1}{d_{ij}}\left(D_{(F)ij}^{kk}+D_{(S)ij}^{kk}\right)-\sum_{k\in\beta}|\nu_{ij}^k|u_{ij}^k(1-p_{ij})\right]m_{ij} \quad (3.54)$$

auf den Nebendiagonalen und

$$-\sum_{j\in\Lambda_i}\left(\sum_{k\in\beta}|\nu_{ij}^k|\frac{1}{d_{ij}}(D_{(F)ij}^{kk}+D_{(S)ij}^{kk})-\sum_{k\in\beta}|\nu_{ij}^k|u_{ij}^k p_{ij}\right)m_{ij} \quad (3.55)$$

auf der Diagonalen.

Die Einträge der Massenmatrix $\mathbf{M}$ auf der Diagonalen werden durch

$$m_i(-\frac{q_i}{nf_i}-\frac{R_i}{\Delta t}) \quad (3.56)$$

gebildet. Die Einträge in den Lastvektor $\mathbf{b}$ ergeben sich durch

$$\mathbf{b}=m_i(\frac{q_i}{nf_i}c_{(in)i}-\frac{c_i^{(t)}}{\Delta t}R_i-\sigma_i). \quad (3.57)$$

Das Gleichungssystem lautet somit

$$(\mathbf{M}+\mathbf{S})\mathbf{x}=\mathbf{b}. \quad (3.58)$$

Es ist schwach besetzt und unsymmetrisch.

3.4.3 Einbau der Randbedingungen

In das vorgestellte Modell wurden die DIRICHLET, NEUMANN und Transmissions-Randbedingungen implementiert.

DIRICHLET Bedingungen

Es wird eine Konzentration auf dem Rand vorgegeben. Die Implementierung der DIRICHLET Bedingungen entspricht exakt der der Strömungsgleichung.

NEUMANN Bedingungen

NEUMANN Randbedingungen schreiben den dispersiven Fluss $\mathbf{D}^* \frac{\partial c}{\partial \mathbf{n}}$ senkrecht auf dem Rand vor. Aufgrund der polygonalen Berandung lassen sich die Massenbilanzen über den kompletten Rand in eine Summer aller linearen Randelemente zerlegen.

$$\int_{\partial\Omega_i} \boldsymbol{\nu}^T \cdot \mathbf{D}^* \cdot \boldsymbol{\nabla} c d\sigma = \sum_{j \in \Lambda_i} \int_{\Gamma_{ij}} \boldsymbol{\nu}_{ij}^T \cdot \mathbf{D}^* \cdot \boldsymbol{\nabla} c d\sigma. \tag{3.59}$$

Falls auf einer der linearen Randkomponenten eine NEUMANN Randbedingung vorgegeben ist, so wird das Integral durch diese Randbedingung ersetzt.

$$\int_{\Gamma_{ij}} \boldsymbol{\nu}_{ij}^T \cdot \mathbf{D}^* \cdot \boldsymbol{\nabla} c d\sigma = j_N. \tag{3.60}$$

Bei natürlichen NEUMANN Randbedingungen werden die entsprechenden Randintegrale gleich 0 gesetzt. Es findet also kein Konzentrationsfluß über den Rand in oder aus dem Kontrollvolumen statt. Die Randbedingungen j_N werden in den Lastvektor integriert. Natürlichen NEUMANN Randbedingungen werden durch so genannte Geisterknoten realisiert, die außerhalb des Modellgebietes liegen, aber direkt daran angrenzen. In diesen Knoten sind die k_f und Dispersions-/Diffusionswerte auf Null gesetzt. Bei der Bildung der Massenbilanz zwischen echten Knoten und Geisterknoten findet somit kein advektiver und dispersiver Fluss statt.

Transmissions Randbedingungen

Um den ungehinderten Stofftransport über einen Modellrand zu modellieren, werden Transmissionsrandbedingungen benötigt. Im allgemeinen schreiben diese den dispersiven und advektiven Fluss über den Rand vor.

$$\int_{\Gamma_{ij}} \boldsymbol{\nu}_{ij}^T \cdot \mathbf{u_{ij}} \left(p_{ij} c_i^{(t+\Delta t)} + (1 - p_{ij}) c_j^{(t+\Delta t)} \right) = j_T. \tag{3.61}$$

3.5 Lösungsmethoden

Die bei der Finite Volumen Diskretisierung entstehenden Gleichungssysteme können mittels iterativer und direkter Lösungsmethoden gelöst werden. Die verwendete GRASS API[1] stellt eine Reihe von direkten und iterativen Gleichungslösern zur Verfügung. So sind als direkte Verfahren die Gauss-Elimination und die LU Zerlegung implementiert. Als iterative Gleichungslöser sind das Jacobi-Verfahren, das Gauß-Seidel Verfahren, das Verfahren der konjugierten Gradienten und das Verfahren der bikonjugierten Gradienten verfügbar. Als Besonderheiten seien die Unterstützung von Kompaktspeichermatrizen sowie die parallele Berechnung der Gleichungssysteme auf Supercomputern in GRASS erwähnt. Diese Eigenschaften von GRASS ermöglicht die Lösung großer realer Problemstellungen.

Iterative Lösungsmethoden

Iterative Methoden haben den großen Vorteil, die besonderen Eigenschaften schwach besetzter Gleichungssysteme voll ausnutzen zu können [Hac93]. Außerdem verbrauchen sie wesentlich weniger Speicherplatz, als direkte Lösungsverfahren und lassen sich teilweise leicht implementieren. Weiterhin sind diese Methoden die einzige Möglichkeit nichtlineare Gleichungssysteme zu lösen[2]. Bei der Benutzung iterativer Lösungsmethoden werden die Matrizen nicht mehr, wie bei dem Gauß Verfahren oder dem Cholesky Verfahren, aufgefüllt. Somit ist es möglich nur die Einträge der Matrizen zu speichern, welche auch besetzt sind. Man nennt dieses Matrizen auch *Kompaktspeichermatrizen* (Sparse-Matrizen). Dies bringt erhebliche Einsparungen an Arbeitsspeicher mit sich.

Eine sehr gute und ausführliche Übersicht über iterative Verfahren schwach besetzter Gleichungssysteme bekommt man in [Hac93] und [Mei99].

Die bei der Strömungs- und Stofftransportmodellierung erzeugten Gleichungssysteme haben spezielle Eigenschaften. Die Diskretisierung der Strömungsgleichung mit der FVM führt immer auf symmetrische, positiv definite, schwach besetzte Gleichungssysteme. Das für diese Systeme ideale Verfahren ist die Methode der konjugierten Gradienten (CG). Im Gegensatz dazu führt die FVM Diskretisierung der Transportgleichung aufgrund ihres parabolischen und hyperbolischen Charakters auf unsymmetrische schwach besetzte Gleichungssysteme. Da das CG Verfahrten symmetrische Gleichungssysteme voraussetzt, kann es hier nicht verwendet werden. Aus diesem Grund wurde das in GRASS implementiert Verfahren der stabilisierte bikonjugierten Gradienten (BiCGStab) verwendet. Beide Verfahren sollen kurz vorgestellt werden.

[1] **A**pplication **P**rogramming **I**nterface

[2] abgesehen von dem Newton Verfahren, in welchem die erzeugten Gleichungssysteme linear sind und somit direkt gelöst werden können

Die Methode der konjugierten Gradienten (CG)

Als effizientestes Verfahren zur Lösung symmetrischer, positiv definiter, schwach besetzter Gleichungssysteme hat sich das Verfahren der konjugierten Gradienten erwiesen. Die Implementierung in GRASS entspricht der in [Mei99] auf Seite 124ff vorgestellten Variante.

Das CG Verfahren gehört zu den so genannten Projektionsmethoden und ist einer der bedeutendsten Algorithmen die im 20. Jahrhundert entwickelt wurden. Das CG-Verfahren setzt sich aus zwei verschiedenen Projektionsverfahren zusammen: aus dem *Verfahren des steilsten Abstiegs* und dem *Verfahren der Konjugierten Richtungen*. Das CG Verfahren kombiniert die Vorteile dieser beiden Methoden. In Abbildung 3.5 ist die Herleitung des CG-Verfahrens dargestellt.

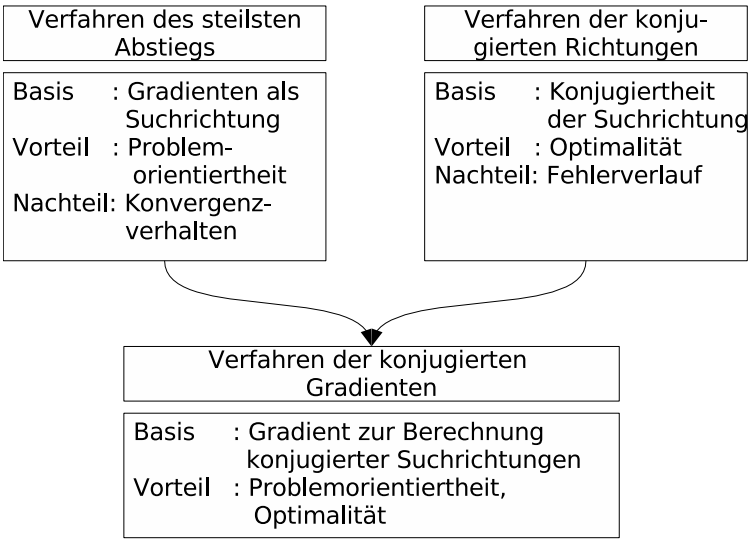


Abbildung 3.5: Herleitung des CG-Verfahrens [Mei99].

Das stabilisierte, bikonjugierte Gradienten Verfahren (BiCGStab)

Das BiCGStab Verfahren ist eine sehr populäre Projektionsmethode zur Lösung unsymmetrischer Matrizen. Es gibt eine Reihe von Projektionsmethoden die unsymmetrische Gleichungssysteme lösen können, etwa das GMRES oder CGS Verfahren. Das BiCGStab Verfahren jedoch sticht aus diesen aufgrund seiner guten Konvergenz und des geringen Speicherbedarfs hervor. Es kann als eine Weiterentwicklung des CG-Verfahrens für unsymmetrische Matrizen angesehen werden. Für eine genauere Beschreibgun des Verfahrens sei auf [Mei99] verwiesen. Der dort beschriebene Algorithmus ist in GRASS implementiert.

4 Das GIS GRASS

Das geographische Informationssystem GRASS (***G**eographical **R**esources **A**nalysis and **S**upport **S**ystem*) ist ein kombiniertes Raster-/Vektor- und Voxel-GIS. Es ist das größte Open Source GIS Softwareprojekt und, gemessen an der Anzahl der Programmcodezeilen, auch eines der größten Open Source Projekte die momentan entwickelt werden. Es enthält über 350 Programme zur Verarbeitung von Raster- Vektor- und Voxeldaten, zur Erzeugung von digitalen Karten, der Darstellung von räumlichen Daten in zwei und drei Dimensionen sowie umfassende Werkzeuge zur Satellitenbildverabreitung.

Das GIS GRASS wird von einem kleinen Entwicklerteam auf freiwilliger Basis entwickelt und gepflegt. Das Team ist über das Internet organisiert und kommuniziert über Mailinglisten, IRC Chats, Wiki's sowie persönlich auf Open Source GIS Konferenzen. Die Community inklusive der Entwicklergemeinde ist international.

Das GIS GRASS wird am ITC-irst in Trento, Italien, von Markus Neteler koordiniert. In regelmäßgen Abständen werden von der Entwicklergemeinde neue Versionen veröffentlicht und der Allgemeinheit auf der GRASS Homepage[1] zur Verfügung gestellt.

GRASS steht seit 1999 unter der GNU Genral Public License (GPL). Diese Lizenz schützt die Software vor dem Missbrauch durch Dritte und garantiert das Urheberrecht eines jeden Entwicklers. Gleichzeitig bietet die GPL die Möglichkeit, dass jeder den Quellcode studieren und analysieren, Bugfixes und Erweiterungen beisteuern sowie eigene Entwicklungen basierend auf dem GPL Quellcode erstellen kann.

4.1 Geschichte

Das GIS GRASS wurde von 1982 bis 1995 am U.S. Army Corps of Engineers Construction Engineering Research Laboratory (CERL) in Champaign, Illinois entwickelt. Es unterstützte dort die Verwaltung von Millionen von Hektar U.S. amerikanischer Grundstücke und Anlagen. Desweiteren wurde es in der Forstindustrie, der Landwirtschaft und dem Umweltmanagement eingesetzt. [NM04] GRASS war in den achziger Jahren und Anfang der neunziger Jahre in den militärischen Einrichtungen der USA weit verbreitet. Mitte der neunziger Jahre erschlossen sich, aufgrund der rapide zugenommenen Rechenleistung von kostengünstigen Standardcomputern, neue Anwendungsgebiete

[1] http://grass.itc.it

für die Geoinformatik. Die Nachfrage nach GIS Lösungen vergrößerte sich beständig und es wurde nun ausreichend Geld für die Geoinformationsinfrastruktur zur Verfügung gestellt. Ein Paradigmenwechsel in der militärischen Führung, *weg von Eigenentwicklungen, hin zu kommerziell verfügbarer Lösungen*, führte dazu, dass viele Einrichtungen von GRASS zu proprietären GIS Lösungen migrierten. Daraufhin stoppte das CERL 1995 die Entwicklung und stellte den Quellcode lizenziert unter der BSD[2] Lizens im Internet zur freien Verfügung. Nun erfolgte die Entwicklung von GRASS auf freiwilliger Basis einer immer größer werdenden internationalen Gemeinschaft. Im Jahr 1997 wurde die Version 4.2 von der Baylor University in den USA veröffentlicht, die zu diesem Zeitpunkt die GRASS Entwicklung steuerte. Ab 1999 wurde die Entwicklung von der Universität Hannover aus am Institut für Physikalische Geographie und Landschaftsökologie von Markus Neteler koordiniert. Seit 2001 befindet sich das Hauptquatier der GRASS Entwicklung am ITC-irst, Trento Italien, koordiniert von Markus Neteler. Die aktuelle (April 2007) stabile Version hat die Nummer 6.2.1 und wurde am 12. Dezember 2006 veröffentlicht. GRASS ist ein wichtiger Meilenstein in der Entwicklung von Geoinformationssystemen und stellte in der Zeit seiner Entwicklung am CERL, eine bis dahin nicht erreichte Funktionalität bereit.

4.2 Aufbau von GRASS

GRASS ist ein multifunktionales Geoinformationssystem. Es verhält sich im Prinzip wie ein normales Anwendungsprogramm, jedoch ist die Programmstruktur etwas anders. GRASS wurde ursprünglich auf Unix-Systemen entwickelt und übernahm von diesen einige Designkonzepte. Ein wichtiges Grundkonzept ist, komplexe Aufgaben in kleinere weniger komplexe Teilbereiche aufzuspalten und diese miteinander zu kombinieren. Somit gibt es kein monolitisches, großes Programm mit einer kaum zu übersehenden Menge an Features, sondern eine Anzahl kleinerer Programme. Diese führen jeweils nur eine spezielle Aufgabe aus, diese aber sehr schnell und exakt. Infolge dessen besteht GRASS aus einer Sammlung von über 350 Programmen, die miteinander kombiniert werden können.

GRASS Programme, im weiteren Module genannt, können über die Unix Shell auf der Kommandozeilenebene, siehe Abbildung 4.1 auf der nächsten Seite oder über ein grafische Oberfläche bedient werden, siehe Abbildung 4.2 auf Seite 60. Wichtig ist, dass die Module über eine gemeinsame Schnittstelle Daten austauschen. Diese gemeinsame Schnittstelle ist das GRASS Datenbanksystem. Dieses System verwaltet neben den Projektmetadaten wie Projektion oder Größe des Projektgebietes, die Raster, Vektor und Volumendaten. Die GRASS Datenbank hat eine spezifische Verzeichnisstruktur. In Abbildung 4.3 auf Seite 61 ist diese Struktur dargestellt. Ein Projektgebiet wird in GRASS als *location* bezeichnet. Sie wird über ihre geographischen Ränder mit Koordinatenangaben und zusätzliche Projektionsangaben definiert. Innerhalb der *location* können beliebig

[2]Berkeley Software Distribution, eine Open Source Lizens der Berkeley University

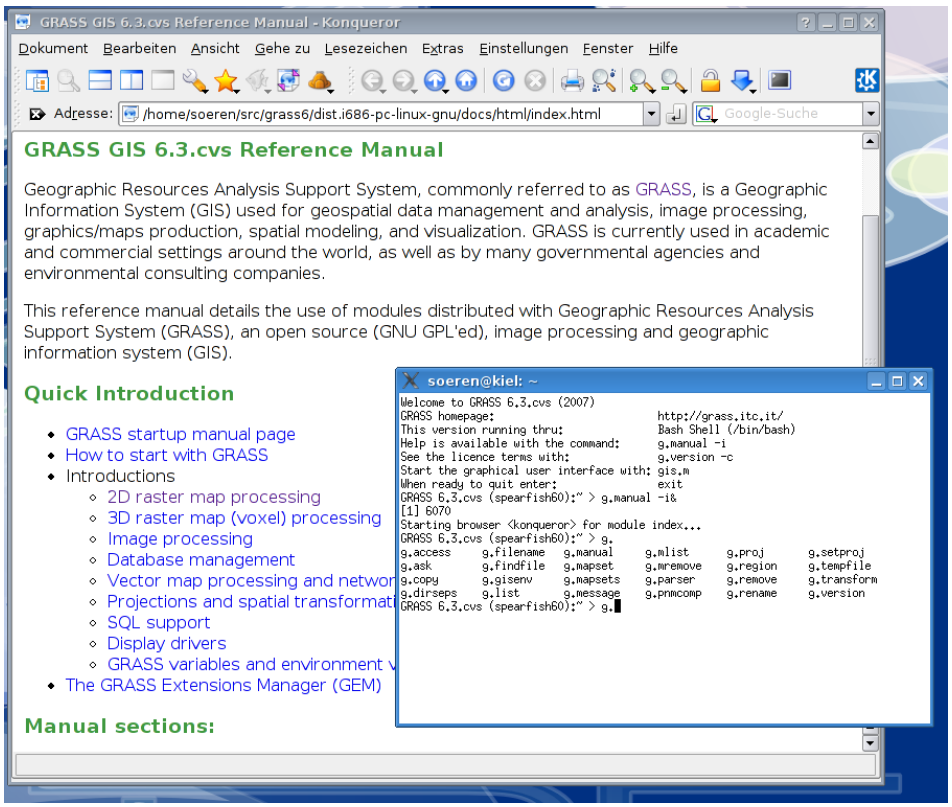


Abbildung 4.1: Die GRASS Shell mit geöffnetem Hilfesystem.

viele Arbeitsgebiete, so genannte *mapsets*, definiert werden. Mehrere *mapsets* bieten sich für Teamarbeit an. Die Module und damit die GIS-Funktionalität sind in GRASS bereits durch ihre Namen sehr klar gegliedert. Es gibt verschiedene Funktionsklassen: Module zur Datenvisualisierung, zur Vektor-, Raster-, und Volumendatenverarbeitung, allgemeine Dateiverarbeitungs-, Kartenerstellungsbefehle usw. Dabei gibt der erste Buchstabe, in Fall von Volumen-, Datenbanken- und Postscriptmodulen sind es zwei Buchstaben, die Funktionsklasse an. Anschließend folgt ein Punkt und ein oder zwei weitere Worte, letztere durch einen weiteren Punkt getrennt. Diese Worte sind der englischen Sprache entnommen und leicht verständlich [NM04]. Tabelle 4.1 listet die wesentlichen Funktionsklassen auf. Aufgrund der hohen Modularität ist es möglich, unterschiedliche GRASS Module miteinander zu kombinieren und somit komplexe Auswertungs- und Analyseprozesse zu erzeugen. Eine weitere Besonderheit ist, dass sich diese Vorgänge vollständig automatisieren lassen. Diese Stärke grenzt GRASS gegenüber anderen proprietären GIS Lösungen ab. GRASS stellt eine große Anzahl an Analysewerkzeugen zur Verfügung, auf die im folgenden kurz eingegangen wird.

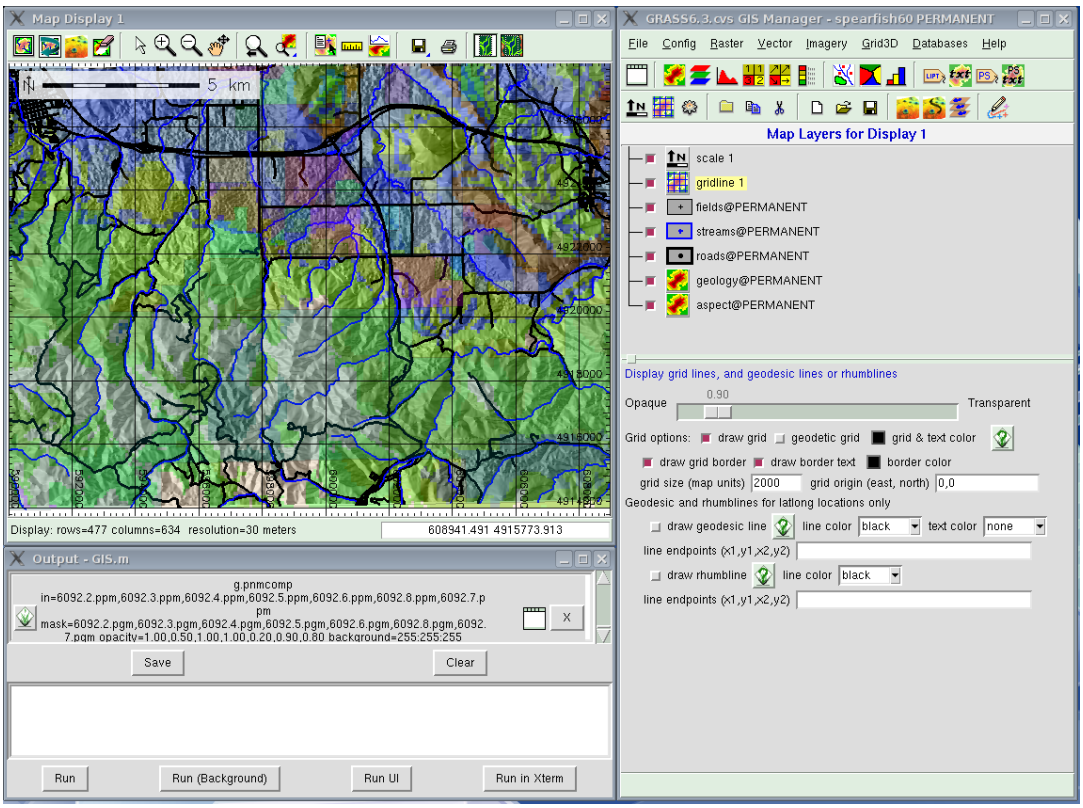


Abbildung 4.2: Die GRASS GUI. Visualisiert werden Raster und Vektordaten aus dem frei erhältlichen spearfish60 GRASS Demodatensatz.

4.2.1 Rasterdaten

GRASS wurde zu Begin seiner Entwicklung als Raster-GIS konzipiert. Es bietet aus diesem Grund eine Vielzahl von Modulen zur Rasterdatenverarbeitung an. Die wichtigsten Verarbeitungsmethoden sind nach [NM04]:

- automatische Raster-/Vektorkonvertierung
- Hangneigungs-/Expositionsberechnung
- Geomorphologische Analysen
- Interpolationen für fehlende Zellenwerte (bilinear, IDW, kubisch, Splines)
- Korrelations-/Kovarianzanalyse
- Puffern von Punkten, Linien, Flächen
- statistische und geostatistische Auswertungen
- Wasserscheiden-, Wassereinzugsgebiet- und Fließpfad-Berechnungen

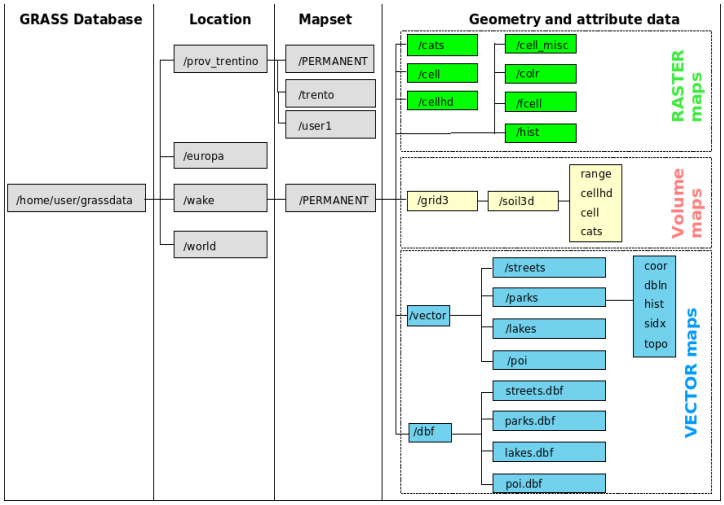


Abbildung 4.3: Die GRASS Datenbank Struktur. (Quelle: Markus Neteler, GRASS programmers manual, http://mpa.itc.it/markus/grass63progman/)

Tabelle 4.1: Funktionsklassen der GRASS Module, nach [NM04]

Präfix	Funktionsklasse	Funktionalität der Module
d.*	display	grafische Ausgabe
db.*	database	Kommandos für die SQL Datenbankverwaltung
g.*	general	Generelle Kommandos für Dateioperationen
i.*	imagery	Bildverarbeitung (Multispektralanalyse, PCA ...)
ps.*	postscript	Kartenerzeugung im Postscript Format
r.*	raster	Rasterdatenverarbeitung (Kartenalgebra, Statistik ...)
r3.*	raster3d	Volumendatenverarbeitung
v.*	vector	Vektordatenverarbeitung

Eine besondere Rolle nehmen in GRASS die Bildverarbeitungsfähigkeiten ein, diese umfassen unter anderem [NM04]:

- Auflösungsverbesserung
- Bildentzerrung (affin, polynomisch) auf Raster- oder Vektorgrundlagen
- Fouriertransformation
- Hauptkomponentenanalyse (PCA)
- Histogrammstreckung und -stauchung
- Image Fusion
- kanonische Komponentenanalyse (CCA)

- Klassifikationen:
  - Radiometrisch: unüberwacht, teilüberwacht und überwacht (Affinity, Maximum Likelihood)
  - Geometrisch/radiometrisch: überwacht (SMAP)
- Kontrastverbesserung
- Koordinatentransformation
- Resampling (bilinear, kubisch, IDW, Splines)

4.2.2 Vektordaten

In den vergangen Jahren wurde die Fähigkeit der Vektordatenverarbeitung von GRASS stark ausgebaut. GRASS besitzt ein topologisches Vektormodell, das mit Punkten, Linien, Polylinen und Flächen in drei Dimensionen umgehen kann. Das Vektormodell erlaubt die Kopplung verschiedener relationaler und objektorientierter Datenbanksysteme mit den GRASS Vektordaten. Unter anderem sind folgende Auswertungsmethoden verfügbar:

- Automatische Vektorisierung von Linien und Flächen
- Manuelle Digitalisierung am Bildschirm oder am Digitalisierbrett
- Distanzberechnung
- Höhenlinienberechnung aus Rasterhöhenmodellen
- Interpolation (Splines)
- Konvertierung Vektor/Raster
- Koordinaten-Transformationen
- Verschneidung von Flächen, Convex hull-Berechnung
- Konvertierung Vektor/Raster
- Geomorphologische Analysen (Profilkrümmung, Hangneigung und -exposition)
- Geostatistik

4.2.3 Volumendaten

Die Verarbeitung von Volumendaten ist ein Alleinstellungsmerkmal von GRASS. Die Unterstützung von Volumendaten wurde schon Mitte der neunziger Jahre implementiert. Seit dem Jahr 2006 wurden diese stark erweitert. Somit stehen aktuell eine Vielzahl von Modellierungs- und Visualisierungsmöglichkeiten für Volumendaten zur Verfügung. Die wichtigsten Verarbeitungsmethoden sind:

Datenkonvertierung von Raster- zu Volumendaten sowie Volumen- zu Rasterdaten, Erstellung geologischer Volumenmodelle basierend auf Höhenkarten, univariate statistische Auswertung, Volumenberechnung, 3D Spline Interpolation und Analyse, 3d Kartenalgebra, Verschnitt von Raster und Volumendaten, 3D Grundwasserströmungsberechnung, Export in verschiedene Visualisierungsformate wie Vis5d und VTK.

4.2.4 Datenaustausch

GRASS kann eine Vielzahl von Raster- und Vektordatenformaten importieren und exportieren. Es benutzt dazu die weit verbreiteten *GDAL*[3] und *OGR*[4] Bibliotheken. Eine vollständige Liste der unterstützten Rasterformate erhält man auf der *GDAL* und *OGR* Homepage *http://www.gdal.org* unter folgendem Link GDAL Formate. Die unterstützten Vektorformate sind hier aufgelistet OGR Formate.

4.2.5 Visualisierung

In GRASS sind verschiedene Visualisierungssysteme implementiert. Es gibt Module mit denen man die grafische Darstellungen von Raster, Vektor und Metadaten von der Kommandozeile aus steuern kann[5]. Des weiteren existiert eine moderne in Tcl/Tk implementierte Benutzeroberfläche, siehe Abbildung 4.2 auf Seite 60. Außerdem besteht die Möglichkeit Postcript-Karten zu erstellen. Zur Visualisierung von 2.5d und 3d Daten steht das OpenGL Visualisierungssystem NVIZ zur Verfügung, mit welchem sich auch Animationen erstellen lassen.

4.3 Programmieren in GRASS

Aufgrund der freien Verfügbarkeit des Quellcodes von GRASS bietet sich die außergewöhnliche Möglichkeit, diesen zu studieren und zu erweitern. Der Quellcode ist portabel und läuft auf einer Reihe von verschiedenen Hardware- und Softwarearchitekturen.

[3] Geospatial Data Abstraction Library
[4] OGR Simple Feature Library
[5] Was im Batchmode sehr praktisch ist

Er wird in einem Software-System zur Versionsverwaltung von Dateien dem *concurrent version system* kurz CVS verwaltet und kann von diesem frei aus dem Internet bezogen werden. Desweiteren stellen die Entwickler wöchentlich einen Snapshot der aktuellen Entwicklung auf der GRASS Homepage zum Download bereit. Zur Kompilierung von GRASS benötigt man einen C-Compiler und die unter Unix zum Entwicklungsstandard gehörende *configure/make* Umgebung. GRASS benötigt verschiedene Softwarebibliotheken. So wird die *PROJ.4* Bibliothek zur Umrechnung von Projektionen benötigt sowie die *GDAL* und *OGR* Bibliotheken zum Austausch von Geodaten mit anderen Programmen. Es bestehen noch weitere optionale Abhängigkeiten zu Datenbanken, mathematischen und grafischen Systembibliotheken.

Das GIS GRASS gliedert sich intern in Bibliotheken, welche Funktionen für den Zugriff auf die GRASS Datenbank zur Verfügung stellen sowie Anwendungsprogrammen, die auf den Bibliotheken aufsetzen. Die Bibliotheken sind ausschliesslich in ANSI C[6] implementiert. Ein Großteil der Module sind ebenfalls in C programmiert sowie in der Skriptsprache der *bash*[7], die so genannten GRASS Shell Skripte. In Abbildung 5.1 ist ein grober Aufbau der GRASS Programm Struktur dargestellt.

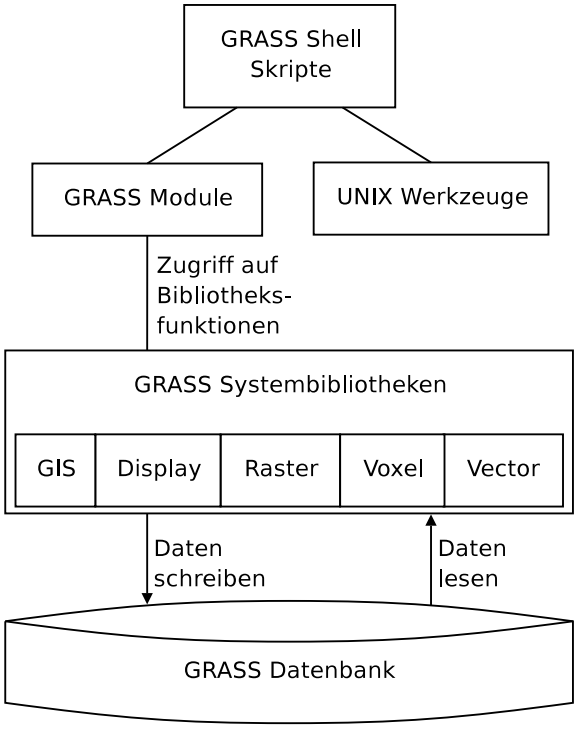


Abbildung 4.4: Programmaufbau des GIS GRASS.

GRASS stellt ein dokumentiertes *application programming interface*, kurz API, für den

[6]C ist eine strukturierte Programmiersprache, die Anfang der siebziger Jahre entwickelt wurde, um auf einem PDP11 Mikrorechner das Betriebssystem Unix zu implementieren. Sie ist die am weitesten verbreitete strukturierte Programmiersprache.

[7]Die bash ist die Standard Shell auf den meisten Unix Betriebssystemen.

Zugriff auf die Bibliotheksfunktionen zur Verfügung. Zur Beschreibung der API gibt es ein *GRASS programmers manual*, welches mittels des Dokumentationssystems Doxygen direkt aus dem Quellcode erzeugt wird.

Aufgrund der Struktur von GRASS gibt es verschiedene Ebenen der Programmierung. Zum einen stehen dem Benutzer verschiedene Skriptsprachen zur Verfügung, zum anderen kann ein fortgeschrittener Benutzer bestehende Module verbessern und erweitern sowie eigene hoch performante C-Module implementieren. GIS Experten mit sehr guten Programmierkenntnissen sind in der Lage, die GRASS Bibliotheken zu warten und neue Funktionen hinzuzufügen.

4.3.1 Skriptprogrammierung

UNIX Shell Skripte können zur Automatisierung von Prozessen benutzt werden. Es sind ASCII Dateien, die mit einem beliebigen Editor erstellt werden können. GRASS bringt bereits eine große Anzahl von Skripten mit. Jedes GRASS Modul kann innerhalb eines Skriptes aufgerufen werden. Dadurch läßt sich ein hoher Grad an Komplexität erreichen. Desweiteren stellt GRASS Funktionen zur standardisierten Benutzerführung zur Verfügung. Infolgedessen haben Shell Skripte das gleiche Interface wie in Module, die in der Programmiersprache C implementiert wurden. In Shell Skripten können viele nützliche UNIX Programme wie *awk, sed, grep, cut* usw. verwendet werden. Für die Skriptprogrammierung in GRASS gelten spezielle Richtlinien, die die Kompatibilität zu den bestehendem Modulen und der Benutzerschnittstelle sicherstellen.

Durch die Skriptfähigkeit von GRASS besitzen Benutzer, die keine C Programmierkenntnisse haben, die Möglichkeit, eigene Programme zu entwickeln und komplexe Berechnungs- oder Auswertungsprozesse zu automatisieren. Ein Nachteil ist, dass die UNIX Shell einen recht begrenzten Sprach- und Funktionmsumfang besitzt und keinen direkten Zugriff auf Bibliotheksfunktionen erlaubt. Außerdem ist die Ausführungsgeschwindigkeit von Algorithmen wesentlich langsamer als in C Programmen.

4.3.2 Programmierung von Modulen und Bibliotheksfunktionen

Grass bietet eine C API mit Zugriff auf hunderte von Bibliotheksfunktionen. Implementiert sind zum Beispiel das Lesen und Schreiben von Raster-, Vektor- und Volumendaten, Flächen- und Entfernungsberechnungen von georeferenzierten Daten in unterschiedlichen Projektionen sowie Funktionen zur Handhabung von Attributen, Datenbankabfragen und der Visualisierung von Karten. Die Bibliotheksfunktionen beginnen immer mit einem Präfix, der auf die Funktionalität hinweist. Die GRASS API ist folgendermaßen strukturiert:

- GIS library (Präfix G_): GRASS Dateiverwaltung, Speicherverwaltung, Parser, Projektionen, Metadatenverwaltung, Zugriff auf Rasterdaten
- vector library (Präfix Vect_): Verwaltung von Vektordatentypen wie Punkte-, Linien- und Flächen sowie deren toplogische und Metadaten Informationen
- database library (Präfix db_): Abstrakte Zugriffsschicht auf verschiedene an GRASS gekoppelte SQL Datenbanken
- g3d library (Präfix G3d_): Verwaltung von Volumendaten und deren Metainformationen, Zugriff auf Volumenzellen und 3d Tiles
- image data library (Präfix I_): Bilderverarbeitung und Dateimanagement
- display library (Präfix D_): Grafische Ausgabe von Raster und Vektordaten
- raster library (Präfix R_): Ausgabe von Rasterdaten auf grafikfähigen Geräten
- gpde library (Präfix N_): Bibliothek zur numerischen Lösung partieller Differentialgleichungen

Die Liste ist nur eine Auswahl an verfügbaren Bibliotheken. Desweiteren gibt es noch Bibliotheken die Funktionen zur Datumsberechnung, Vektor- und Matrizenalgorithmen, Quadtree und Octree Implementierungen usw. zur Verfügung stellen.

Um neue Bibliotheksfunktionen hinzuzufügen oder bestehende zu warten, benötigt man tiefgreifende C Programmierkenntnisse sowie ein Verständnis für professionelles Softwaredesign. Man sollte mit den grundlegenden Algorithmen der Geoinformatik ebenso vertraut sein, wie mit der Konzeption einer API.

Die Programmierrichtlinien von GRASS orientieren sich stark an denen der GNU Free Software Foundation FSF[8]. Desweiteren gibt es klare Richtlinien für das Kompilieren, das Testen von Modulen, das Erstellen von Dokumentation und zur Namensgebung von Modulen. Implementiert wird im ANSI C Standard um plattformübergreifende Kompatibilität zu gewährleisten.

[8] http://www.gnu.org/prep/standards/html_node/index.html

5 Ergebnisse

5.1 Realisierung des Strömungs- und Transportmodells

Basierend auf dem modularen Aufbau von GRASS wird eine Konzept entwickelt, mit dem der Strömungs- und Stofftransport sowohl separat als auch gekoppelt berechnet werden kann. Zur Verwendung kommen alle Ebenen der GRASS Programmierung, die im vorherigen Kapitel vorgestellt wurden. Es sind Bibliotheksfunktionen, C Module sowie Shell Skripte für die Modellierung entworfen und implementiert worden, siehe Abschnitt 5.1 auf der nächsten Seite. Der numerischen Kern der Strömungs- und Transportberechnung ist als Erweiterung für die *gpde* Bibliothek konzipiert. Somit läßt sich das numerische Modell durch Bibliotheksfunktionen in jedes beliebige Modul einbinden. Basierend auf der erweiterten numerischen Bibliothek *gpde*, wurden zwei C-Programme entwickelt. Das Modul *r.gwflow* für die Berechnung der Grundwasserströmung, das Modul *r.solute.transport* für die Berechnung des Stofftransportes. Beide Module verarbeiten Rasterdaten. Somit können alle in GRASS verfügbaren Rastermodule für das Prä- und Postprozessing verwendet werden. Die Kopplung der Strömungs- und Transportmodule sowie weiterer benötigter Raster- oder Vektormodule, wird mit Hilfe verschiedener Shell Skripte realisiert. Im weiteren sei auf die Implementierung der einzelnen Komponenten eingegangen.

5.1.1 Erweiterung der gpde Bibliothek

Die in GRASS implementierte *gpde* Bibliothek, stellt ein einfach zu benutzendes Interface[1] zur Berechnung partieller Differentialgleichungen zur Verfügung. Sie bringt Funktionen zum Einlesen und Verarbeiten von Rasterdaten mit sich, sowie zur Assemblierung linearer Gleichungssysteme und deren Lösung. Der prinzipielle Ablauf einer numerischen Berechnung in GRASS unter Verwendung der *gpde* Funktionalität ist ist in Abbildung 5.2 dargestellt. Der Ablauf besteht meist aus folgenden Komponenten:
Zuerst werden die für die Berechnung benötigten Rasterdaten eingelesen. Diese können anschließend mit *gpde* Bibliotheksfunktionen weiter aufbereitet werden. Wenn die Datenbasis erstellt ist, kann das lineare Gleichungssystem assembliert werden. Die Assemblierungsfunktion ruft dann für jede zu benutzende Rasterzelle eine so genannte Callback

[1] Diese Behauptung resultiert aus persönlicher Erfahrung.

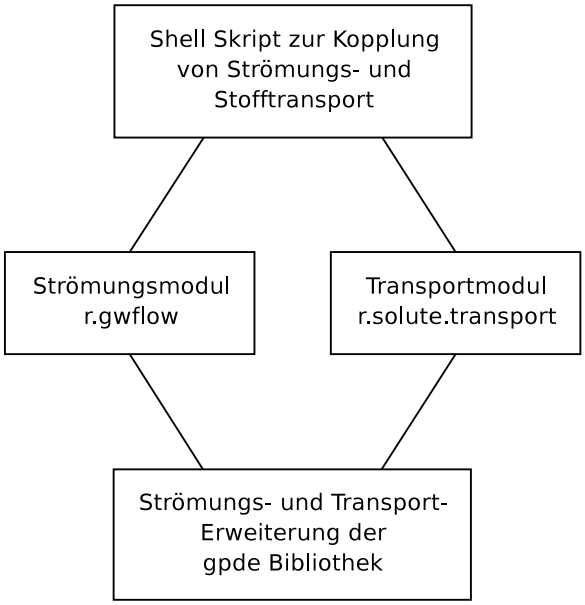


Abbildung 5.1: Das Modulkonzept des gekoppelten Strömungs- und Stofftransportmodells in GRASS.

Funktion auf. In dieser Funktion müssen numerischen Algorithmen definiert werden, die die Einträge in der Massen- und Steifigkeitsmatrix sowie im Vektor b erzeugen. Dieser Callback wird zur Laufzeit gesetzt. Anschließend wird das erstellte lineare Gleichungssystem mit einem in der *gpde* Bibliothek verfügbaren Gleichungslöser gelöst. Die Ergebnisse werden dann als Rasterdatei in der GRASS Datenbank gespeichert.

Basierend auf den Finite Volumen Formulierungen des Strömungs- und Stofftransportes, wurden zwei Callback Funktionen entwickelt und der *gpde* Bibliothek hinzugefügt. Diese Funktionen stellen eine konkrete Realisierung der Gleichungen (3.26) und (3.52) in zwei Dimensionen dar. Sie erstellen die Massenbilanzen für jede Rasterzelle und somit die Matrix- und Vektoreinträge im linearen Gleichungssystem. Die Erweiterung orientiert sich streng an den Vorgaben der *gpde* Bibliothek und ist gut dokumentiert. Sie wurde unter anderem auf verschiedenen Hochleistungscomputern, wie der SGI ALtix 3700 der TU Berlin, auf parallele Verarbeitung getestet und validiert. Sie ist offizieller Bestandteil der momentanen experimentellen Entwicklerversion des GIS GRASS und somit online für jeden Verfügbar. Die Erweiterung ist, wie das GIS GRASS, unter der GPL lizensiert. Der Quelltext inklusive Doxygen-Dokumentation der Erweiterung der numerischen Bibliothek ist dem Anhang A beigefügt. Zusätzlich wurden Unit- und Integrations-Tests der*gpde* Test-Suite zur automatischen Validierung der implementierten Algorithmen hinzugefügt.

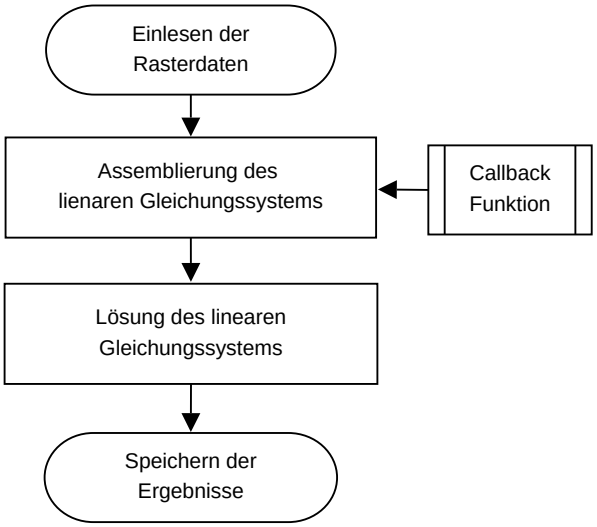


Abbildung 5.2: Prinzipieller Ablauf einer numerischen Berechnung in der *gpde* Bibliothek.

5.1.2 Das Grundwassermodul r.gwflow

Das Modul *r.gwflow* ist für die Berechnung von stationären/transienten, gespannten und ungespannten Grundwasserverhältnissen in zwei Dimensionen konzipiert. Es ist in der Programmiersprache C implementiert und nutzt die *gpde* Bibliothek um die Strömungsgleichung (1.8) zu lösen. Die Implementierung ist ANSI C konform und orientiert sich an modernen Softwaredesign Konzepten. Das Modul kann sowohl von der Kommandozeile aus, als auch über das automatisch generierte Benutzerinterface, bedient werden. Folgende physikalischen Parameter, Anfangs- und Randbedingungen müssen als Rasterdaten bereitgestellt werden, siehe Abbildung 5.4:

- die initialen Grundwasserhöhen
- die obere und untere Begrenzung des Grundwasserleiters
- den X- und Y-Anteil des Tensors der Durchlässigkeitsbeiwerte
- Brunnen und innere Quellen oder Senken
- die Neubildung
- der spezifische Speicherkoeffizient
- die aktiven-, inaktiven- und DIRICHLET-Zellen

Zusätzlich sind noch der Typ des Grundwasserleiters und der Zeitschritt der Berechnung anzugeben. Erzeugt wird eine Rasterkarte mit neuen Grundwasserhöhen für den

gewählten Zeitpunkt. Optional können noch der Typ des Gleichungslösers sowie die Art des Gleichungssystems angegeben werden.

Der interne Programmablauf von *r.gwflow* gliedert sich in folgende Schritte:
Nach dem Programmstart werden zuerst die vom Benutzer angegebenen Programmparameter ausgewertet. Dann werden die zur Verfügung gestellten Rasterdaten mittels Funktionen der *gpde* Bibliothek eingelesen und verarbeitet. Zur Verarbeitung gehört unter anderem das Setzen der natürlichen NEUMANN Randbedingungen. Falls der Grundwasserleiter gespannt ist, wird das Gleichungssystem aufgestellt und gelöst. Anschließend werden die Ergebnisse aus dem Gleichungssystem in eine Rasterkarte übertragen. Falls der Grundwasserleiter ungespannt ist, erfolgt eine etwas andere Berechnung. Das Aufstellen des Gleichungssystems und dessen Lösung wird dann so oft wiederholt, bis der geometrische Fehler der Höhe der Grundwasseroberfläche zwischen zwei aufeinanderfolgenden Iterationen klein genug ist. Dann werden diese Ergebnisse in eine Rasterkarte geschrieben. Das Schema der Implementierung zur Berechnung der Grundwasserhöhen mit *r.gwflow* ist in Abbildung 5.3 dargestellt. Ein typischer Aufruf des Modules von der Kommandozeile ist von folgender Art:

```
r.gwflow --o -s solver=cg top=top bottom=bottom phead=phead\
  status=status hc_x=hydcond hc_y=hydcond \
  q=null s=syield r=recharge output=gwresult\
  dt=8640000000000 type=confined
```

Zum Zweck einer Onlinedokumentation wurde eine *manpage* erstellt, die die Parameter des Modules erleutert und die Handhabung anhand von Beispielen aufzeigt. Der Quelltext befinden sich in Anhang B. Das Modul *r.gwflow* ist seit März 2007 offizieller Bestandteil der experimentellen Entwicklerversion des GIS GRASS. Wie das GIS GRASS ist *r.gwflow* unter der GPL lizenziert.

5.1.3 Das Transportmodul r.solute.transport

Für die Berechnung von transienten Stofftransport in zwei Dimensionen mit räumlich variablen Retardationsfaktor -ohne Stoffabbau- wurde das Modul *r.solute.transport* konzipiert. Es löst die Transportgleichung (7.1). Wie *r.gwflow* ist es in der Programmiersprache C implementiert und nutzt für die numerische Berechnung der Stoffkonzentrationen die *gpde* Bibliothek. Die Implementierung ist ANSI C konform und orientiert sich an modernen Softwaredesign Konzepten. Das Modul kann sowohl von der Kommandozeile aus, als auch über das automatisch generierte Benutzerinterface, bedient werden. Die als Rasterdaten zur Verfügung zu stellenden physikalischen Parameter, Anfangs- und Randbedingungen sind folgende:

- die initialen Grundwasserhöhen

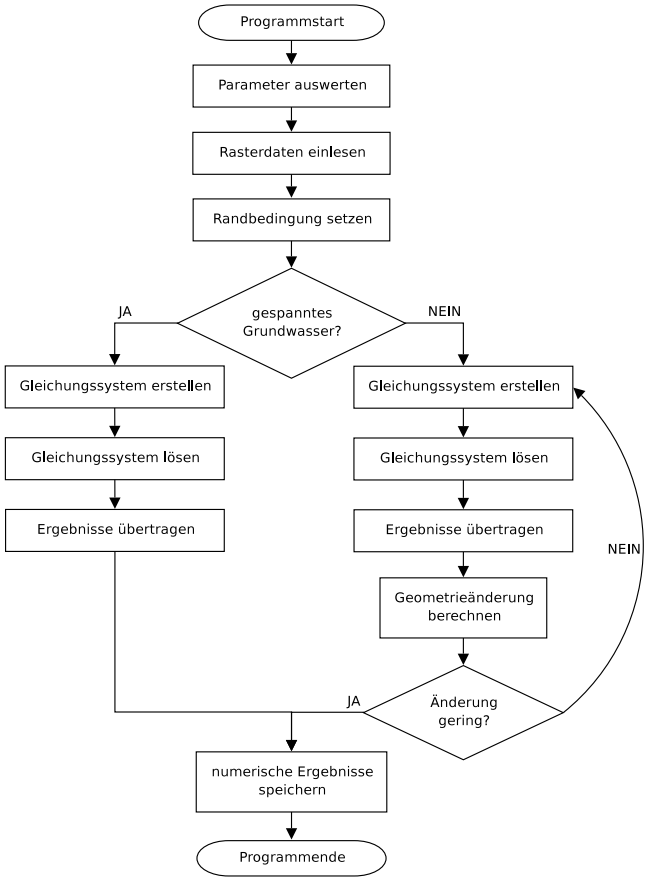


Abbildung 5.3: Flussdiagramm des r.gwflow Moduls.

- die initialen Stoffkonzentrationen
- die obere und untere Begrenzung des Grundwasserleiters
- den X- und Y-Anteil des Tensors der Durchlässigkeitsbeiwerte
- den X- und Y-Anteil des Diffusionstensors
- Brunnen und innere Quellen oder Senken des Strömungsmodells
- innere Quellen oder Senken der Stoffkonzentrationen
- der Retardationsfaktor
- die effektive Porosität
- die aktiven-, inaktiven-, DIRICHLET- und Transmissions-Zellen

Desweiteren müssen die longitudinalen und transversalen Dispersivitätslängen und die Zeitschrittweite angegeben werden. Der Benutzer kann außerdem wählen, ob das

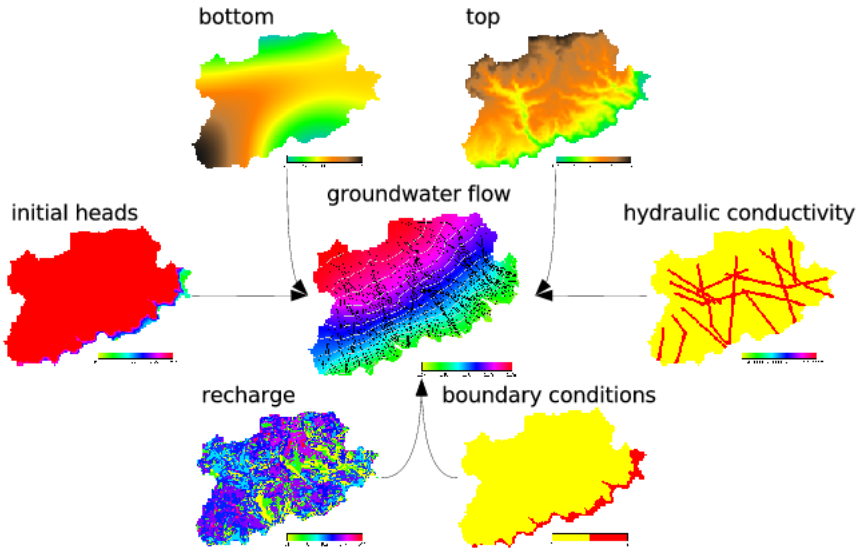


Abbildung 5.4: Eingabekonzept von r.gwflow [Geb07].

Courant-Kriterium eingehalten werden soll. Erzeugt wird eine Rasterkarte mit den räumlichen Stoffkonzentrationen für den gewählten Zeitschritt. Optional können noch der Typ des Gleichungslösers sowie die Art des Gleichungssystems angegeben werden.

Der Programmablauf von *r.solute.transport* ist ähnlich dem von *r.gwflow* und gliedert sich wie folgt:
Nach dem Programmstart werden zuerst die vom Benutzer angegebenen Programmparameter ausgewertet. Dann werden die zur Verfügung gestellten Rasterdaten mittels Funktionen der *gpde* Bibliothek eingelesen und aufbereitet, so daß man aus ihnen ein lineares Gleichungssystem erstellen kann. Dazu gehört unter anderem das Setzen der natürlichen Neumann Randbedingungen sowie die Berechnung des Strömungsfeldes basierend auf den zur Verfügung gestellten initialen Grundwasserhöhen, der effektiven Porosität und den Tensorkomponenten der Durchlässigkeitsbeiswerte. Basierend auf diesen Daten sowie den Dispersivitätslängen wird das Tensorfeld der Dispersion berechnet. Dann erfolgt die Auswertung des Courant-Kriteriums, speziell dem Courant-Fridrich-Lewy-Kriterium, kurz CFL. Dieses Kriterium berechnet die Zeitschrittlänge die optimal zu der gewählten räumlichen Diskretisierung und dem berechneten Strömungsfeld passt. Das Einhalten des CFL Kriteriums ist nicht zwingend bei impliziten Verfahren. Jedoch können bei zu großen Zeitschritten Oszilationen oder Singularitäten entstehen. Desweiteren gewährleistet das CFL Kriterium eine geringe numerische Dispersion. Falls das Kriterium eingehalten wird, erfolgt die Aufstellung und Lösung des linearen Gleichungssystems. Anschließend werden die Ergebnisse aus dem Gleichungssystem übertragen und als Rasterkarte gespeichert. Bei einer Nichteinhaltung des CFL Kriteriums wird eine bestimmte Anzahl neuer Zeitschritte berechnet, die das CFL Kriterium erfüllen und deren Summe den vorgegebenen Zeitschritt ergibt. In einer Schleife wird nun, basierend auf den neuen Zeitschritten, das Gleichungssystem so oft aufgestellt, gelöst und die Ergebnisse übertragen, bis die Anzahl der zu berechnenden Zeitschritte erreicht wird und die

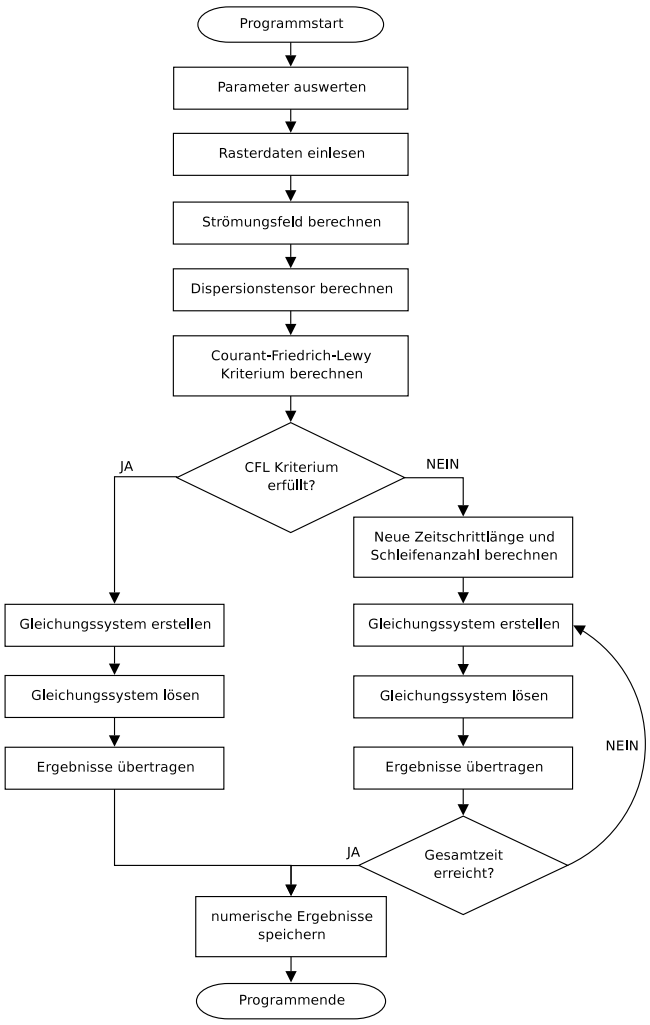


Abbildung 5.5: Flussdiagramm des r.solute.transport Moduls.

Summe der Zeitschritte den Benutzervorgaben entspricht. Die Ergebnisse werden als Rasterkarte gespeichert. Das Schema der Implementierung zur Berechnung des Stofftransports mit *r.solute.transport* ist in Abbildung 5.5 dargestellt. Ein typischer Aufruf des Modules von der Kommandozeile ist von folgender Art:

```
r.solute.transport --o -cs solver=bicgstab top=top bottom=bottom\
  phead=gwresult status=tstatus hc_x=hydcond hc_y=hydcond\
  R=R cs=cs q=null nf=poros output=stresult dt=86400000\
  diff_x=diff diff_y=diff c=c al=100 at=10
```

Neben dem Quelltext wurde eine *manpage* erstellt, die die Parameter des Modules erleutert und die Handhabung anhand von Beispielen aufzeigt. Der Quelltext befinden sich in Anhang C. Das Modul *r.solute.transport* gehört aufgrund seines experimentellen Charakters noch nicht zur offiziellen Entwicklerversion des GIS GRASS.

5.1.4 Shell Skripte zur Kopplung des Strömungs- und Transportmodells

Für die Kopplung des Strömungs- und Transportmodells wurden verschiedene Shell Skripte konzipiert. Der prinzipielle Aufbau eines solchen Skriptes ist in Abbildung 5.6 dargestellt.

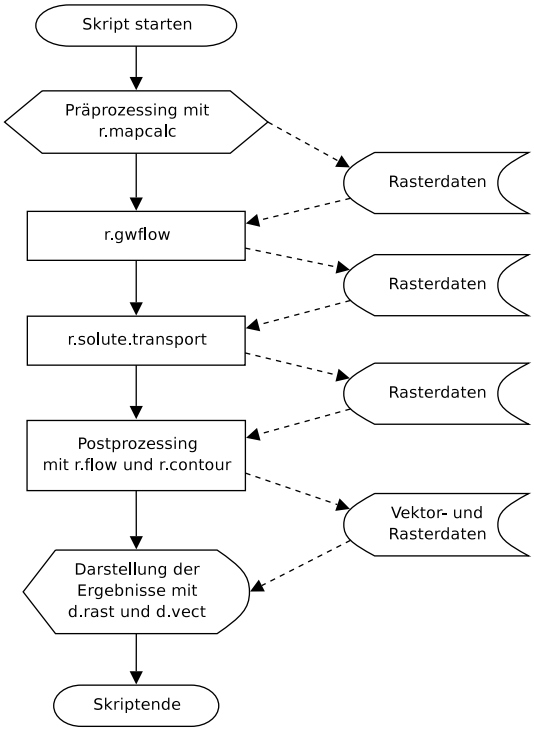


Abbildung 5.6: Prinzipieller Aufbau eines Shell Skriptes. Benutzt werden neben den Strömungs- und Transportmodulen weitere Module für das Prä- und Postprozessing. Die Schnittstelle zwischen den einzelnen Modulen sind die erzeugten Raster- und Vektordaten.

Die Kopplung der Module erfolgt über die erzeugten Daten, also die GRASS Datenbank. Es wurden drei wesentliche Verfahren für die Kopplung von Strömung- und Stofftransport entwickelt, die als Shell Skripte umgesetzt werden können:

Stationäre Kopplung

Im Falle der stationären Kopplung von Strömung und Stofftransport wird zuerst ein stationärer Strömungszustand mit *r.gwflow* berechnet. Basierend auf diesen Strömungsdaten erfolgt die Berechnung des Stofftransportes mit *r.solute.transport* für einen beliebigen, vom Strömungsmodell unabhängigen Zeitschritt. Dies ist die einfachste Möglichkeit der Kopplung. Shell Skripte von diesem Typ entsprechen im wesentlichem

dem Aufbau in Abbildung 5.6. Skripte dieses Typs wurden zur automatischen Erstellung und Berechnung von Validierungsdatensätzen verwendet. Einige Implementierungen dieses Typs befinden sich in Anhang D.

Zeitliche Kopplung

Bei der zeitlichen Kopplung werden die Strömungs und Stofftransportzustände für die gleichen aufeinanderfolgenden Zeitschritte berechnet. Dies wird normalerweise durch eine Schleife realisiert, die über die gewählte Anzahl von Zeitschritten iteriert. Es bietet sich außerdem an, die Zeitschrittlänge variabel zu gestalten. Vor allem bei der Berechnung der Grundwasserströmung sind zunehmende Zeitschrittlängen sinnvoll. In jedem Schleifendurchlauf werden zuerst die Strömungsdaten mit *r.gwflow* für den aktuellen Zeitschritt berechnet. Diese Daten fließen in das darauf folgende Transportmodul *r.solute.transport* ein, welches mit den gleichen Zeitschritt rechnet wie das Strömungsmodell. Dabei regelt *r.solute.transport* bei aktiviertem CFL Kriterium automatisch die Zeitschrittlänge, um Oszilation und unphysikalische Berechnungen zu verhindern. So können dort ebenfalls beliebig große Zeitschritte angegeben werden. Die Zeitschritte sollten bei starken, kurzfristigen Grundwasserspiegelschwankenungen nicht zu groß gewählt werden, damit die Kopplung der Modelle noch physikalisch sinnvoll ist. Zur Erzeugung von Animationen kann in jedem Zeitschritt mittels der GRASS Display-Module, das Zwischenergebnis als Grafik gespeichert oder direkt in einen GRASS Monitor gerendert werden. Somit kann man den Fortschritt der Simulation direkt beobachten. Skripte diesen Typs wurden für die Berechnung von Stofftransport unter Verwendung unterschiedlicher physikalischer Parameter, wie die Dispersivitätslängen, die Neubildungsrate, Brunnenleistungen und Retardationsfaktoren verwendet. Implementierungen dieser Art befinden sich in Anhang D.

Zeitliche Kopplung mit variablen Randbedingungen

Das Modell der zeitlichen Kopplung mit variablen Randbedingungen funktioniert im Prinzip genau wie die *normale* zeitliche Kopplung. Zusätzlich können allerdings für jeden Zeitschritt Randbedingungen wie z.B. die Pumpleistung von Brunnen, Eingabe von gelösten Stoffen in das Grundwasser, die Niederschlagsmenge oder der Retardationfaktor variieren. Scripte dieses Typs befinden sich in Anhang D.

5.2 Validierung und Berechnungen

Um die Korrektheit der implementierten numerischen Algorithmen sicherzustellen, ist eine Validierung mit existierenden Referenzmodellen unerlässlich. Deshalb werden analytische und numerische Beispielrechnungen aus [KR95] und [LKW96] für die Validierung

des Strömungs- und Stofftransportmodells herangezogen. Die verwendeten Validierungsmodelle dienen somit als Referenz und werden im weiteren vorgestellt und die Ergebnisse erläutert. Nach der Validierung der wird ein komplexes Beispielszenario der Kopplung der Strömung und des Stofftransportes vorgestellt.

Die Berechnung der Strömungs- und Stofftransportmodelle erfolgt mittels der implementierten Module *r.gwflow* und *r.solute.transport*. Grundlage für die Modellierung sowie des Prä- und Postprozessing, bildet die neueste Entwicklerversion 6.3 des GIS GRASS.

Für die Modellierung der Beispiele und deren Analyse werden verschiedene GRASS Module verwendet. Die Gittererzeugung und damit die Gebietsdiskretisierung erfolgt mit dem Modul *g.region*. Mit diesem wird die Ausdehnung des Gebietes und die Größe und Anzahl der Elemente festgelegt. Die Modellierung der Parameter-Rasterkarten (z.B. die initiale Konzentrationsverteilung, k_f, q ...) wird mit *r.mapcalc* durchgeführt. Unter der Verwendung der Module *d.rast*, *d.vect*, *d.legend*, *d.barscale*, *d.rast.num* und *d.text* erfolgt die Darstellung der Ergebnisse. Zusätzlich werden zu Analysezwecken Stromlinien und Isolinien mit *r.flow* und *r.contour* erzeugt und zusammen mit den Ergebnissen visualisiert. Für die Berechnung normierter Konzentrationen kommen die Module *r.univar* und *r.mapcalc* zur Verwendung. Die Kombination der Module mittels der Shell Script Programmierung, erlaubt die komplette Automatisierung der Validierungs- und Testrechnungen. Die Modellierung, Berechnung und Visualisierung kann somit ohne Interaktion durch den Benutzer erfolgen. Alle hier verwendeten Shell Scripte sind im Anhang D verfügbar.

5.2.1 Validierung des Strömungsmodells

Die Validierung des Strömungsmodells erfolgt anhand numerischer Beispiele aus dem Standardwerk *Grundwassermodellierung* von Kinzelbach und Rausch [KR95]. Aus der großen Anzahl von Modellen wurden zwei Beispiele ausgewählt, die repräsentativ für bestimmte Strömungssituationen sind und sich leicht nachbilden lassen. Zum einen ist dies ein transientes, gespanntes Grundwassermodell in dessen Mitte sich ein Förderbrunnen befindet, zum anderen wird der stationäre Strömungszustand eines ungespannten Grundwasserleiters berechnet. Die Modelle sind mit Seitenanzahl und Parameterangabe in Tabelle 5.2.1 aufgelistet.

Tabelle 5.1: Parameter der Beispielrechnungen aus [KR95] zur Validierung des Strömungsmodells

	Beispiel 6.1 S.129ff	Beispiel 6.10 S.165ff
Ausdehnung	$700m \cdot 700m$	$2000m \cdot 950m$
Gitterweite $\Delta x \cdot \Delta y$	$100m \cdot 100m$	$50m \cdot 50m$
Mächtigkeit	$20m$	$3m - 5,4m$
S_s	$0,0001\frac{1}{m}$	$0,0001\frac{1}{m}$
k_f	$0,0005\frac{m}{s}$	$0{,}001\ \frac{m}{s}$
Grundwasserverhältnisse	gespannt	ungespannt
Neubildung	0	$0,6 \cdot 10^{-8}\frac{m^3}{s \cdot m^2}$
Pumprate q	$-0,1\frac{m^3}{s}$	keine
Randbedingungen	DIRICHLET, Quellterm	DIRICHLET, NEUMAN
Intervall	$10.000s$	stationär

Validierung anhand des Beispiels 6.1 aus [KR95] S.129ff

Anhand eines einfachen Beispiels wird die Berechnung transienter, gespannter Grundwasserverhältnisse dargestellt und validiert. Das Gebiet umfasst $7 \cdot 7$ Elemente mit einer Gesamtausdehnung von $700m \cdot 700m$. Die Kantenlängen jedes Elementes betragen $100m \cdot 100m$. Am linken und rechten Rand sind DIRICHLET Randbedingungen definiert. Die initialen Grundwasserhöhen betragen im gesamten Gebiet $50m$. Im Zentrum befindet sich ein Brunnen mit einer konstanten Entnahmerate von $-0,1\frac{m^3}{s}$. Die Wasserförderung beginnt zum Zeitpunkt $t = 0$. Die k_f-Wert Verteilung ist homogen und isotrop mit $0,0005\frac{m}{s}$. Der spezifische Speicherkoeffizient ist ebenfalls für das gesamte Gebiet konstant und beträgt $0,0001\frac{1}{m}$. Der Grundwasserleiter ist gespannt und hat eine einheitliche Mächtigkeit von $20m$. Die Berechnung der Grundwasserzustände erfolgt für einen Zeitraum von $10.000s$. Das Ergebnis ist in Abbildung 5.7 auf der nächsten Seite dargestellt. Zusätzlich sind für einen besseren Vergleich die piezometrischen Druckhöhen in jedem Element angegeben. Die mittels *r.gwflow* errechneten Ergebnisse, stimmen exakt mit den von Kinzelbach und Rausch überein. Es ist somit anzunehmen, daß *r.gwflow* für diese Art von Problemstellungen valide Ergebnisse berechnet.

Validierung anhand des Beispiels 6.10 aus [KR95] S.165ff

Berechnet wird der Wasserandrang in einer Baugrube. Mit diesem Beispiel wird die Fähigkeit von *r.gwflow* validiert, ungespannte, stationäre Grundwasserverhältnisse mit Neubildung zu berechnen. Das Gebiet umfasst $40 \cdot 19$ Elemente mit einer Gesamtausdehnung von $2000m \cdot 950m$. Die Kantenlänge jedes Elementes beträgt $50m \cdot 50m$. Am unteren

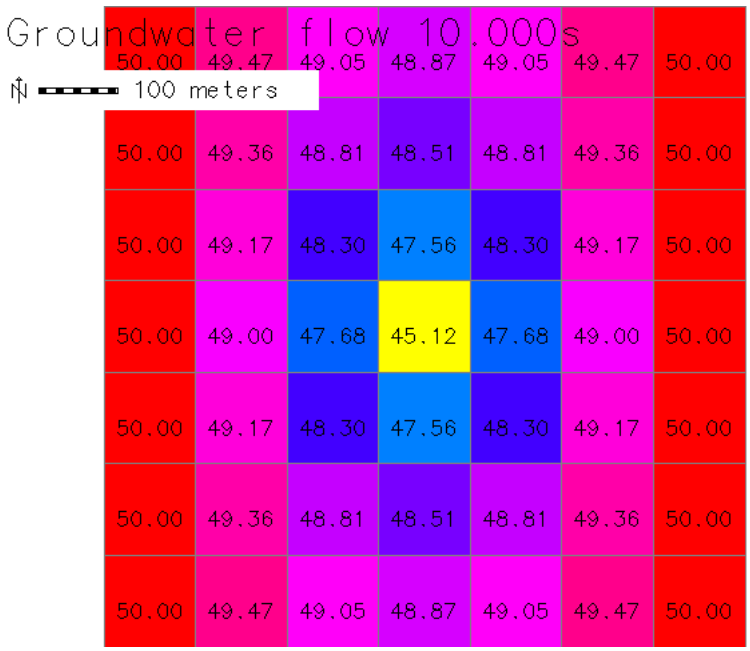


Abbildung 5.7: Ergebnis der Validierungsrechnung des Beispiels 6.1 aus [KR95] S.129ff durch *r.gwflow*. Dargestellt sind die piezometrischen Druckhöhen in m nach $10.000s$ Wasserförderung aus dem Brunnen im Zentrum des Gebietes. Die initialen Grundwasserhöhen sind homogen auf $50m$ gesetzt.

Rand sind DIRICHLET Randbedingungen mit einer Grundwasserhöhe von $5m$ definiert. Die initialen Grundwasserhöhen betragen im gesamten Gebiet $5m$. Die Baugrube am linken Rand wird durch 4 Elemente modelliert in denen DIRICHLET Randbedingungen mit einer Grundwasserhöhe von $3m$ definiert sind. Die k_f-Wert Verteilung ist homogen und isotrop mit $0,001\frac{m}{s}$. Der spezifische Speicherkoeffizient ist mit $0,0001\frac{1}{m}$ angegeben. Der Grundwasserleiter ist ungespannt und hat eine variable Mächtigkeit zwischen $3m - 5,4m$. Die Neubildungsrate beträgt $0,6 \cdot 10^{-8}\frac{m^3}{s \cdot m^2}$ im gesamten Gebiet. Es wird ein stationärer Grundwasserzustand berechnet. Das Ergebnis ist in Abbildung 5.8 auf der nächsten Seite dargestellt. Für den Vergleich des Ergebnisses wurden zusätzlich Isolinien (schwarz) gleicher piezometrischer Druckhöhen sowie Stromlinien (grau) erzeugt. Obwohl das Strömungsmodell im Falle ungespannter Grundwasserverhältnisse, aufgrund der Verletzung der Diskretisierungsvorgaben, einen numerischen Fehler erzeugt der nicht weiter abgeschätzt wurde, stimmen die minimalen und maximalen Druckhöhen exakt mit den von Kinzelbach und Rausch überein. Die dargestellten Isolinien haben ebenfalls die gleiche Position. Es wird also davon ausgegangen, daß *r.gwflow* diese Art von Problemstellung zuverlässig berechnet.

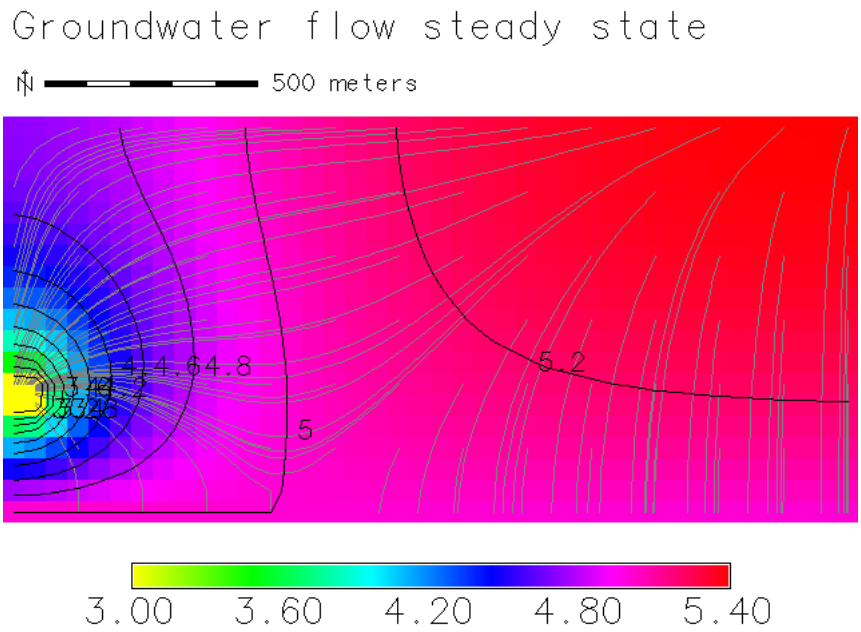


Abbildung 5.8: Ergebnis der Validierungsrechnung des Beispiels 6.10 aus [KR95] S.165ff durch *r.gwflow*. Farbig Dargestellt sind die piezometrischen Druckhöhen in m sowie die Isolinien (schwarz) gleicher Druckhöhen und die Stromlinien (grau). Der Strömungszustand ist stationär.

5.2.2 Validierung des Transportmodells

Das implementierte Transportmodell wird anhand von numerischen und analytischen Beispielen aus *Strömungs und Transportmodellierung* von Lege, Kolditz und Zielke [LKW96] ab Seite 175ff validiert.

Das dem Transportmodell zugrundeliegende Strömungsmodell ist wie folgt aufgebaut: Das Gebiet hat eine Ausdehnung von $2000m \cdot 1000m$, die Gitterweite beträgt $25m \cdot 25m$. Es wird ein gespannter, $25m$ mächtiger, Grundwasserleiter modelliert. Auf der linken und rechten Seite des Gebietes sind DIRICHLET Randbedingungen definiert. Für die Gewährleistung einer konstanten Abstandsgeschwindigkeit von $5,88\frac{m}{s}$, bei einer effektiven Porosität von $0,17$, wird auf dem linken Rand eine Grundwasserhöhe von $276m$ angegeben. Der rechte Rand hat eine Grundwasserhöhe von $50m$. Der Durchlässigkeitsbeiwert ist im gesamten Gebiet $0,0001\frac{m}{s}$. Der Strömungszustand wird stationär berechnet. Im Transportmodell wird die Abstandsgeschwindigkeit durch die Variation der effektiven Porosität gesteuert. Die Grundwasserströmung ist parallel zur X-Achse ausgerichtet.

Alle im weiteren aufgeführten Transportmodelle für die Validierung basieren auf folgende Parametern: Die Gebietsausdehnung und die Gitterweite ist identisch mit dem Strömungsmodell. Modelliert wird der permanente Stoffeintrag in ein zur X-Achse paralleles, stationäres Strömungsfeld. Auf dem linken Rand des Modells wird eine DIRICHLET Randbedingung mit $0\frac{kg}{m^3}$ definiert. Der rechte Rand wird als Transmissionrandbedingung

modelliert, um den ungehinderten Stoffffluss über den Gebietsrand zu gewährleisten. Der Stoffeintrag erfolgt in der Nähe des linken Randes am Element mit dem Index [10,10]. Die Eingabemenge ist variabel. Alle berechneten Konzentrationen werden auf $c_{max} = 1$ normiert.

Für den Vergleich der zwischen den hier berechneten numerischen Ergebnissen und denen im Buch sei angemerkt, daß aufgrund der unterschiedlichen Algorithmen zur Erzeugung der Isolinien gleicher Konzentration, es zu kleinen Abweichungen in der Darstellung der Ergebnisse kommt.

Validierung anhand der Beispiele 1.1 und 1.2 aus [LKW96] S.178ff

Es werden zwei Berechnungen mit jeweils unterschiedlichen Abstandsgeschwindigkeiten durchgeführt. Die Variation der Geschwindigkeit erfolgt über die effektive Porosität. Alle Simulationsparameter sind in Tabelle 6.2 aufgelistet. Die Ergebnisse der Berechnungen mit *r.solute.transport* sind in Abbildung 5.10 und Abbildung 5.11 dargestellt. Abbildung 5.9 stellt das diesen Beispielen zu Grunde liegende Strömungsmodell dar.

Die Ergebnisse zeigen mit den analytischen und numerischen Beispielen aus der Literatur gute Übereinstimmungen. Dabei ist anzumerken, daß eine größere Übereinstimmung mit dem Ergebnis des analytischen Modells zu erkennen ist.

Tabelle 5.2: Parameter der ersten Validierung des Transportmodules *r.solute.transport*. Grundlage sind die Beispiele 1.1 und 1.2 aus [LKW96] S.175ff.

	Beispiel 1.1 aus [LKW96] S.178	Beispiel 1.2 aus [LKW96] S.179
Ausdehnung	$2000m \cdot 1000m$	$2000m \cdot 1000m$
Gitterweite $\Delta x \cdot \Delta y$	$25m \cdot 25m$	$25m \cdot 25m$
Mächtigkeit z	$25m$	$25m$
k_f	$0,0001\frac{m}{s}$	$0,0001\frac{m}{s}$
n_f	$0,17$	$1,0$
v_a	$5,88\frac{m}{d}$	$1,0\frac{m}{d}$
Quelle	$120\frac{kg}{d}$	$120\frac{kg}{d}$
α_L	$100m$	$100m$
α_T	$10m$	$10m$
Intervall	$1000d$	$1000d$

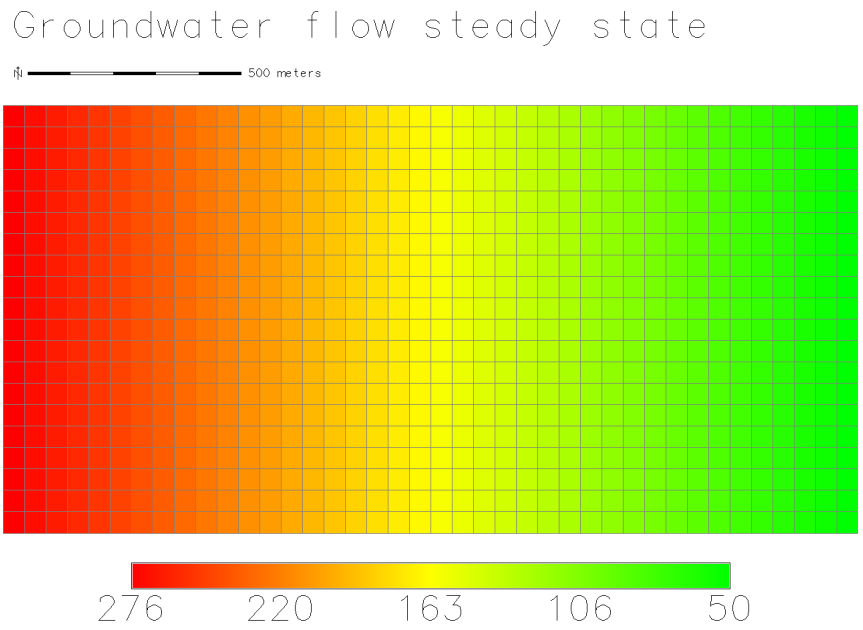


Abbildung 5.9: Strömungsmodell für Beispiel 1.1 und 1.2. Dargestellt ist das stationäre Strömungsmodell mit Standrohrspiegelhöhen in m und das Diskretisierungsgitter (grau).

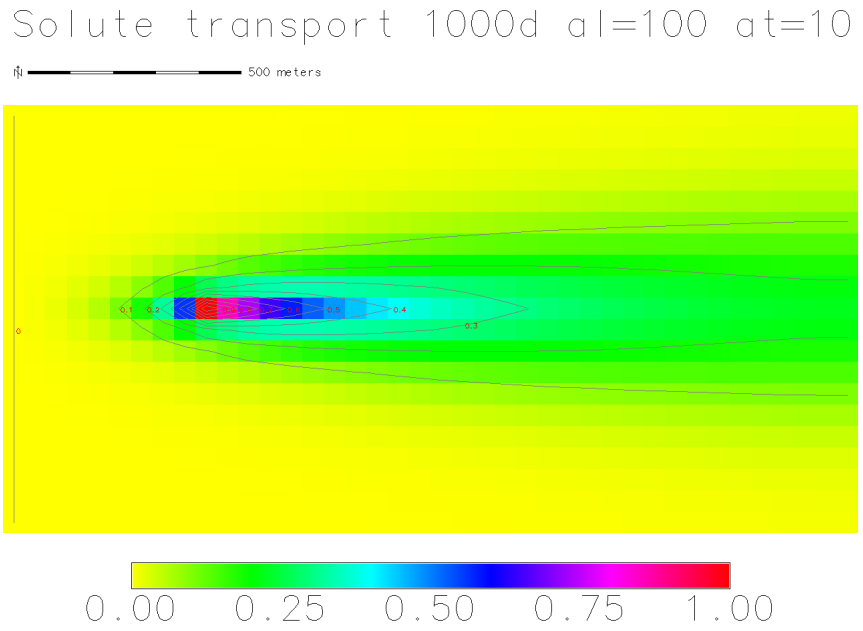


Abbildung 5.10: Ergebnis der Validierungsrechnung des Beispiels 1.1. Dargestellt ist die Konzentrationsverteilung (normiert auf $c_{max} = 1$) nach 1000 Tagen kontinuierlicher Injektion.

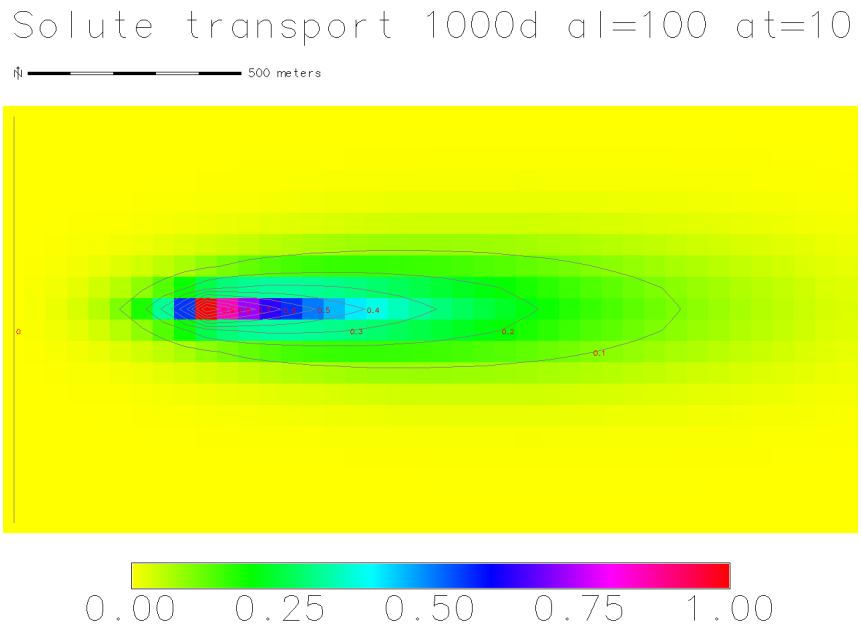


Abbildung 5.11: Ergebnis der Validierungsrechnung des Beispiels 1.2. Dargestellt ist die Konzentrationsverteilung (normiert auf $c_{max} = 1$) nach 1000 Tagen kontinuierlicher Injektion.

Validierung anhand der Beispiele 2.1 und 2.2 aus [LKW96] S.181ff

Der Stoffeintrag erfolgt nun über einen Injektionsbrunnen mit einer Pumprate von $243,5\frac{l}{d}$. Die Menge des eingeleiteten Stoffes beträgt $0,2\frac{kg}{d}$. Alle Simulationsparameter sind in Tabelle 6.3 aufgelistet. Zwei Berechnungen mit jeweils unterschiedlichen Dispersionslängen wurden durchgeführt. Die erste Berechnung erfolgt mit Dispersionslängen von $\alpha_L = 50m$ und $\alpha_T = 5m$. In der zweiten Berechnung werden die Dispersionslängen auf $\alpha_L = 10m$ und $\alpha_T = 1m$ reduziert. Die Ergebnisse der Berechnungen mit *r.solute.transport* sind in Abbildung 5.13 und Abbildung 5.14 dargestellt. In Abbildung 5.12 ist das diesen Beispielen zu Grunde liegende Strömungsmodell dargestellt.

Die Ergebnisse zeigen gute Übereinstimmungen mit den numerischen Beispielen der in [LKW96] S.181 vorgestellten FEM und Random-Walk Methoden. Es ist allerdings zu beachten, dass die Gebietsverfeinerung des FEM und Random-Walk Modells, aufgrund der Gebietsdiskretisierung durch GRASS, nicht modelliert werden konnte.

Tabelle 5.3: Parameter der zweiten Validierung des Transportmodules *r.solute.transport*. Grundlage sind die Beispiele 2.1 und 2.2 aus [LKW96] S.175ff.

	Beispiel aus [LKW96] 2.1 S.181	Beispiel aus [LKW96] 2.2 S.181
Ausdehnung	2000m · 1000m	2000m · 1000m
Gitterweite $\Delta x \cdot \Delta y$	25m · 25m	25m · 25m
Mächtigkeit z	25m	25m
k_f	0,0001	0,0001
n_f	0,17	1,0
v_a	5,88 $\frac{m}{d}$	1,0 $\frac{m}{d}$
Pumprate	243,5 $\frac{l}{d}$	243,5 $\frac{l}{d}$
Quelle	0,2 $\frac{kg}{d}$	0,2 $\frac{kg}{d}$
α_L	50m	10m
α_T	5m	1m
Intervall	250d	250d

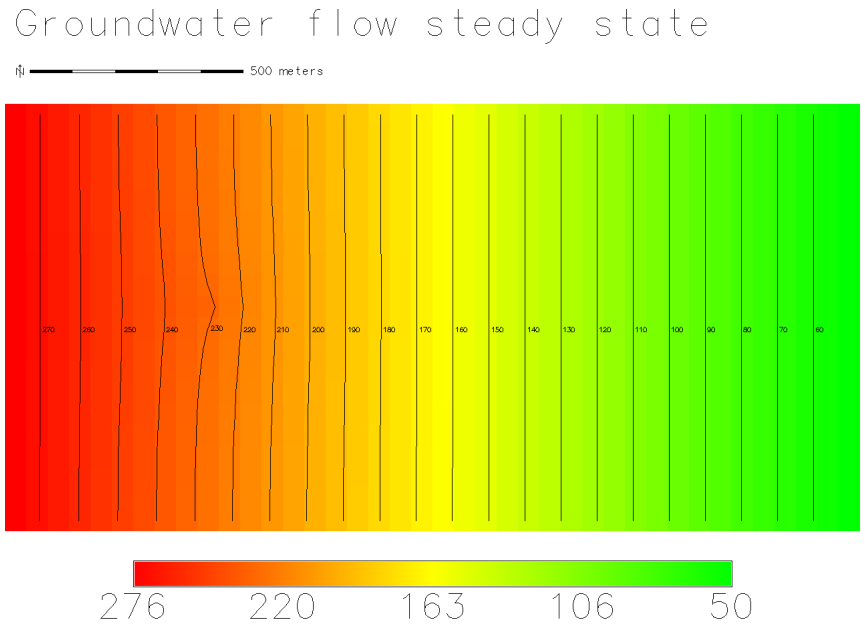


Abbildung 5.12: Strömungsmodell für Beispiel 2.1 und 2.2. Dargestellt ist stationäre Strömungsmodell mit den Standrohrspiegelhöhen sowie deren Isolinien (schwarz).

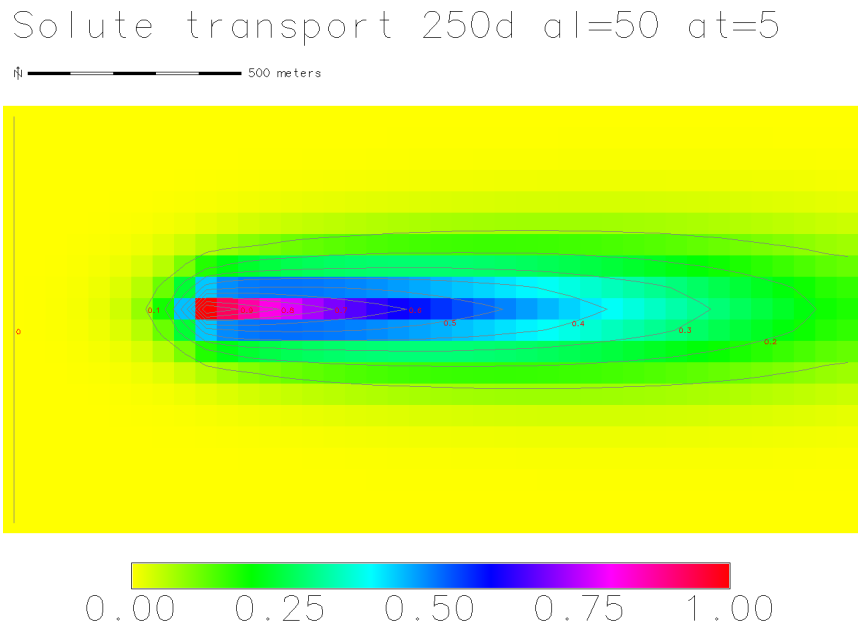


Abbildung 5.13: Ergebnis der Validierungsrechnung des Beispiels 2.1. Dargestellt ist die Konzentrationsverteilung (normiert auf $c_{max} = 1$) nach 250 Tagen kontinuierlicher Injektion, $\alpha_L = 50m$, $\alpha_T = 5m$.

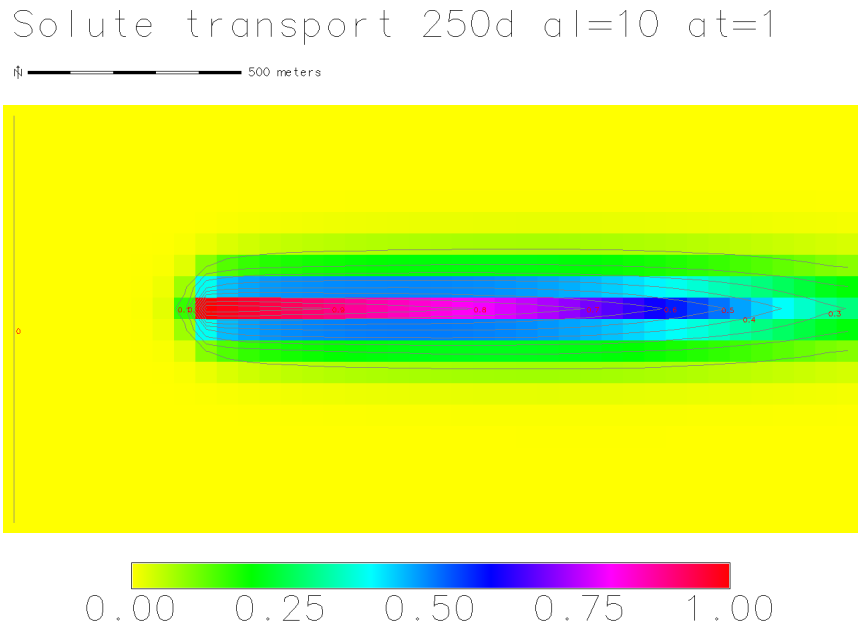


Abbildung 5.14: Ergebnis der Validierungsrechnung des Beispiels 2.2. Dargestellt ist die Konzentrationsverteilung (normiert auf $c_{max} = 1$) nach 250 Tagen kontinuierlicher Injektion, $\alpha_L = 10m$, $\alpha_T = 1m$.

5.2.3 Punktueller Stoffeintrag in ein inhomogenes Strömungsfeld

Aus der großen Anzahl von Modellen die im Rahmen dieser Arbeit berechnet wurden, soll ein Testfall vorgestellt werden, der die zeitliche Kopplung des Strömungs und Transportmodells demonstriert. Dazu wird mittels *r.gwflow* ein inhomogenes Strömungsfeld mit DIRICHLET-Randbedingungen und zwei Förderbrunnen in einem gespannten Grundwasserleiter berechnet. In dieses Strömungsfeld erfolgt eine punktuelle Stoffeingabe. Die Parameter dieses gekoppelten Strömungs und Transportmodells, sind in Tabelle 5.4 aufgelistet. Im weiteren wird detailiert auf die jeweiligen Modelle eingegangen.

Tabelle 5.4: Parameter der Simulation eines punktuellen Stoffeintrages in ein inhomogenes Strömungsfeld.

	Simulationsparameter
Ausdehnung	$200m \cdot 100m$
Gitterweite $\Delta x \cdot \Delta y$	$1m \cdot 1m$
Mächtigkeit z	$20m$
k_f	$0,00005\frac{m}{s}$
n_f	$0,17$
Eintrag	$20\frac{kg}{m^3}$
Grundwasserverhältnisse	gespannt
Neubildung	$0,6 \cdot 10^{-8}\frac{m^3}{s \cdot m^2}$
Pumprate q	$-0,001\ \frac{m^3}{s}$
Intervall	$86400s$
Gesamtzeitraum	$350d$
α_L	$0.1m$
α_T	$0.01m$

Das Strömungsmodell

Das Strömungsmodell beschreibt einen homogenen, isotropen, gespannten Grundwasserleiter mit zwei Förderbrunnen sowie Grundwasserneubildung. Der Strömungszustand wird stationär berechnet. Das Gebiet hat eine Ausdehnung von $200m \cdot 100m$, die Gitterweite beträgt $1m \cdot 1m$. Die Mächtigkeit des gespannten Grundwasserleiters beträgt im gesamten Gebiet $20m$. Der Durchlässigkeitsbeiwert ist konstant $0,00005\frac{m}{s}$, ebenso die effektive Porosität mit $0,17$. Die Neubildung beträgt im gesamten Gebiet $0,6 \cdot 10^{-8}\frac{m^3}{s \cdot m^2}$. Am linken Rand des Gebietes ist eine DIRICHLET Randbedingung mit einer Grundwasserhöhe von $50m$ festgelegt. Die DIRICHLET Randbedingung am linken Rand ist mit variabler Grundwasserhöhe von $47m - 45m$ definiert. Zwei Brunnen im östlichen Teil des Gebietes beeinflussen mit einer konstanten Pumprate von $-0,001\ \frac{m^3}{s}$ das Strömungsfeld.

In Abbildung 5.15 ist das Ergebnis der stationären Strömungsberechnung dargestellt. Die mittels *r.contour* berechneten Isolinien der piezometrischen Druckhöhen und die mit *r.flow* berechneten Stromlinien sind zur Darstellung der Strömungssituation zusätzlich abgebildet. Die Stromlinien verlaufen vom westlichen Rand zum östlichen, oder enden in den Senken der Förderbrunnen. Dieses Grundwassermodell bildet die Grundlage des Stofftransportes. Die in das Transportmodell einfließenden Grundwassergeschwindigkeiten sind in Abbildung 5.16 auf der nächsten Seite dargestellt. Dabei wurden mittels *r.mapcalc* die Absolutwerte der Geschwindigkeiten in x- und y-Richtung aufaddiert.

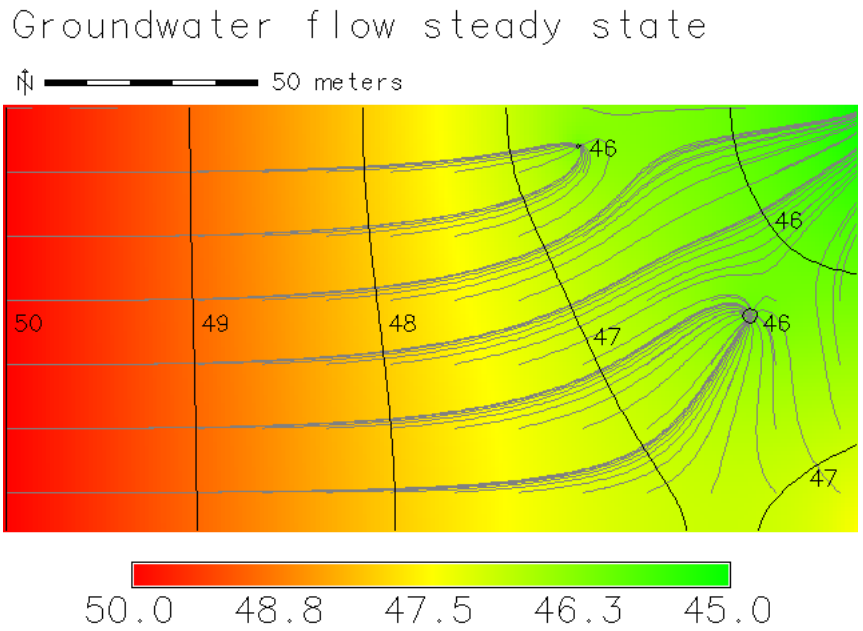


Abbildung 5.15: Stationäre Grundwasserströmung. Dargestellt sind die Standrohrspiegelhöhen in m, die Isolinien (schwarz) der piezometrischen Druckhöhe und die Stromlinien (grau).

Das Transportmodell

Aufbauend auf dem stationären Strömungsmodell wird in der Nähe des linken Randes ein punktueller Stoffeintrag modelliert. Es werden $20kg$ eines nicht reaktiven Stoffes (idealer Tracer) einer einzelnen Rasterzelle zugegeben. Die Rasterzelle hat den Index [15,75] und ist in Abbildung 5.17 auf Seite 88 im Teilbild (a) als kleiner roter Punkt zu erkennen. Der linke Rand ist als DIRICHLET-Rand modelliert mit einer Konzentration von $0\frac{kg}{m^3}$ und liegt direkt auf der DIRICHLET-Randbedingung des Strömungsmodells. Somit strömt unkontaminiertes Wasser in das Gebiet ein. Der rechte Rand ist als Transmissionsrandbedingung konzipiert. Hier ist der ungehinderte advektive und dispersive Fluß über den Rand gewährleistet. Die Anfangskonzentrationen sind, abgesehen von der Eingabestelle, im gesamten Gebiet $0\frac{kg}{m^3}$. Die Brunnen des Strömungsmodells fließt mit

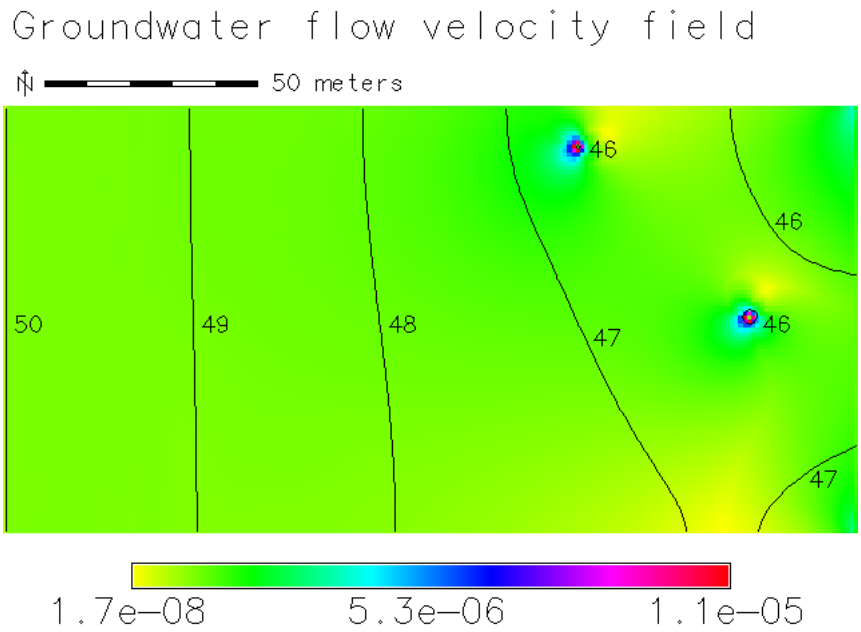


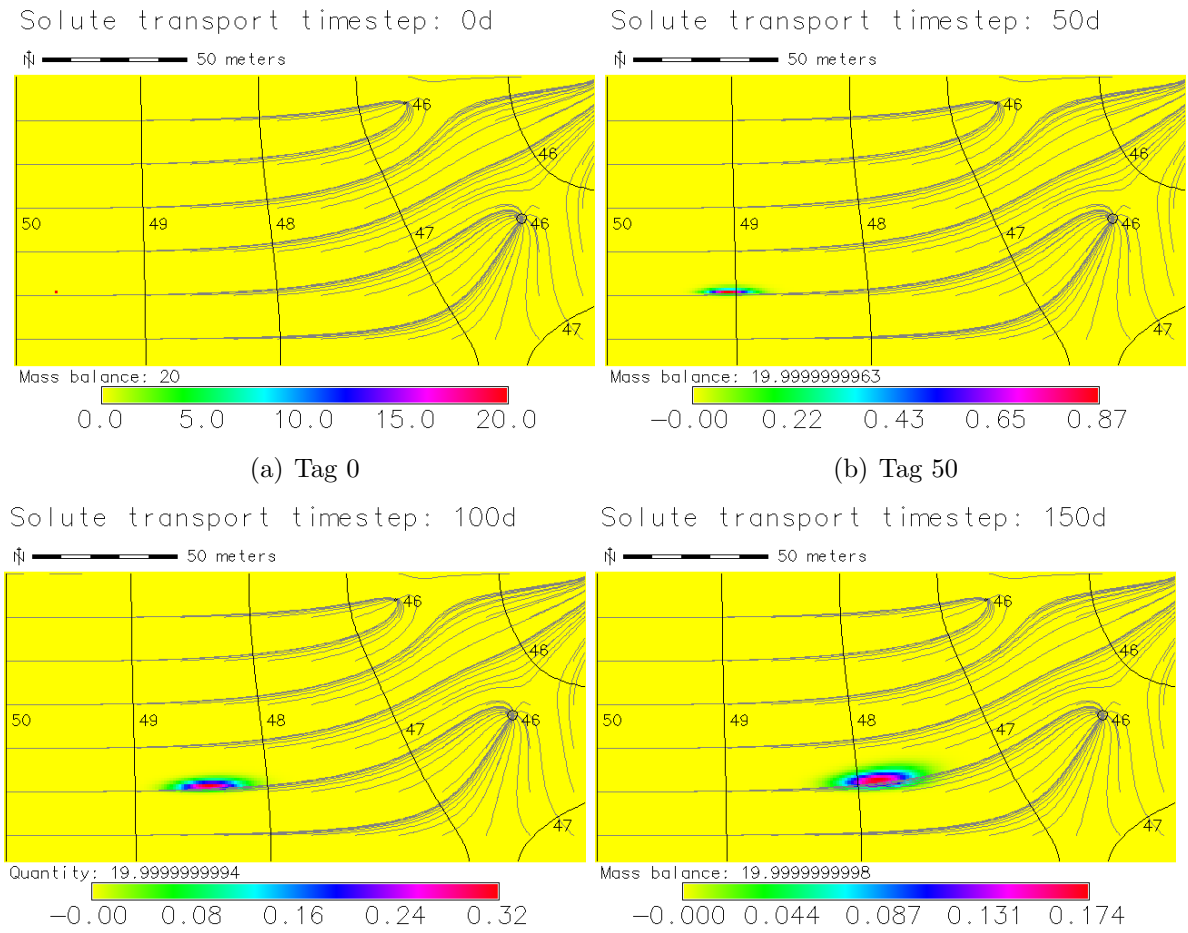
Abbildung 5.16: Filtergeschwindigkeitsfeld der stationären Grundwasserströmung in $\frac{m}{s}$. Dargestellt sind zusätzlich die Isolinien (schwarz) der piezometrischen Druckhöhe.

ihrer Förderleistung in das Transportmodell ein. Dies entspricht einer Entnahme. Der Stofftransport wird für 350 Tage berechnet.

In den Abbildungen 5.17 auf der nächsten Seite und 5.18 auf Seite 89 sind im Abstand von 50 Tagen die Konzentrationen in kg pro Volumeneinheit dargestellt. Für die Berechnung wurde der Gesamtzeitraum von 350 Tagen, in 350 Zeitschritte unterteilt. Ein einzelner Zetischritt hat eine Länge von einem Tag. Zusätzlich wurde das CFL Kriterium in *r.solute.transport* aktiviert. Dadurch werden, abhängig von der Abstandsgeschwindigkeit, die einzelnen Zeitschritte in kleinere Schritte aufgeteilt, Zusätzlich zu den Konzentrationen sind in Abbildung 5.17 auf der nächsten Seite und 5.18 auf Seite 89 die Stromlinien (grau) des Strömungsmodells und die Isolinien (schwarzs) der piezometrischen Druckhöhen dargestellt. Anhand dieser Zusatzinformationen ist gut zu erkennen, wie der Stoff in dem inhomogenen Strömungsfels transportiert wird. Der Stofftransport folgt den dargestellten Stromlinien. Die Stofffahne weitet sich mit zunehmender Entfernung vom Eingabeort auf und wird größtenteils von einem Förderbrunnen aus dem Gebiet heraustransportiert. Der Rest fließt über die Transmissionsrandbedingung am rechten Rand ab. Zusätzlich zu den Konzentrationen, den Strom- und Isolinien ist die Massenbilanz über das gesamte Gebiet in den Abbildungen angegeben.

Es ist zu erkennen, daß es bis zum 150. Tag der Simulation Schwankungen in der Massenblilanz gibt, die weder auf die Fördebrunnen, noch auf die am rechten Rand befindliche Transmissionsrandbedignung zurückzuführen sind. Diese Schwankungen im Bereich von $1 \cdot 10^{-10} kg$ sind vermutlich auf die numerische Dispersion, Oszilationen sowie nicht zu vermeidende Rundungsfehler zurückzuführen. Allerdings ist der Fehler so gering, daß

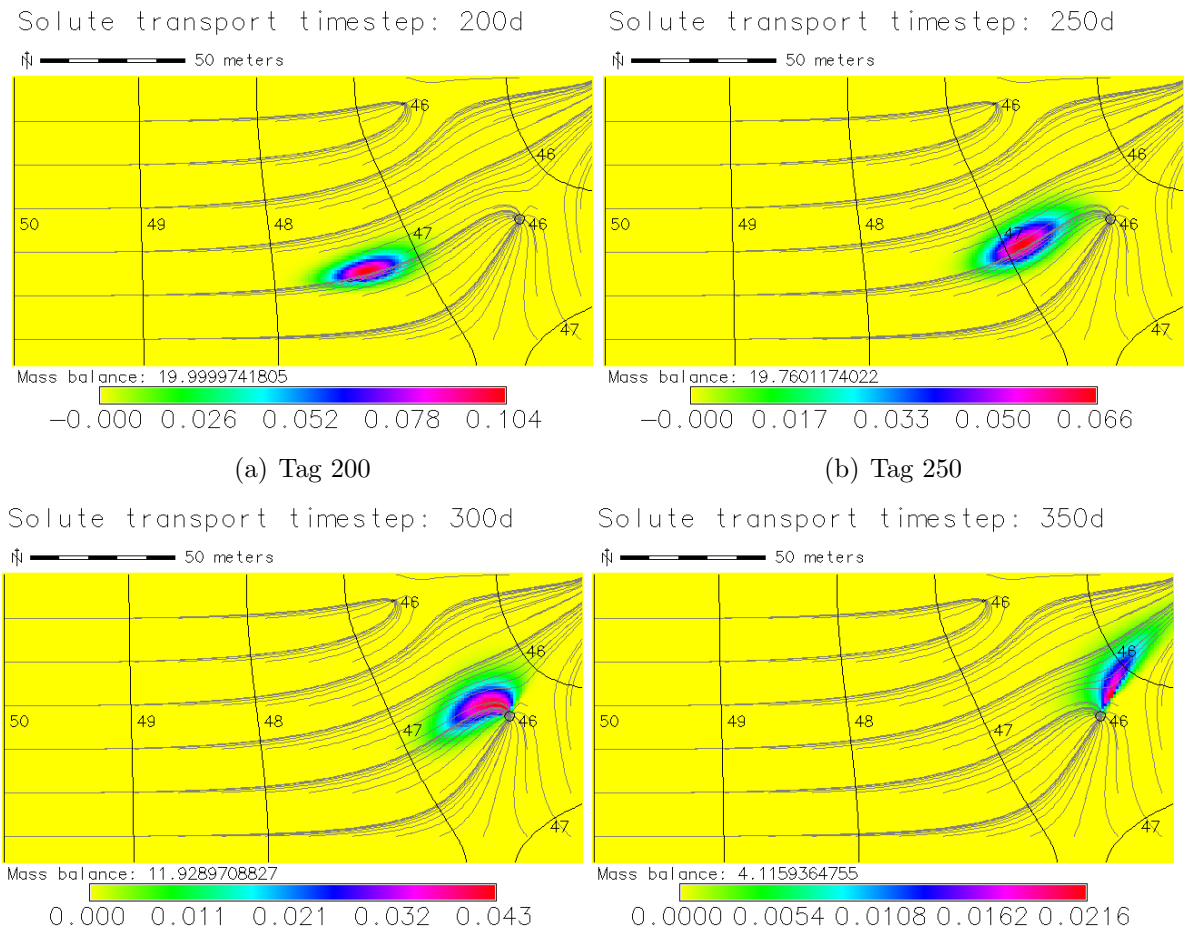
er meiner Meinung nach unterhalb der Nachweisgrenze vieler im Grundwasser gelösten Stoffe liegt und somit nicht relevant ist. Ab dem 250. Tag ist eine deutliche Abnahme der Gesamtmasse zu erkennen, was auf die Förderung durch den Brunnen in der Mitte des rechten Randes zurückzuführen ist. Bis zum 350. Tag der Simulation haben $\frac{3}{4}$ der Gesamtmasse das Gebiet über einen Förderbrunnen, oder die Transmissionsrandbedingung verlassen.



(a) Tag 0 (b) Tag 50

(c) Tag 100 (d) Tag 150

Abbildung 5.17: Simulation über 350-Tage mit punktuellem Stoffeintrag und zwei Sanierungsbrunnen. Dargestellt sind die Konzentrationen in kg, die Isolinien der Druckhöhen sowie die Stromlinien der ersten 150 Simulationstage.



(a) Tag 200

(b) Tag 250

(c) Tag 300

(d) Tag 350

Abbildung 5.18: Simulation über 350-Tage mit punktuellem Stoffeintrag und zwei Sanierungsbrunnen. Dargestellt sind die Konzentrationen in kg, die Isolinien der Druckhöhen sowie die Stromlinien der Simulationstage 200 bis 350.

6 Diskussion

6.1 Die mathematische Formulierung des Strömungs- und Stofftransports

Die hergeleitete Finite Volumen Formulierung basiert auf strukturierten Rechtecksgittern und orientiert sich somit exakt an dem Raster- und Volumenformat von GRASS. Dies garantiert eine konsistente Modellierung hydrogeologischer Fragestellungen mit GRASS sowie ein reibungsfreies ineinandergreifen verschiedener existierender hydrologischer Module mit den neuen Strömungs- und Transportmodulen. Die Modellierung erfolgt somit auf der gleichen Datenbasis, ohne Konvertierung oder Interpolation der Berechnungsparameter auf verschiedene Gitter.

Der Nachteil dieses Ansatzes liegt in der fehlenden Möglichkeit, die Gitterstruktur lokal dynamisch an die Problemstellung anzupassen. Was besonders bei Quellen und Senken sowie scharfen Transportfronten numerisch von Vorteil ist. Auf strukturierten Gittern sind lokale Gitterverfeinerungen relativ schwierig zu realisieren. Mit der hier implementierten Methodik, ist dies nicht möglich. Um eine lokale Verfeinerung der Berechnungsgitters zu erreichen, muß das gesamte Netz verfeinert und somit einen wesentlich höheren Rechenaufwand in Kauf genommen werden.

Im Vergleich zu frei verfügbaren und kommerziellen Programmen zur Grundwasser- und Transportmodellierung, fehlen noch einige wichtige Features, wie zum Beispiel Cauchy Randbedingungen. Auch wurde im Transportmodell eine Vereinfachungen getroffen, die für eine gute Approximation der realen Bedingungen unüblich ist. So ist der Dispersionstensor nur auf der Diagonalen besetzt, was zu einer ungenaueren Approximation der hydrodynamischen Dispersion führt. Auch entsprechen die gewählten Stabilisierungsverfahren im Transportmodell, *full upwinding* und *exponential upwinding*, nicht dem aktuellen wissenschaftlichen Stand. Sie stellen jedoch die Berechnung brauchbarer Ergebnisse sicher. Der Nachteil der implentierten Stabilisierungsverfahren ist die erhöhte numerische Dispersion und die lineare Fehlerordnung [Hin03].

Die implementierten Algorithmen reichen jedoch aus komplexe Strömungs- und Transportprozesse zu berechnen. Der numerische Ansatz ist stabil und hat für beide Modelle in der Zeitdiskretisierung eine lineare Fehlerordnung. Die Gebietsdiskretisierung auf Basis der Finite Volumen Formulierung hat eine quadratische Fehlerordnung. Das bedeutet,

dass mit der Halbierung der geometrischen Schrittweite, sich der Fehler um den Faktor vier verringert [Hin03].

Die verwendeten numerischen Gleichungslöser sind modern und im Bereich der Strömungs- und Tranbsportmodellierung Stand der Technik [Mei99].

6.2 GRASS und hydrogeologische Fragestellungen

Während der Ausarbeitung des Konzepts zur Integration eines gekoppelten Strömungs- und Stofftransportmodells in GRASS, stellt sich die Frage, in wie weit sich das GIS GRASS zur Beantwortung und Modellierung hydrogeologischer Fragestellungen eignet.

Wie erwähnt, bietet GRASS eine große Anzahl von Prä- und Postprozessing Modulen zur Aufbereitung und Visualisierung geografischer Daten an. Neben den Möglichkeiten der Digitalisierung von Karten, dem Import von Satelliten- und Luftbildern sowie dem Einlesen und der Interpolation von Messdaten aus Felduntersuchungen oder Bohrprofilen, bietet GRASS weitere Module zur Beantwortung spezieller hydrogeologische Fragestellungen. Einige dieser Module sollen kurz angesprochen werden.

Auf der Basis von Höhenkarten lassen sich mit *r.slope.aspect* Hangneigungs- und Aspekt-Karten berechnen. Mit den Modulen *r.flow* und *r.terraflow* ist es möglich, die Akkumulation des Oberflächenabflusses zu berechnen. Auf der Basis dieser Daten lassen sich Aussagen über das Erosionsrisiko, Hangrutschungen und der Wassersättigung der Bodenzone treffen. Mit *r.watershed, r.basins.fill* und *r.water.outlet* lassen sich Wassereinzugsgebiete und Wasserscheiden bestimmen. Hydrologische Simulationen lassen sich mit *r.sim.sediment, r.sim.water* und *r.topmodel* erstellen. Meist stehen hydrogeologische Parameter nur als stichprobenartige Informationen zur Verfügung. GRASS bietet eine umfangreiche Unterstützung um Punktdaten, die in vielen verschiedenen Formaten vorliegen können, zu importieren, oder bestehende Datenbanken[1] als Datenquelle einzubinden. Es existieren verschiedene Vektormodule, die aus punkthaften Informationen flächenbasierte Interpolationen erstellen. Dafür stehen IDW, nearest neighbour und Spline Interpolationsmethoden zur Verfügung. Mittels der IDW Methode ist es zum Beispiel möglich, Thiesenpolygone zur flächenhaften Verteilung von Niederschlägen zu berechnen. Neben den hier vorgestellten, gibt es eine Vielzahl weiterer Module, die zur Beantwortung hydrogeologischer Fragestellungen verwendet werden können.

Die Kombination der Prä- und Postprozess Module mit den existierenden hydrologischen und hydrogeologischen Modulen zeigt, daß GRASS sehr gute Voraussetzungen zur Modellierung von Strömungs- und Transportproblemen bietet.

[1] Access, Oracle, Postgresql, Mysql, SQlite usw.

6.3 Konzeption

Das gewählte Grundkonzept zur Integration eines Strömungs- und Transportmodells in GRASS, stellt sich als sehr flexibel heraus und läßt Spielraum für Erweiterungen. So ist es möglich aufgrund des modularen Aufbaus und den Skriptfähigkeiten von GRASS, die Strömungs- und Transportmodule auf unterschiedliche Weise miteinander zu kombinieren und somit an verschiedene Modellsituationen anzupassen. Es ist möglich die Post- und Präprozessing Fähigkeiten von GRASS in die Modellierung mit einzubeziehen. Dieses Konzept erlaubt unter anderem die Automatisierung komplexer Prozesse, wie zum Beispiel die Berechnung hydraulischer Verhältnisse aufgrund von Daten aus einem großen, flächenhaften Grundwasser-Monitoring Programm. Die Berechnung und Auswertung kann zeitnah geschehen, wenn die erhobenen Daten online in einer Datenbank gespeichert werden, auf die das GIS GRASS zugreifen kann. Desweiteren besteht die Möglichkeit, Kalibrierungsprozesse automatisch ablaufen zu lassen. Die für diese Arbeit erstellten Shellscripte deuten an, welches Potential in der modularen Konzeption steckt.

Der Nachteil der Modularität und der Verwendung des GIS GRASS besteht darin, das potentielle Benutzer einen großen Einarbeitungsaufwand bewältigen müssen. So ist das GIS GRASS kein triviales, intuitives oder leicht zu bedienendes Programm. Die implementierten Module setzen Spezialwissen und Erfahrung in der Strömungs- und Transportmodellierung voraus. Auch stellt die Skriptprogrammierung den normalen Anwender anfangs vor eine scheinbar unüberwindliche Hürde.[2] Ein weiterer Punkt ist die geringe Verbeitung von GRASS in Deutschland und damit der fehlende Anwenderschaft mit dem entsprechendem Spezialwissen. Die Auslagerung der Kopplung des Strömungs- und Transportmodells auf die Shellscript Ebene führt dazu, dass man für komplexe Kopplungsszenarien die Shellscript Programmierung beherrschen muß.

6.4 Implementierung

Im Rahmen der Implementierung wurde großer Wert auf moderne Programmierverfahren und fortschrittliches Softwaredesign gelegt. Es wurde darauf geachtet, den Programmierstandards der Free Software Foundation FSF[3] zu entsprechen sowie die Qualitätsvorgaben[4] des GRASS Entwicklerteams einzuhalten. Der Quellcode ist dokumentiert und wurde in das Versionsverwaltungssystem von GRASS integriert. Weiterhin wurden *manpages* für die Programme *r.gwflow* und *r.solute.transport* erstellt sowie Tests

[2] Diese Aussage basiert auf meinen Erfahrungen mit GIS Anwendern, die mit Skriptprogrammierung und dem GIS GRASS noch nicht in Berührung gekommen sind, oder gerade dabei waren, sich einzuarbeiten.

[3] Quelle: http://www.gnu.org/prep/standards/html_node/index.html

[4] Quelle: http://grass.itc.it/grass63/source/SUBMITTING

zur automatischen Validierung der numerischen Module implementiert. Für die numerische Berechnung wurde eine in GRASS implementierte Bibliothek verwendet, die die Lösung sehr großer Problemstellungen auf leistungsstarken Großrechnern erlaubt. Dieser Umstand relativiert die in GRASS nicht vorhandene Fähigkeit der lokalen Verfeinerung von Gittern, indem es erlaubt, auf sehr feinen Gittern für das gesamte Berechnungsgebiet zu arbeiten. Deshalb wurde versucht die Module und die Erweiterungen der *gpde* Bibliothek *Thread Safe* zu implementieren, um von der Parallelität der *gpde* Bibliothek zu profitieren. Rechenversuche auf Multiprozessor-Rechnern zeigen, dass dieser Ansatz gelungen ist.

Das Konzept von GRASS-Bibliotheken ermöglicht einem erfahrenen Programmierer, die erstellte Erweiterung der *gpde* Bibliothek in andere Programme zu integrieren. Somit sind weit komplexere Kopplungsszenarien auf der C Programmierebene denkbar, die auf Basis der Shellprogrammierung nur schwer zu realisieren wären.

6.5 Numerische Ergebnisse

Die numerische Validierung des Strömungs- und Transportmodells anhand von Literaturbeispielen zeigt, dass die implementierten Programme die gewählten Referenzmodelle zuverlässig berechnen können. Die Übereinstimmung der Ergebnisse mit den Strömungs- und Stofftransportmodellen ist sehr zufriedenstellend. Im Falle des Strömungsmodells stimmen die Ergebnisse exakt überein. Dies zeigt, dass im Falle der Referenzmodelle sowohl die mathematische FVM Formulierung korrekt ist, als auch die algorithmische Implementierung.

Die minimalen Abweichungen des Transportmodells sind zum einen auf die Vereinfachungen im Modell zurückzuführen und zum anderen auf die unterschiedlichen Darstellungsmethoden und des verwendeten Diskretisierungsgitters.

Es ist anzumerken, dass das implementierte Strömungs- und Transportmodell momentan nur zuverlässige Ergebnisse im Falle der Referenzmodelle liefert. Somit läßt sich keine Aussage darüber treffen, ob dieses Modell auch in anderen Situationen korrekte Ergebnisse liefert. Dies läßt sich nur über weitere Validierungsberechnungen mit weiteren Referenzmodellen sicherstellen.

6.6 Fazit

In dieser Arbeit wird die Konzeption und Integration eines funktionierenden gekoppelten Strömungs- und Transportmodells in das GIS GRASS demonstriert. Das gewählte Konzept integrierte sich in das GIS GRASS und profitiert von den Modellierungsmöglichkeiten des Geoinformationssystems. Hier zeigt sich der Vorteil von freier

Software gegenüber einem geschlossenem Softwaremodell. Während man bei einem geschlossenem Softwaremodell mit einer Art Blackbox arbeitet und keinen Einfluss auf die implementierten Algorithmen hat, oder die Software nach eigenen Bedürfnissen erweitern kann, bietet freie Software den vollen Zugriff auf alle Teile des Programms. Erst durch diese Freiheit ist es möglich, eigene Entwicklungen und Ideen in ein Softwaresystem zu integrieren, Fehler zu beseitigen, oder dieses, wie in diesem Fall, zu erweitern.

6.7 Ausblick

Zukünftige Entwicklungen sollten die Implementierung fehlender Features umfassen, um ein komplettes Grundwassermodellierungsmodell in GRASS zur Verfügung zu stellen. Anschließend sollte die Implementierung auf drei Dimensionen erweitert werden, da GRASS in dieser Hinsicht einzigartige Modellierungsfähigkeiten aufweist. Eine dreidimensionale Strömungs- und Transportmodellierung in GRASS wäre für andere Modellierungsprogramme, die mit GIS Software interagieren, richtungsweisend. Da die mathematischen Grundlagen für den 3D Fall in dieser Arbeit hergeleitet wurden, sollte es nicht allzu schwierig sein, dies in die Tat umzusetzen.

7 Zusammenfassung

In der Hydrogeologie werden zur Modellierung von Strömung und Stofftransport im Grundwasser Kombinationen aus Geoinformationssystemen und numerischen Programmen eingesetzt. Diese Methode bringt den Nachteil mit sich, Daten des GIS in ein zu den Modellierungsprogrammen kompatibles Datenformat umzuwandeln. Ziel dieser Arbeit ist, ein modernes Strömungs- und Transportmodell direkt in ein GIS zu implementieren, um somit Dateninkompatibilität und Datenverlust durch Konvertierung im Modellierungsprozess zu vermeiden und von den Fähigkeiten des GIS zu profitieren. Zusätzlich wird untersucht in wie weit sich das gewählte GIS zur Beantwortung hydrogeologischer Aufgabenstellung eignet. Dabei ist die Auswahl des Geoinformationssystems (GIS) entscheidend. Diese Wahl hat direkten Einfluss auf die Modellierungsfähigkeiten und die zu implementierenden numerischen Methoden. In dieser Arbeit wird das GIS GRASS als Grundlage für die Konzeption und Implementierung gewählt. Aufbauend auf dem Raster und Volumen Datenformat des GIS GRASS werden die numerischen Verfahren hergeleitet und unter Verwendung moderner Konzepte des Softwaredesignes realisiert.

Theoretische Grundlagen

Kapitel 1 Grundlagen der Strömungs- und Stofftransportmodellierung

Grundlage des Transportmodells bildet ein Strömungsmodell, welches zu Beginn hergeleitet wird. Es wird in die Begrifflichkeit der Permeabilität, der Porosität, des spezifischen Speicherkoeffizienten und das Konzept der Durchlässigkeitsbeiwert-Tensoren eingeführt. Das DARCYische Gesetz wird als Grundlage der Strömungsdifferentialgleichung beschrieben, welche anschließend auf Basis eines repräsentativen Kontrollvolumens hergeleitet wird. Die allgemeine Strömungsdifferentialgleichung

$$\boldsymbol{\nabla} \cdot \mathbf{K}_f \cdot \boldsymbol{\nabla} h + q = S_s \frac{\partial h}{\partial t}$$

bildet die Grundlage des Strömungsmodells.

Im zweiten Teil des ersten Kapitels werden die physikalischen Parameter und Gesetzmäßigkeiten des Stofftransportes eingeführt. Auf Basis der Advektion, der molekularen Diffusion sowie der hydrodynamischen Dispersion und des Stoffabbaus wird die

allgemeine Form der Transportdifferentialgleichung hergeleitet. Die Gleichung ist wie folgt definiert:

$$\frac{\partial c}{\partial t} \cdot R = -\mathbf{u} \cdot \boldsymbol{\nabla} c + \boldsymbol{\nabla}(\mathbf{D}^* \boldsymbol{\nabla} c) + \sigma - \frac{q}{n_f}(c - c_{(in)}). \tag{7.1}$$

Neben den Anfangs- und Randbedingungen werden die Lösungsvoraussetzungen für diese Gleichung besprochen.

Kapitel 2 Numerische Grundlagen

Hier wird auf die Grundlagen der verwendeten numerischen Methoden eingegangen. Es werden verschiedene Formen der Gebietsdiskretisierung und Zeitdiskretisierung vorgestellt. Anschließend wird anhand eines Beispiels die *Zellen zentrierte* Finiten Volumen Methode eingeführt. Diese Methode wird im weiteren für die Diskretisierung der Strömungs- und Transportgleichung verwendet.

Methodik

Kapitel 3 Anwendung der numerischen Verfahren

Die Anwendung der numerischen Methoden auf die hergeleiteten Differentialgleichungen erfolgt in diesem Kapitel. Zuerst wird die Gebietsdiskretisierung festgelegt. Diese folgt exakt dem Raster- und Volumendatenformat des GIS GRASS. Es wird also auf strukturierten, äquidistanten Gittern gearbeitet. Als Zeitdiskretisierung wird sowohl für das Strömungsmodell als auch für das Transportmodell ein impliziter Euler Ansatz gewählt. Anschließend erfolgt die Anwendung der *Zellen zentrierte* Finiten Volumen Methode auf die Strömungsdifferentialgleichung. Dabei wird die Annahme getroffen, dass der Tensor der Durchlässigkeitsbeiwerte nur auf der Diagonalen besetzt ist und sich somit an den Achsen des kartesischen Koordinatensystems orientiert. Folgendes lineares Gleichungssystem wird hergeleitet:

$$\sum_{j \in \Lambda_i} (h_j^{(t+\Delta t)} - h_i^{(t+\Delta t)}) \left(\sum_{k \in \beta} |\nu_{ij}^k| \frac{k_{ij}^{kk}}{d_{ij}} \right) m_{ij} + m_i S_i \frac{h_i^{(t+\Delta t)}}{\Delta t} = m_i S_i \frac{h_i^{(t)}}{\Delta t} + m_i q_i \ .$$

Nach der Erläuterung der Integration der Randbedingungen in das lineare Gleichungssystem, erfolgt die Anwendung der *Zellen zentrierte* Finiten Volumen Methode auf die Transportgleichung. Der Diffusions/Dispersionsterm und der Advektionsterm werden jeweils separat betrachtet und anschließend zusammengeführt. Zur einfacheren numerischen Behandlung wird die Annahme getroffen, dass der Diffusions/Dispersionstensor nur auf der Diagonalen besetzt ist. Für die Diskretisierung des Advektionstern wird eine Konvexkombination verwendet und zur Stabilisierung ein *full upwinding* sowie ein

exponential upwinding Schema eingeführt. Es ergibt sich folgendes lineares Gleichungssystem:

$$\sum_{j\in\Lambda_i}[\left(c_j^{(t+\Delta t)}-c_i^{(t+\Delta t)}\right)\sum_{k\in\beta}|\nu_{ij}^k|\frac{1}{d_{ij}}\left(D_{(F)ij}^{kk}+D_{(S)ij}^{kk}\right)-$$
$$\sum_{k\in\beta}|\nu_{ij}^k|u_{ij}^k\left(p_{ij}c_i^{(t+\Delta t)}+(1-p_{ij})c_j^{(t+\Delta t)}\right)]m_{ij}+m_i\left(-\frac{q_i}{nf_i}-\frac{R_i}{\Delta t}\right)c_i^{(t+\Delta t)}=$$
$$m_i\left(\frac{q_i}{nf_i}c_{(in)i}-\frac{c_i^{(t)}}{\Delta t}R_i-\sigma_i\right)\ . \quad (7.2)$$

Für die Lösung der entstehenden lineare Gleichungssysteme werden geeignete Verfahren wie das Verfahren der *konjugierten Gradienten* (CG) und das Verfahren der *stabilisierten bikonjugierten Gradienten* (BiCGStab) vorgestellt.

Kapitel 4 Das GIS GRASS

Das Open Source Geoinformationsystem GRASS ist ein multifunktionales GIS das Raster-, Vektor- und Volumendaten verarbeiten kann. Es zeichnet sich durch einen enormen Funktionsumfang und eine sehr gute modulare Konzeption aus. Eine weitere Stärke von GRASS ist die Zugänglichkeit des Programmcodes. Diese Offenheit erlaubt die Implementierung des Strömungs- und Transportmodells in GRASS. Zu diesem Zweck wird auf die verschiedene Programmiermethoden der Bibliotheks-, Modul- und Shellprogrammierung eingegangen.

Realisierung, Validierung und Diskussion der Zielsetzung

Kapitel 5 Ergebnisse

Die Realisierung und Validierung des in GRASS implementierten Strömungs- und Transportmodells wird in diesem Kapitel erläutert. Die Basis des numerischen Strömungs- und Transportmodells bildet ein modulares Konzept das auf Bibliotheksfunktionen, Programmmodulen und Shell Skripten aufbaut. Implementiert wird eine Erweiterung der GRASS Bibliothek *gpde* die die Strömungs- und Transportdifferentialgleichung auf Basis der FVM löst. Aufbauend auf dieser Bibliothek werden die Programme *r.gwflow* für die Strömungsberechnung und *r.solute.transport* für die Stofftransportberechnung in zwei Dimensionen implementiert. Gekoppelt werden diese Module durch verschiedene Shell Skripte. Die programmierten Erweiterungen sowie die Module werden anhand von Literaturbeispielen validiert. Das modulare Konzept wird anhand eines zeitlich gekoppelten Strömungs- und Stofftransportmodells demonstriert.

Kapitel 6 Diskussion

In Kapitel sechs werden die Vor- und Nachteile der mathematischen Grundlagen, des GIS GRASS und der gewählten Konzeption diskutiert. Dabei werden folgende Schlussfolgerung getroffen

- Die numerischen Methoden sind ausreichend um reale Fragestellungen der Strömung und des Stofftransportes zu modellieren.
- Das GIS GRASS eignet sich sehr gut für die Bearbeitung hydrogeologischer Fragestellungen .
- Die Offenheit des Quellcodes von GRASS bietet enorme Möglichkeiten für Erweiterungen und eignet sich somit für die Implementierung des hier vorgestellten Konzeptes.
- Die Implementierung folgt modernen Konzepten des Softwaredesignes und integriert sich in das GIS GRASS.
- Die Validierungsergebnisse sind zufriedenstellend und zeigen, dass die mathematischen Modelle und deren Implementierung im Falle der Validierungsszenarien korrekt sind.
- Die implementierten Module und Erweiterungen ermöglichen aufgrund ihrer Modularität die Berechnung komplexer zeitlich gekoppelter Szenarien.

Literaturverzeichnis

[Bey93] Bey, j.: *Finite-Volumen- und Mehrgitter-Verfahren für elliptische Randwertprobleme*. 2. Auflage. B.G.Teubner (Stuttgart, Leibzig), 1993 (Advanced in Numerical Mathematics)

[Geb07] Gebbert, S.: *GRASS GIS Workshop - 3D-Analysen in der Hydrogeologie*. http://www.nature-consult.de/index.php?option=com_content&task=view&id=91&Itemid=91. Version: 2007

[Hac93] Hackbusch, W.: *Iterative Lösung großer schwachbesetzter Gleichungssysteme*. 2. Auflage. B.G.Teubner (Stuttgart, Leibzig), 1993 (Teubner Studienbücher)

[Hin03] Hinkelmann, R.: *Efficient Numerical Methods and Information-Processing Techniques in Environment Water*. Institut Wasserbau, Universität Stuttgart, 2003 (Mitteilungen)

[KA00] Knabner, P. ; Angermann, L.: *Numerik partieller Differentialgleichungen*. Springer (Berlin; Heidelberg; New York; Barcelona; Hongkong; London; Mailand; Paris; Singapur; Tokio), 2000. – ISBN 3540662316

[KR95] Kinzelbach, W. ; Rausch, R.: *Grundwassermodellierung*. Gebrüder Borntraeger (Berlin, Stuttgart), 1995

[LKW96] Lege, T. ; Kolditz, O. ; W., Zielke: *Strömungs und Transportmodellierung*. 2. Auflage. Springer (Berlin; Heidelberg; New York; Barcelona; Hongkong; London; Mailand; Paris; Singapur; Tokio), 1996 (Handbuch zur Erkundung des Untergrundes von Deponien und Altlasten)

[Mei99] Meister, A.: *Numerik linearer Gleichungssysteme, Eine Einführung in moderne Verfahren*. vieweg (Braunschweig/Wiesbaden), 1999

[MU83] Matthess, G. ; Ubell, K.: *Allgemeine Hydrogeologie Grundwasserhaushalt*. Gebrüder Borntraeger (Berlin, Stuttgart), 1983 (Lehrbuch der Hydrogeologie Band 1)

[NM04] Neteler, M. ; Mitasova, H.: *Open Source GIS: A GRASS GIS Approach*. Second. Kluwer Academic Publishers/Springer, Boston, 2004 (SECS 773). – 424 S. – ISBN: 1-4020-8064-6; also published as eBook: ISBN 1-4020-8065-4

[RSW02] Rausch, R. ; Schäfer, W. ; Wagner, Ch.: *Einführung in die Transportmodellierung im Grundwasser*. Gebrüder Borntraeger (Berlin, Stuttgart), 2002

Anhang

A Quelltext der gpde Erweiterung

A.1 Strömungsmodell

A.1.1 Header Datei

```
1
2  /****************************************************************************
3  *
4  * MODULE:       Grass PDE Numerical Library
5  * AUTHOR(S):    Soeren Gebbert, Berlin (GER) Dec 2006
6  *     soerengebbert <at> gmx <dot> de
7  *
8  * PURPOSE:      groundwater flow in porous media
9  *     part of the gpde library
10 *
11 * COPYRIGHT:    (C) 2000 by the GRASS Development Team
12 *
13 *               This program is free software under the GNU General Public
14 *               License (>=v2). Read the file COPYING that comes with GRASS
15 *               for details.
16 *
17 *****************************************************************************/
18
19 #ifndef _N_GWFLOW_H_
20 #define _N_GWFLOW_H_
21 #include "N_pde.h"
22
23 #define N_GW_CONFINED 0   /*confined groundwater */
24 #define N_GW_UNCONFINED 1 /*unconfined groundwater */
25
26 /*!
27  * \brief This data structure contains all data needed to compute the
28  * groundwater mass balance in two dimension
29  * */
30 typedef struct
31 {
32   N_array_2d *phead;    /*!piezometric head */
33   N_array_2d *phead_start;  /*!start conditions */
34   N_array_2d *hc_x;   /*!x part of the hydraulic conductivity tensor */
35   N_array_2d *hc_y;   /*!y part of the hydraulic conductivity tensor */
36   N_array_2d *q;    /*!sources and sinks */
37   N_array_2d *r;    /*!reacharge at the top of the gw leayer */
38   N_array_2d *s;    /*!specific yield */
39   N_array_2d *nf;   /*!effective porosity */
40
41   N_array_2d *top;    /*!top surface of the quifer */
42   N_array_2d *bottom;   /*!bottom of the aquifer */
43
44   N_array_2d *status;   /*!active/inactive/dirichlet cell status */
45
46   double dt;      /*!calculation time */
47   int gwtype;     /*!Which type of groundwater, N_GW_CONFINED or N_GW_UNCONFIED */
48
49 } N_gwflow_data2d;
50
51 extern N_data_star *N_callback_gwflow_2d (void *gwdata, N_geom_data * geom, int col, int row);
52 extern N_gwflow_data2d *N_alloc_gwflow_data2d (int cols, int rows);
53 extern void N_free_gwflow_data2d (N_gwflow_data2d * data);
54 #endif
```

A.1.2 Hauptdatei

```
1
2  /****************************************************************************
```

```
3   *
4   * MODULE:       Grass PDE Numerical Library
5   * AUTHOR(S):    Soeren Gebbert, Berlin (GER) Dec 2006
6   *     soerengebbert <at> gmx <dot> de
7   *
8   * PURPOSE:      groundwater flow in porous media
9   *     part of the gpde library
10  *
11  * COPYRIGHT:    (C) 2000 by the GRASS Development Team
12  *
13  *               This program is free software under the GNU General Public
14  *               License (>=v2). Read the file COPYING that comes with GRASS
15  *               for details.
16  *
17  *****************************************************************************/
18
19  #include "grass/N_gwflow.h"
20
21  /* ************************************************************* */
22  /* ******************** N_alloc_gwflow_data2d ******************** */
23  /* ************************************************************* */
24  /*!
25   * \brief Alllocate memory for the groundwater calculation data structure in 2 dimensions
26   *
27   * The groundwater calculation data structure will be allocated including
28   * all appendant 2d arrays. The offset for the 2d arrays is one
29   * to establish homogeneous Neumann boundary conditions at the calculation area border.
30   * This data structure is used to create a linear equation system based on the computation of
31   * groundwater flow in porous media with the finite volume method.
32   *
33   * \param cols int
34   * \param rows int
35   * \return N_gwflow_data2d *
36   * */
37  N_gwflow_data2d *N_alloc_gwflow_data2d(int cols, int rows)
38  {
39      N_gwflow_data2d *data;
40
41      data = (N_gwflow_data2d *) G_calloc(1, sizeof(N_gwflow_data2d));
42
43      data->phead = N_alloc_array_2d(cols, rows, 1, DCELL_TYPE);
44      data->phead_start = N_alloc_array_2d(cols, rows, 1, DCELL_TYPE);
45      data->status = N_alloc_array_2d(cols, rows, 1, DCELL_TYPE);
46      data->hc_x = N_alloc_array_2d(cols, rows, 1, DCELL_TYPE);
47      data->hc_y = N_alloc_array_2d(cols, rows, 1, DCELL_TYPE);
48      data->q = N_alloc_array_2d(cols, rows, 1, DCELL_TYPE);
49      data->s = N_alloc_array_2d(cols, rows, 1, DCELL_TYPE);
50      data->nf = N_alloc_array_2d(cols, rows, 1, DCELL_TYPE);
51      data->r = N_alloc_array_2d(cols, rows, 1, DCELL_TYPE);
52      data->top = N_alloc_array_2d(cols, rows, 1, DCELL_TYPE);
53      data->bottom = N_alloc_array_2d(cols, rows, 1, DCELL_TYPE);
54
55      return data;
56  }
57
58  /* ************************************************************* */
59  /* ***************** N_free_gwflow_data2d ********************** */
60  /* ************************************************************* */
61  /*!
62   * \brief Release the memory of the groundwater flow data structure in two dimensions
63   *
64   * \param data N_gwflow_data2d *
65   * \return void
66   * */
67  void N_free_gwflow_data2d(N_gwflow_data2d * data)
68  {
69      N_free_array_2d(data->phead);
70      N_free_array_2d(data->phead_start);
71      N_free_array_2d(data->status);
72      N_free_array_2d(data->hc_x);
73      N_free_array_2d(data->hc_y);
74      N_free_array_2d(data->q);
75      N_free_array_2d(data->s);
76      N_free_array_2d(data->nf);
77      N_free_array_2d(data->r);
78      N_free_array_2d(data->top);
79      N_free_array_2d(data->bottom);
80
81      G_free(data);
82
83      data = NULL;;
84
85      return;
86  }
87
88  /* ************************************************************* */
89  /* ***************** N_callback_gwflow_2d ********************** */
90  /* ************************************************************* */
91  /*!
92   * \brief This callback function creates the mass balance of a 5 point star
```

```
 *
 * The mass balance is based on the common groundwater flow equation:
 *
 * \f[Ss \frac{\partial h}{\partial t} = \nabla {\bf K} \nabla h + q \f]
 *
 * This equation is discretizised with the finite volume method in two dimensions.
 *
 * \param gwdata N_gwflow_data2d *
 * \param geom N_geom_data *
 * \param col int
 * \param row int
 * \return N_data_star *
 *
 * */
N_data_star *N_callback_gwflow_2d(void *gwdata, N_geom_data * geom, int col,
          int row)
{
    double T_e = 0, T_w = 0, T_n = 0, T_s = 0;
    double z_e = 0, z_w = 0, z_n = 0, z_s = 0;
    double dx, dy, Az;
    double hc_x, hc_y;
    double z, top;
    double hc_xw, hc_yn;
    double z_xw, z_yn;
    double hc_xe, hc_ys;
    double z_xe, z_ys;
    double hc, hc_start;
    double Ss, r, nf, q;
    double C, W, E, N, S, V;
    N_gwflow_data2d *data;
    N_data_star *mat_pos;

    /*cast the void pointer to the right data structure */
    data = (N_gwflow_data2d *) gwdata;

    dx = geom->dx;
    dy = geom->dy;
    Az = N_get_geom_data_area_of_cell(geom, row);

    /*read the data from the arrays */
    hc_start = N_get_array_2d_d_value(data->phead_start, col, row);
    hc = N_get_array_2d_d_value(data->phead, col, row);
    top = N_get_array_2d_d_value(data->top, col, row);


    if (hc > top) {   /*If the aquifer is confined */
  z = N_get_array_2d_d_value(data->top, col,
          row) -
      N_get_array_2d_d_value(data->bottom, col, row);
  z_xw =
      N_get_array_2d_d_value(data->top, col - 1,
          row) -
      N_get_array_2d_d_value(data->bottom, col - 1, row);
  z_xe =
      N_get_array_2d_d_value(data->top, col + 1,
          row) -
      N_get_array_2d_d_value(data->bottom, col + 1, row);
  z_yn =
      N_get_array_2d_d_value(data->top, col,
          row - 1) -
      N_get_array_2d_d_value(data->bottom, col, row - 1);
  z_ys =
      N_get_array_2d_d_value(data->top, col,
          row + 1) -
      N_get_array_2d_d_value(data->bottom, col, row + 1);
       }
    else {      /* the aquifer is unconfined */

  /* If the aquifer is unconfied use an explicite scheme to solve
   * the nonlinear equation. We use the phead from the first iteration */
  z = N_get_array_2d_d_value(data->phead, col, row) -
      N_get_array_2d_d_value(data->bottom, col, row);
  z_xw = N_get_array_2d_d_value(data->phead, col - 1, row) -
      N_get_array_2d_d_value(data->bottom, col - 1, row);
  z_xe = N_get_array_2d_d_value(data->phead, col + 1, row) -
      N_get_array_2d_d_value(data->bottom, col + 1, row);
  z_yn = N_get_array_2d_d_value(data->phead, col, row - 1) -
      N_get_array_2d_d_value(data->bottom, col, row - 1);
  z_ys = N_get_array_2d_d_value(data->phead, col, row + 1) -
      N_get_array_2d_d_value(data->bottom, col, row + 1);
    }

    /*geometrical mean of cell height*/
    z_w = N_calc_geom_mean(z_xw, z);
    z_e = N_calc_geom_mean(z_xe, z);
    z_n = N_calc_geom_mean(z_yn, z);
    z_s = N_calc_geom_mean(z_ys, z);

    /* Inner sources */
    q = N_get_array_2d_d_value(data->q, col, row);
```

```
183     nf = N_get_array_2d_d_value(data->nf, col, row);
184
185     /* specific yield */
186     Ss = N_get_array_2d_d_value(data->s, col, row) * Az;
187     /* reacharge */
188     r = N_get_array_2d_d_value(data->r, col, row);
189
190     /*get the surrounding permeabilities */
191     hc_x = N_get_array_2d_d_value(data->hc_x, col, row);
192     hc_y = N_get_array_2d_d_value(data->hc_y, col, row);
193     hc_xw = N_get_array_2d_d_value(data->hc_x, col - 1, row);
194     hc_xe = N_get_array_2d_d_value(data->hc_x, col + 1, row);
195     hc_yn = N_get_array_2d_d_value(data->hc_y, col, row - 1);
196     hc_ys = N_get_array_2d_d_value(data->hc_y, col, row + 1);
197
198     /* calculate the transmissivities */
199     T_w = N_calc_harmonic_mean(hc_xw, hc_x) * z_w;
200     T_e = N_calc_harmonic_mean(hc_xe, hc_x) * z_e;
201     T_n = N_calc_harmonic_mean(hc_yn, hc_y) * z_n;
202     T_s = N_calc_harmonic_mean(hc_ys, hc_y) * z_s;
203
204     /*mass balance center cell to western cell */
205     W = -1 * T_w * dy / dx;
206     /*mass balance center cell to eastern cell */
207     E = -1 * T_e * dy / dx;
208     /*mass balance center cell to northern cell */
209     N = -1 * T_n * dx / dy;
210     /*mass balance center cell to southern cell */
211     S = -1 * T_s * dx / dy;
212
213     /*the diagonal entry of the matrix */
214     C = -1 * (W + E + N + S - Ss / data->dt);
215
216     /*the entry in the right side b of Ax = b */
217     V = (q + hc_start * Ss / data->dt) + r * Az;
218
219     G_debug(5, "N_callback_gwflow_2d: called [%i][%i]", row, col);
220
221     /*create the 5 point star entries */
222     mat_pos = N_create_5star(C, W, E, N, S, V);
223
224     return mat_pos;
225 }
```

A.2 Transportmodell

A.2.1 Header Datei

```
1
2  /****************************************************************************
3  *
4  * MODULE:       Grass PDE Numerical Library
5  * AUTHOR(S):    Soeren Gebbert, Berlin (GER) Dec 2006
6  *     soerengebbert <at> gmx <dot> de
7  *
8  * PURPOSE:      solute transport in porous media
9  *     part of the gpde library
10 *
11 * COPYRIGHT:    (C) 2000 by the GRASS Development Team
12 *
13 *               This program is free software under the GNU General Public
14 *               License (>=v2). Read the file COPYING that comes with GRASS
15 *               for details.
16 *
17 *****************************************************************************/
18
19 #include "N_pde.h"
20
21 #ifndef _N_SOLUTE_TRANSPORT_H_
22 #define _N_SOLUTE_TRANSPORT_H_
23
24
25 typedef struct
26 {
27   N_array_2d *c;    /*concentration */
28   N_array_2d *c_start;    /*concentration at start*/
29   N_array_2d *diff_x;   /*x part of the diffusion tensor */
30   N_array_2d *diff_y;   /*y part of the diffusion tensor */
31   N_array_2d *nf;   /*effective porosity*/
32   N_array_2d *cs;   /*concentration sources and sinks */
33   N_array_2d *q;    /*well sources and sinks */
34   N_array_2d *R;    /*retardation */
```

```
35    N_array_2d *cin;    /*concentration  */
36
37    N_gradient_field_2d *grad;  /*velocity field*/
38
39    N_array_2d *status;   /*active/inactive/dirichlet cell status */
40    N_array_2d *top;    /* top surface of the aquifer */
41    N_array_2d *bottom;   /* bottom surface of the aquifer */
42
43    N_array_2d *disp_xx;    /*x part of the dispersivity tensor*/
44    N_array_2d *disp_yy;    /*x part of the dispersivity tensor*/
45    N_array_2d *disp_xy;    /*xy part of the dispersivity tensor*/
46
47    double dt;      /*calculation time */
48    double al, at;    /*dispersivity length longditudinal and transversal*/
49
50  } N_solute_transport_data2d;
51
52
53  extern N_data_star *N_callback_solute_transport_2d (void *solutedata, N_geom_data * geom, int col, int row);
54  extern N_solute_transport_data2d *N_alloc_solute_transport_data2d (int cols, int rows);
55  extern void N_free_solute_transport_data2d (N_solute_transport_data2d * data);
56  extern void N_calc_solute_transport_disptensor_2d(N_solute_transport_data2d * data);
57  extern void N_calc_solute_transport_transmission_2d(N_solute_transport_data2d * data);
58  #endif
```

A.2.2 Hauptdatei

```
1
2   /****************************************************************************
3   *
4   * MODULE:       Grass PDE Numerical Library
5   * AUTHOR(S):    Soeren Gebbert, Berlin (GER) Dec 2006
6   *     soerengebbert <at> gmx <dot> de
7   *
8   * PURPOSE:      solute transport in porous media
9   *     part of the gpde library
10  *
11  * COPYRIGHT:    (C) 2007 by the GRASS Development Team
12  *
13  *               This program is free software under the GNU General Public
14  *               License (>=v2). Read the file COPYING that comes with GRASS
15  *               for details.
16  *
17  *****************************************************************************/
18
19  #include "grass/N_solute_transport.h"
20  #include <math.h>
21
22  /* ******************************************************************** *
23   * ******************************************************************** *
24   * ******************************************************************** */
25  /*!
26   * \brief This callback function creates the mass balance of a 5 point star
27   *
28   * The mass balance is based on the common solute transport equation:
29   *
30   * \f[\frac{\partial c_g}{\partial t} R = \nabla \cdot ({\bf D} \nabla c_g - {\bf u} c_g) + \sigma + \frac{q}{n_f}(c_g - c_in) \f]
31   *
32   * This equation is discretizised with the finite volume method in two dimensions.
33   *
34   *
35   * \param solutedata  * N_solute_transport_data2d - a void pointer to the data structure
36   * \param geom N_geom_data *
37   * \param col   int
38   * \param row   int
39   * \return N_data_star * - a five point data star
40   *
41   * */
42  N_data_star *N_callback_solute_transport_2d(void *solutedata,
43                N_geom_data * geom, int col,
44                int row)
45  {
46      double Df_e = 0, Df_w = 0, Df_n = 0, Df_s = 0;
47      double z_e = 0, z_w = 0, z_n = 0, z_s = 0;
48      double dx, dy, Az;
49      double diff_x, diff_y;
50      double disp_x, disp_y;
51      double z;
52      double diff_xw, diff_yn;
53      double disp_xw, disp_yn;
54      double z_xw, z_yn;
55      double diff_xe, diff_ys;
56      double disp_xe, disp_ys;
57      double z_xe, z_ys;
58      double cin = 0, cg, cg_start;
59      double R, nf, cs, q;
```

```
60      double C, W, E, N, S, V;
61      double vw = 0, ve = 0, vn = 0, vs = 0;
62      double Ds_w = 0, Ds_e = 0, Ds_n = 0, Ds_s = 0;
63      double Dw = 0, De = 0, Dn = 0, Ds = 0;
64      double rw = 0.5, re = 0.5, rn = 0.5, rs = 0.5;
65
66      N_solute_transport_data2d *data = NULL;
67      N_data_star *mat_pos;
68      N_gradient_2d grad;
69
70      /*cast the void pointer to the right data structure */
71      data = (N_solute_transport_data2d *) solutedata;
72
73      N_get_gradient_2d(data->grad, &grad, col, row);
74
75      dx = geom->dx;
76      dy = geom->dy;
77      Az = N_get_geom_data_area_of_cell(geom, row);
78
79      /*read the data from the arrays */
80      cg_start = N_get_array_2d_d_value(data->c_start, col, row);
81      cg = N_get_array_2d_d_value(data->c, col, row);
82
83      /* calculate the cell height */
84      z = N_get_array_2d_d_value(data->top, col,
85                row) -
86    N_get_array_2d_d_value(data->bottom, col, row);
87      z_xw =
88    N_get_array_2d_d_value(data->top, col - 1,
89                row) -
90    N_get_array_2d_d_value(data->bottom, col - 1, row);
91      z_xe =
92    N_get_array_2d_d_value(data->top, col + 1,
93                row) -
94    N_get_array_2d_d_value(data->bottom, col + 1, row);
95      z_yn =
96    N_get_array_2d_d_value(data->top, col,
97                row - 1) -
98    N_get_array_2d_d_value(data->bottom, col, row - 1);
99      z_ys =
100   N_get_array_2d_d_value(data->top, col,
101               row + 1) -
102   N_get_array_2d_d_value(data->bottom, col, row + 1);
103
104     /*arithmetical mean of cell height */
105     z_w = N_calc_arith_mean(z_xw, z);
106     z_e = N_calc_arith_mean(z_xe, z);
107     z_n = N_calc_arith_mean(z_yn, z);
108     z_s = N_calc_arith_mean(z_ys, z);
109
110     /*get the surrounding diffusion tensor entries */
111     diff_x = N_get_array_2d_d_value(data->diff_x, col, row);
112     diff_y = N_get_array_2d_d_value(data->diff_y, col, row);
113     diff_xw = N_get_array_2d_d_value(data->diff_x, col - 1, row);
114     diff_xe = N_get_array_2d_d_value(data->diff_x, col + 1, row);
115     diff_yn = N_get_array_2d_d_value(data->diff_y, col, row - 1);
116     diff_ys = N_get_array_2d_d_value(data->diff_y, col, row + 1);
117
118     /* calculate the diffusion at the cell borders using the harmonical mean */
119     Df_w = N_calc_harmonic_mean(diff_xw, diff_x);
120     Df_e = N_calc_harmonic_mean(diff_xe, diff_x);
121     Df_n = N_calc_harmonic_mean(diff_yn, diff_y);
122     Df_s = N_calc_harmonic_mean(diff_ys, diff_y);
123
124     /* calculate the dispersion */
125     /*get the surrounding dispersion tensor entries */
126     disp_x = N_get_array_2d_d_value(data->disp_xx, col, row);
127     disp_y = N_get_array_2d_d_value(data->disp_yy, col, row);
128     if(N_get_array_2d_d_value(data->status, col - 1, row) == N_CELL_TRANSMISSION) {
129   disp_xw = disp_x;
130     }else{
131       disp_xw = N_get_array_2d_d_value(data->disp_xx, col - 1, row);
132     }if(N_get_array_2d_d_value(data->status, col + 1, row) == N_CELL_TRANSMISSION){
133   disp_xe = disp_x;
134     }else{
135       disp_xe = N_get_array_2d_d_value(data->disp_xx, col + 1, row);
136     }if(N_get_array_2d_d_value(data->status, col, row - 1) == N_CELL_TRANSMISSION){
137   disp_yn = disp_y;
138     }else{
139       disp_yn = N_get_array_2d_d_value(data->disp_yy, col, row - 1);
140     }if(N_get_array_2d_d_value(data->status, col, row + 1) == N_CELL_TRANSMISSION){
141   disp_ys = disp_y;
142     }else{
143       disp_ys = N_get_array_2d_d_value(data->disp_yy, col, row + 1);
144     }
145
146     /* calculate the dispersion at the cell borders using the harmonical mean */
147     Ds_w = N_calc_harmonic_mean(disp_xw, disp_x);
148     Ds_e = N_calc_harmonic_mean(disp_xe, disp_x);
149     Ds_n = N_calc_harmonic_mean(disp_yn, disp_y);
```

```
150       Ds_s = N_calc_harmonic_mean(disp_ys, disp_y);
151
152       /* put the diffusion and dispersion together */
153       Dw = ((Df_w + Ds_w)) / dx;
154       De = ((Df_e + Ds_e)) / dx;
155       Ds = ((Df_s + Ds_s)) / dy;
156       Dn = ((Df_n + Ds_n)) / dy;
157
158       vw = grad.WC;
159       rw = N_exp_upwinding(-1 * vw, dx, Dw);
160       ve = grad.EC;
161       re = N_exp_upwinding(ve, dx, De);
162       vs = grad.SC;
163       rs = N_exp_upwinding(-1 * vs, dy, Ds);
164       vn = grad.NC;
165       rn = N_exp_upwinding(vn, dy, Dn);
166
167       /*mass balance center cell to western cell */
168       W = -1 * (Dw) * dy * z_w - vw * (1 - rw) * dy * z_w;
169       /*mass balance center cell to eastern cell */
170       E = -1 * (De) * dy * z_e + ve * (1 - re) * dy * z_e;
171       /*mass balance center cell to southern cell */
172       S = -1 * (Ds) * dx * z_s - vs * (1 - rs) * dx * z_s;
173       /*mass balance center cell to northern cell */
174       N = -1 * (Dn) * dx * z_n + vn * (1 - rn) * dx * z_n;
175
176       /* Retardation */
177       R = N_get_array_2d_d_value(data->R, col, row);
178       /* Inner sources */
179       cs = N_get_array_2d_d_value(data->cs, col, row);
180       /* effective porosity */
181       nf = N_get_array_2d_d_value(data->nf, col, row);
182       /* groundwater sources and sinks */
183       q = N_get_array_2d_d_value(data->q, col, row);
184       /* concentration of influent water */
185       cin = N_get_array_2d_d_value(data->cin, col, row);
186
187       /*the diagonal entry of the matrix */
188       C = (Dw - vw * rw) * dy * z_w +
189     (De + ve * re) * dy * z_e +
190     (Ds - vs * rs) * dx * z_s +
191     (Dn + vn * rn) * dx * z_n + Az * z * R / data->dt - q / nf;
192
193       /*the entry in the right side b of Ax = b */
194       V = (cs + cg_start * Az * z * R / data->dt + q / nf * cin);
195
196       /*
197         fprintf(stderr, "nf %g\n", nf);
198         fprintf(stderr, "q %g\n", q);
199         fprintf(stderr, "cs %g\n", cs);
200         fprintf(stderr, "cin %g\n", cin);
201         fprintf(stderr, "cg %g\n", cg);
202         fprintf(stderr, "cg_start %g\n", cg_start);
203         fprintf(stderr, "Az %g\n", Az);
204         fprintf(stderr, "z %g\n", z);
205         fprintf(stderr, "R %g\n", R);
206         fprintf(stderr, "dt %g\n", data->dt);
207       */
208
209       G_debug(6, "N_callback_solute_transport_2d: called [%i][%i]", row, col);
210
211       /*create the 5 point star entries */
212       mat_pos = N_create_5star(C, W, E, N, S, V);
213
214       return mat_pos;
215   }
216
217   /* ************************************************************************* *
218    * ************************************************************************* *
219    * ************************************************************************* */
220   /*!
221    * \brief Alllocate memory for the solute transport data structure in two dimensions
222    *
223    * The solute transport data structure will be allocated including
224    * all appendant 2d arrays. The offset for the 2d arrays is one
225    * to establish homogeneous Neumann boundary conditions at the calculation area border.
226    * This data structure is used to create a linear equation system based on the computation of
227    * solute transport in porous media with the finite volume method.
228    *
229    * \param cols   int
230    * \param rows   int
231    * \return N_solute_transport_data2d *
232    * */
233
234
235   N_solute_transport_data2d *N_alloc_solute_transport_data2d(int cols, int rows)
236   {
237       N_solute_transport_data2d *data = NULL;
238
239       data =
```

```
240     (N_solute_transport_data2d *) G_calloc(1,
241                   sizeof
242                   (N_solute_transport_data2d));
243
244       data->c = N_alloc_array_2d(cols, rows, 1, DCELL_TYPE);
245       data->c_start = N_alloc_array_2d(cols, rows, 1, DCELL_TYPE);
246       data->status = N_alloc_array_2d(cols, rows, 1, DCELL_TYPE);
247       data->diff_x = N_alloc_array_2d(cols, rows, 1, DCELL_TYPE);
248       data->diff_y = N_alloc_array_2d(cols, rows, 1, DCELL_TYPE);
249       data->q = N_alloc_array_2d(cols, rows, 1, DCELL_TYPE);
250       data->cs = N_alloc_array_2d(cols, rows, 1, DCELL_TYPE);
251       data->R = N_alloc_array_2d(cols, rows, 1, DCELL_TYPE);
252       data->nf = N_alloc_array_2d(cols, rows, 1, DCELL_TYPE);
253       data->cin = N_alloc_array_2d(cols, rows, 1, DCELL_TYPE);
254       data->top = N_alloc_array_2d(cols, rows, 1, DCELL_TYPE);
255       data->bottom = N_alloc_array_2d(cols, rows, 1, DCELL_TYPE);
256
257       /*Allocate the dispersivity tensor */
258       data->disp_xx = N_alloc_array_2d(cols, rows, 1, DCELL_TYPE);
259       data->disp_yy = N_alloc_array_2d(cols, rows, 1, DCELL_TYPE);
260       data->disp_xy = N_alloc_array_2d(cols, rows, 1, DCELL_TYPE);
261
262       data->grad = N_alloc_gradient_field_2d(cols, rows);
263
264       return data;
265   }
266
267
268   /* ************************************************************************* *
269    * ************************************************************************* *
270    * ************************************************************************* */
271   /*!
272    * \brief Release the memory of the solute transport data structure in two dimensions
273    *
274    * \param data N_solute_transport_data2d *
275    * \return void *
276    * */
277   void N_free_solute_transport_data2d(N_solute_transport_data2d * data)
278   {
279       N_free_array_2d(data->c);
280       N_free_array_2d(data->c_start);
281       N_free_array_2d(data->status);
282       N_free_array_2d(data->diff_x);
283       N_free_array_2d(data->diff_y);
284       N_free_array_2d(data->q);
285       N_free_array_2d(data->cs);
286       N_free_array_2d(data->R);
287       N_free_array_2d(data->nf);
288       N_free_array_2d(data->cin);
289       N_free_array_2d(data->top);
290       N_free_array_2d(data->bottom);
291
292       N_free_array_2d(data->disp_xx);
293       N_free_array_2d(data->disp_yy);
294       N_free_array_2d(data->disp_xy);
295
296       G_free(data);
297
298       data = NULL;
299
300       return;
301   }
302   /*!
303    * \brief Compute the transmission boundary condition in 2d
304    *
305    * This function calculates the transmission boundary condition
306    * for each cell with status N_CELL_TRANSMISSION. The surrounding
307    * gradient field is used to verfiy the flow direction. If a flow
308    * goes into a cell, the concentration (data->c) from the neighbour cell is
309    * added to the transmission cell. If the flow from several neighbour
310    * cells goes into the cell, the concentration mean is calculated.
311    *
312    * The new concentrations are written into the data->c_start array,
313    * so they can be handled by the matrix assembling function.
314    *
315    * \param data N_solute_transport_data2d *
316    * \return void *
317    * */
318   void N_calc_solute_transport_transmission_2d(N_solute_transport_data2d * data)
319   {
320       int i, j, count = 1;
321       int cols, rows;
322       double c;
323       N_gradient_2d grad;
324
325       cols = data->grad->cols;
326       rows = data->grad->rows;
327
328       G_debug(2,
329         "N_calc_solute_transport_transmission_2d: calculating transmission boundary");
```

```
330
331       for (j = 0; j < rows; j++) {
332     for (i = 0; i < cols; i++) {
333         if(N_get_array_2d_d_value(data->status, i, j) == N_CELL_TRANSMISSION) {
334           count = 0;
335           /*get the gradient neighbours */
336           N_get_gradient_2d(data->grad, &grad, i, j);
337           c = 0;
338           /*
339           c = N_get_array_2d_d_value(data->c_start, i, j);
340           if(c > 0)
341       count++;
342            */
343
344           if(grad.WC > 0 && !N_is_array_2d_value_null(data->c, i - 1, j)) {
345             c += N_get_array_2d_d_value(data->c, i - 1, j);
346       count++;
347           }
348           if(grad.EC < 0 && !N_is_array_2d_value_null(data->c, i + 1, j)) {
349             c += N_get_array_2d_d_value(data->c, i + 1, j);
350       count++;
351           }
352           if(grad.NC < 0 && !N_is_array_2d_value_null(data->c, i, j - 1)) {
353             c += N_get_array_2d_d_value(data->c, i, j - 1);
354       count++;
355           }
356           if(grad.SC > 0 && !N_is_array_2d_value_null(data->c, i, j + 1)) {
357             c += N_get_array_2d_d_value(data->c, i, j + 1);
358       count++;
359           }
360           if(count != 0)
361             c = c/(double)count;
362           /*make sure it is not NAN*/
363           if(c > 0 || c == 0 || c < 0)
364                   N_put_array_2d_d_value(data->c_start, i, j, c);
365         }
366     }
367       }
368
369       return;
370   }
371
372   /*!
373    * \brief Compute the dispersivity tensor based on the solute transport data in 2d
374    *
375    * The dispersivity tensor is stored in the data structure.
376    * To compute the dispersivity tensor, the dispersivity lentghs and the gradient field
377    * must be present.
378    *
379    * This is just a simple tensor computation which should be extended.
380    *
381    * \todo Change the tensor calculation to a mor realistic algorithm
382    *
383    * \param data N_solute_transport_data2d *
384    * \return void *
385    * */
386   void N_calc_solute_transport_disptensor_2d(N_solute_transport_data2d * data)
387   {
388       int i, j;
389       int cols, rows;
390       double vx, vy, vv;
391       double disp_xx, disp_yy, disp_xy;
392       N_gradient_2d grad;
393
394       cols = data->grad->cols;
395       rows = data->grad->rows;
396
397       G_debug(2,
398         "N_calc_solute_transport_disptensor_2d: calculating the dispersivity tensor");
399
400       for (j = 0; j < rows; j++) {
401     for (i = 0; i < cols; i++) {
402
403         disp_xx = 0;
404         disp_yy = 0;
405         disp_xy = 0;
406
407         /*get the gradient neighbours */
408         N_get_gradient_2d(data->grad, &grad, i, j);
409         vx = (grad.WC + grad.EC) / 2;
410         vy = (grad.NC + grad.SC) / 2;
411         vv = sqrt(vx * vx + vy * vy);
412
413         if (vv != 0) {
414       disp_xx = data->al * vx * vx / vv + data->at * vy * vy / vv;
415       disp_yy = data->at * vx * vx / vv + data->al * vy * vy / vv;
416       disp_xy = (data->al - data->at) * vx * vy / vv;
417         }
418
419         G_debug(5,
```

```
420           "N_calc_solute_transport_disptensor_2d: [%i][%i] disp_xx %g disp_yy %g disp_xy %g",
421           i, j, disp_xx, disp_yy, disp_xy);
422         N_put_array_2d_d_value(data->disp_xx, i, j, disp_xx);
423         N_put_array_2d_d_value(data->disp_yy, i, j, disp_yy);
424         N_put_array_2d_d_value(data->disp_xy, i, j, disp_xy);
425     }
426       }
427
428       return;
429   }
```

B Quelltext r.gwflow

```
1
2  /****************************************************************************
3  *
4  * MODULE:       r.gwflow
5  *
6  * AUTHOR(S):    Original author
7  *               Soeren Gebbert soerengebbert <at> gmx <dot> de
8  *     27 11 2006 Berlin
9  * PURPOSE:      Calculates confiend and unconfined transient two dimensional groundwater flow
10 *
11 * COPYRIGHT:    (C) 2006 by the GRASS Development Team
12 *
13 *               This program is free software under the GNU General Public
14 *           License (>=v2). Read the file COPYING that comes with GRASS
15 *           for details.
16 *
17 *****************************************************************************/
18 #include <stdio.h>
19 #include <stdlib.h>
20 #include <string.h>
21 #include <math.h>
22 #include <grass/gis.h>
23 #include <grass/glocale.h>
24 #include <grass/N_pde.h>
25 #include <grass/N_gwflow.h>
26
27
28 /*- Parameters and global variables ---------------------------------------*/
29 typedef struct
30 {
31     struct Option *output, *phead, *status, *hc_x, *hc_y, *q, *s, *r, *top,
32   *bottom, *vector, *type, *dt, *maxit, *error, *solver, *sor;
33     struct Flag *sparse;
34 } paramType;
35
36 paramType param;    /*Parameters */
37
38 /*- prototypes ------------------------------------------------------------*/
39 void set_params();    /*Fill the paramType structure */
40 void copy_result(N_array_2d * status, N_array_2d * phead_start, double *result,
41      struct Cell_head *region, N_array_2d * target);
42 N_les *create_solve_les(N_geom_data * geom, N_gwflow_data2d * data,
43       N_les_callback_2d * call, const char * solver, int maxit,
44       double error, double sor);
45
46 /* ******************************************************************** */
47 /* Set up the arguments we are expecting ******************************** */
48 /* ******************************************************************** */
49 void set_params()
50 {
51     param.phead = G_define_option();
52     param.phead->key = "phead";
53     param.phead->type = TYPE_STRING;
54     param.phead->required = YES;
55     param.phead->gisprompt = "old,raster,raster";
56     param.phead->description = _("The initial piezometric head in [m]");
57
58     param.status = G_define_option();
59     param.status->key = "status";
60     param.status->type = TYPE_STRING;
61     param.status->required = YES;
62     param.status->gisprompt = "old,raster,raster";
63     param.status->description = _("boundary condition status, 0-inactive, 1-active, 2-dirichlet");
64
65     param.hc_x = G_define_option();
66     param.hc_x->key = "hc_x";
67     param.hc_x->type = TYPE_STRING;
68     param.hc_x->required = YES;
69     param.hc_x->gisprompt = "old,raster,raster";
70     param.hc_x->description =
71   _("X-part of the hydraulic conductivity tensor in [m/s]");
72
73     param.hc_y = G_define_option();
74     param.hc_y->key = "hc_y";
```

```
75      param.hc_y->type = TYPE_STRING;
76      param.hc_y->required = YES;
77      param.hc_y->gisprompt = "old,raster,raster";
78      param.hc_y->description =
79    _("Y-part of the hydraulic conductivity tensor in [m/s]");
80
81      param.q = G_define_option();
82      param.q->key = "q";
83      param.q->type = TYPE_STRING;
84      param.q->required = NO;
85      param.q->gisprompt = "old,raster,raster";
86      param.q->description = _("Water sources and sinks in [m^3/s]");
87
88      param.s = G_define_option();
89      param.s->key = "s";
90      param.s->type = TYPE_STRING;
91      param.s->required = YES;
92      param.s->gisprompt = "old,raster,raster";
93      param.s->description = _("Specific yield in [1/m]");
94
95      param.r = G_define_option();
96      param.r->key = "r";
97      param.r->type = TYPE_STRING;
98      param.r->required = NO;
99      param.r->gisprompt = "old,raster,raster";
100     param.r->description = _("Reacharge map e.g: 6*10^-9 per cell in [m^3/s*m^2]");
101
102     param.top = G_define_option();
103     param.top->key = "top";
104     param.top->type = TYPE_STRING;
105     param.top->required = YES;
106     param.top->gisprompt = "old,raster,raster";
107     param.top->description = _("Top surface of the aquifer in [m]");
108
109     param.bottom = G_define_option();
110     param.bottom->key = "bottom";
111     param.bottom->type = TYPE_STRING;
112     param.bottom->required = YES;
113     param.bottom->gisprompt = "old,raster,raster";
114     param.bottom->description = _("Bottom surface of the aquifer in [m]");
115
116     param.output = G_define_option();
117     param.output->key = "output";
118     param.output->type = TYPE_STRING;
119     param.output->required = YES;
120     param.output->gisprompt = "new,raster,raster";
121     param.output->description =
122   _
123   ("The map storing the numerical result [m].");
124
125     param.vector = G_define_option();
126     param.vector->key = "velocity";
127     param.vector->type = TYPE_STRING;
128     param.vector->required = NO;
129     param.vector->gisprompt = "new,raster,raster";
130     param.vector->description =
131   _
132   ("Calculate the groundwater filter velocity vector field [m/s]\n"
133          "and write the x, and y components to maps named name_[xy]\n");
134
135     param.type = G_define_option();
136     param.type->key = "type";
137     param.type->type = TYPE_STRING;
138     param.type->required = NO;
139     param.type->answer = "confined";
140     param.type->options = "confined,unconfined";
141     param.type->description =
142   _("The type of groundwater flow.");
143
144     param.dt = N_define_standard_option(N_OPT_CALC_TIME);
145     param.maxit = N_define_standard_option(N_OPT_MAX_ITERATIONS);
146     param.error = N_define_standard_option(N_OPT_ITERATION_ERROR);
147     param.solver = N_define_standard_option(N_OPT_SOLVER_SYMM);
148     param.sor = N_define_standard_option(N_OPT_SOR_VALUE);
149
150     param.sparse = G_define_flag();
151     param.sparse->key = 's';
152     param.sparse->description =
153   _
154   ("Use a sparse matrix, only available with iterative solvers");
155
156 }
157
158 /* ************************************************************************* */
159 /* Main function ********************************************************** */
160 /* ************************************************************************* */
161 int main(int argc, char *argv[])
162 {
163     struct GModule *module = NULL;
164     N_gwflow_data2d *data = NULL;
```

```
165       N_geom_data *geom = NULL;
166       N_les *les = NULL;
167       N_les_callback_2d *call = NULL;
168       double *tmp_vect = NULL;
169       struct Cell_head region;
170       double error, sor, max_norm = 0, tmp;
171       int maxit, i, inner_count = 0;
172       char * solver;
173       int x, y, stat;
174       N_gradient_field_2d *field = NULL;
175       N_array_2d *xcomp = NULL;
176       N_array_2d *ycomp = NULL;
177       char *buff = NULL;
178
179
180       /* Initialize GRASS */
181       G_gisinit(argv[0]);
182
183       module = G_define_module();
184       module->keywords = _("raster");
185       module->description =
186     _
187     ("Numerical calculation program for transient, confined and unconfined groundwater flow in two dimensions");
188
189       /* Get parameters from user */
190       set_params();
191
192       if (G_parser(argc, argv))
193     exit(EXIT_FAILURE);
194
195       /*Set the maximum iterations */
196       sscanf(param.maxit->answer, "%i", &(maxit));
197       /*Set the calculation error break criteria */
198       sscanf(param.error->answer, "%lf", &(error));
199       sscanf(param.sor->answer, "%lf", &(sor));
200       /*set the solver*/
201       solver = param.solver->answer;
202
203       if (strcmp(solver, N_SOLVER_DIRECT_LU) == 0 && param.sparse->answer)
204     G_fatal_error(_ ("The direct LU solver do not work with sparse matrices"));
205       if (strcmp(solver, N_SOLVER_DIRECT_GAUSS) == 0 && param.sparse->answer)
206     G_fatal_error(_ ("The direct Gauss solver do not work with sparse matrices"));
207
208
209       /*get the current region */
210       G_get_set_window(&region);
211
212       /*allocate the geometry structure  for geometry and area calculation*/
213       geom = N_init_geom_data_2d(&region, geom);
214
215       /*Set the function callback to the groundwater flow function */
216       call = N_alloc_les_callback_2d();
217       N_set_les_callback_2d_func(call, (*N_callback_gwflow_2d));  /*gwflow 2d */
218
219       /*Allocate the groundwater flow data structure */
220       data = N_alloc_gwflow_data2d(geom->cols, geom->rows);
221
222       /* set the groundwater type */
223       if (param.type->answer) {
224     if (strncmp("unconfined", param.type->answer, 10) == 0) {
225         data->gwtype = N_GW_UNCONFINED;
226     }
227     else {
228         data->gwtype = N_GW_CONFINED;
229     }
230       }
231
232       /*Set the calculation time */
233       sscanf(param.dt->answer, "%lf", &(data->dt));
234       G_message("Calculation time: %g", data->dt);
235
236       /*read all input maps into the memory and take care of the
237        * null values.*/
238       N_read_rast_to_array_2d(param.phead->answer, data->phead);
239       N_read_rast_to_array_2d(param.phead->answer, data->phead_start);
240       N_read_rast_to_array_2d(param.status->answer, data->status);
241       N_convert_array_2d_null_to_zero(data->status);
242       N_read_rast_to_array_2d(param.hc_x->answer, data->hc_x);
243       N_convert_array_2d_null_to_zero(data->hc_x);
244       N_read_rast_to_array_2d(param.hc_y->answer, data->hc_y);
245       N_convert_array_2d_null_to_zero(data->hc_y);
246       N_read_rast_to_array_2d(param.q->answer, data->q);
247       N_convert_array_2d_null_to_zero(data->q);
248       N_read_rast_to_array_2d(param.s->answer, data->s);
249       N_convert_array_2d_null_to_zero(data->s);
250       N_read_rast_to_array_2d(param.top->answer, data->top);
251       N_convert_array_2d_null_to_zero(data->top);
252       N_read_rast_to_array_2d(param.bottom->answer, data->bottom);
253       N_convert_array_2d_null_to_zero(data->bottom);
254
```

```
255       /*Reacharge is optional */
256       if (param.r->answer) {
257     N_read_rast_to_array_2d(param.r->answer, data->r);
258     N_convert_array_2d_null_to_zero(data->r);
259       }
260
261       /* Set the inactive values to zero, to assure a no flow boundary */
262       for (y = 0; y < geom->rows; y++) {
263     for (x = 0; x < geom->cols; x++) {
264         stat = (int)N_get_array_2d_d_value(data->status, x, y);
265         if (stat == N_CELL_INACTIVE) {  /*only inactive cells */
266       N_put_array_2d_d_value(data->hc_x, x, y, 0);
267       N_put_array_2d_d_value(data->hc_y, x, y, 0);
268       N_put_array_2d_d_value(data->s, x, y, 0);
269       N_put_array_2d_d_value(data->q, x, y, 0);
270         }
271     }
272       }
273
274
275       /*assemble the linear equation system  and solve it */
276       les = create_solve_les(geom, data, call, solver, maxit, error, sor);
277
278       /* copy the result into the phead array for output or unconfined calculation */
279       copy_result(data->status, data->phead_start, les->x, &region, data->phead);
280
281     /*****************************************************/
282     /*explicite calculation of free groundwater surface */
283     /*****************************************************/
284       if (data->gwtype == N_GW_UNCONFINED) {
285     /* allocate memory and copy the result into a new temporal vector */
286     if (!(tmp_vect = (double *)calloc(les->rows, sizeof(double))))
287         G_fatal_error(_("Out of memory"));
288
289     /*copy data */
290     for (i = 0; i < les->rows; i++)
291         tmp_vect[i] = les->x[i];
292
293     /*count the number of inner iterations */
294     inner_count = 0;
295
296     do {
297         G_message(_("Calculation of unconfined groundwater flow loop %i\n"),
298             inner_count + 1);
299
300         /* we will allocate a new les for each loop */
301         if (les)
302       N_free_les(les);
303
304         /*assemble the linear equation system  and solve it */
305         les = create_solve_les(geom, data, call, solver, maxit, error, sor);
306
307         /*calculate the maximum norm of the groundwater height difference */
308         tmp = 0;
309         max_norm = 0;
310         for (i = 0; i < les->rows; i++) {
311       tmp = fabs(les->x[i] - tmp_vect[i]);
312       if (max_norm < tmp)
313           max_norm = tmp;
314
315       /*copy the result */
316       tmp_vect[i] = les->x[i];
317         }
318
319         G_message(_
320             ("Maximum difference between this and last increment: %g"),
321             max_norm);
322
323         /* copy the result into the phead array */
324         copy_result(data->status, data->phead_start, les->x, &region,
325         data->phead);
326          /**/ inner_count++;
327     }
328     while (max_norm > 0.01 && inner_count < 100);
329
330     if (tmp_vect)
331         free(tmp_vect);
332       }
333
334       /*write the result to the output file */
335       N_write_array_2d_to_rast(data->phead, param.output->answer);
336
337       /*release the memory */
338       if (les)
339     N_free_les(les);
340
341
342       /*Compute the the velocity field if required and write the result into three rast maps */
343       if (param.vector->answer) {
344     field =
```

```
345         N_compute_gradient_field_2d(data->phead, data->hc_x, data->hc_y,
346             geom, NULL);
347
348     xcomp = N_alloc_array_2d(geom->cols, geom->rows, 1, DCELL_TYPE);
349     ycomp = N_alloc_array_2d(geom->cols, geom->rows, 1, DCELL_TYPE);
350
351     N_compute_gradient_field_components_2d(field, xcomp, ycomp);
352
353     G_asprintf(&buff, "%s_x", param.vector->answer);
354     N_write_array_2d_to_rast(xcomp, buff);
355     G_asprintf(&buff, "%s_y", param.vector->answer);
356     N_write_array_2d_to_rast(ycomp, buff);
357     if (buff)
358         G_free(buff);
359
360     if (xcomp)
361         N_free_array_2d(xcomp);
362     if (ycomp)
363         N_free_array_2d(ycomp);
364     if (field)
365         N_free_gradient_field_2d(field);
366       }
367
368
369       if (data)
370     N_free_gwflow_data2d(data);
371       if (geom)
372     N_free_geom_data(geom);
373       if (call)
374     G_free(call);
375
376       return (EXIT_SUCCESS);
377   }
378
379
380   /* ************************************************************************* */
381   /* this function copies the result from the x vector to a N_array_2d struct  */
382   /* ************************************************************************* */
383   void
384   copy_result(N_array_2d * status, N_array_2d * phead_start, double *result,
385         struct Cell_head *region, N_array_2d * target)
386   {
387       int y, x, rows, cols, count, stat;
388       double d1 = 0;
389       DCELL val;
390
391       rows = region->rows;
392       cols = region->cols;
393
394       count = 0;
395       for (y = 0; y < rows; y++) {
396     G_percent(y, rows - 1, 10);
397     for (x = 0; x < cols; x++) {
398         stat = (int)N_get_array_2d_d_value(status, x, y);
399         if (stat == N_CELL_ACTIVE) {  /*only active cells */
400       d1 = result[count];
401       val = (DCELL) d1;
402       count++;
403         }
404         else if (stat == N_CELL_DIRICHLET) {  /*dirichlet cells */
405       d1 = N_get_array_2d_d_value(phead_start, x, y);
406       val = (DCELL) d1;
407         }
408         else {
409       G_set_null_value(&val, 1, DCELL_TYPE);
410         }
411         N_put_array_2d_d_value(target, x, y, val);
412     }
413       }
414
415       return;
416   }
417
418   /* ************************************************************** */
419   /* ***** create and solve the linear equation system ************ */
420   /* ************************************************************** */
421   N_les *create_solve_les(N_geom_data * geom, N_gwflow_data2d * data,
422         N_les_callback_2d * call, const char *solver, int maxit,
423         double error, double sor)
424   {
425
426       N_les *les;
427
428       /*assemble the linear equation system */
429       if (param.sparse->answer)
430     les =
431         N_assemble_les_2d(N_SPARSE_LES, geom, data->status, data->phead,
432               (void *)data, call);
433       else
434     les =
```

```
435         N_assemble_les_2d(N_NORMAL_LES, geom, data->status, data->phead,
436                 (void *)data, call);
437 
438       /*solve the equation system */
439       if (strcmp(solver, N_SOLVER_ITERATIVE_JACOBI) == 0)
440     N_solver_jacobi(les, maxit, sor, error);
441 
442       if (strcmp(solver, N_SOLVER_ITERATIVE_SOR) == 0)
443     N_solver_SOR(les, maxit, sor, error);
444 
445       if (strcmp(solver, N_SOLVER_ITERATIVE_CG) == 0)
446     N_solver_cg(les, maxit, error);
447 
448       if (strcmp(solver, N_SOLVER_ITERATIVE_BICGSTAB) == 0)
449     N_solver_bicgstab(les, maxit, error);
450 
451       if (strcmp(solver, N_SOLVER_DIRECT_LU) == 0)
452     N_solver_lu(les);
453 
454       if (strcmp(solver, N_SOLVER_DIRECT_GAUSS) == 0)
455     N_solver_gauss(les);
456 
457       if (les == NULL)
458     G_fatal_error(_
459             ("Could not create and solve the linear equation system"));
460 
461       return les;
462   }
```

C Quelltext r.solute.transport

```
1
2  /****************************************************************************
3  *
4  * MODULE:       r.solute.transport
5  *
6  * AUTHOR(S):    Original author
7  *               Soeren Gebbert soerengebbert <at> gmx <dot> de
8  *     27 11 2006 Berlin
9  * PURPOSE:      Calculates transient two dimensional solute transport
10 *     in porous media
11 *
12 * COPYRIGHT:    (C) 2006 by the GRASS Development Team
13 *
14 *               This program is free software under the GNU General Public
15 *           License (>=v2). Read the file COPYING that comes with GRASS
16 *           for details.
17 *
18 *****************************************************************************/
19 #include <stdio.h>
20 #include <stdlib.h>
21 #include <string.h>
22 #include <math.h>
23 #include <grass/gis.h>
24 #include <grass/glocale.h>
25 #include <grass/N_pde.h>
26 #include <grass/N_solute_transport.h>
27
28
29 /*- Parameters and global variables ---------------------------------------*/
30 typedef struct
31 {
32     struct Option *output, *phead, *hc_x, *hc_y,
33   *c, *status, *diff_x, *diff_y, *q, *cs, *r, *top, *nf, *cin,
34   *bottom, *vector, *type, *dt, *maxit, *error, *solver, *sor,
35   *al, *at;
36     struct Flag *sparse;
37     struct Flag *cfl;
38 } paramType;
39
40 paramType param;    /*Parameters */
41
42 /*- prototypes ------------------------------------------------------------*/
43 void set_params();    /*Fill the paramType structure */
44 void copy_result(N_array_2d * status, N_array_2d * c_start, double *result,
45      struct Cell_head *region, N_array_2d * target);
46 N_les *create_solve_les(N_geom_data * geom, N_solute_transport_data2d * data,
47       N_les_callback_2d * call, const char *solver, int maxit,
48       double error, double sor);
49
50 /* ************************************************************************* */
51 /* Set up the arguments we are expecting ********************************** */
52 /* ************************************************************************* */
53 void set_params()
54 {
55     param.c = G_define_option();
56     param.c->key = "c";
57     param.c->type = TYPE_STRING;
58     param.c->required = YES;
59     param.c->gisprompt = "old,raster,raster";
60     param.c->description = _("The initial concentration in [g/m^3]");
61
62     param.phead = G_define_option();
63     param.phead->key = "phead";
64     param.phead->type = TYPE_STRING;
65     param.phead->required = YES;
66     param.phead->gisprompt = "old,raster,raster";
67     param.phead->description = _("The piezometric head in [m]");
68
69     param.hc_x = G_define_option();
70     param.hc_x->key = "hc_x";
71     param.hc_x->type = TYPE_STRING;
72     param.hc_x->required = YES;
73     param.hc_x->gisprompt = "old,raster,raster";
74     param.hc_x->description =
```

```
75      _("The x-part of the hydraulic conductivity tensor in [m/s]");
76
77        param.hc_y = G_define_option();
78        param.hc_y->key = "hc_y";
79        param.hc_y->type = TYPE_STRING;
80        param.hc_y->required = YES;
81        param.hc_y->gisprompt = "old,raster,raster";
82        param.hc_y->description =
83      _("The y-part of the hydraulic conductivity tensor in [m/s]");
84
85
86        param.status = G_define_option();
87        param.status->key = "status";
88        param.status->type = TYPE_STRING;
89        param.status->required = YES;
90        param.status->gisprompt = "old,raster,raster";
91        param.status->description =
92      _
93      ("The status for each cell, = 0 - inactive cell, 1 - active cell, 2 - dirichlet- and 3 - transfer boundary condition");
94
95        param.diff_x = G_define_option();
96        param.diff_x->key = "diff_x";
97        param.diff_x->type = TYPE_STRING;
98        param.diff_x->required = YES;
99        param.diff_x->gisprompt = "old,raster,raster";
100       param.diff_x->description =
101     _("The x-part of the diffusion tensor in [m/s]");
102
103       param.diff_y = G_define_option();
104       param.diff_y->key = "diff_y";
105       param.diff_y->type = TYPE_STRING;
106       param.diff_y->required = YES;
107       param.diff_y->gisprompt = "old,raster,raster";
108       param.diff_y->description =
109     _("The y-part of the diffusion tensor in [m/s]");
110
111       param.q = G_define_option();
112       param.q->key = "q";
113       param.q->type = TYPE_STRING;
114       param.q->required = NO;
115       param.q->gisprompt = "old,raster,raster";
116       param.q->description = _("groundwater sources and sinks in [m^3/s]");
117
118       param.cin = G_define_option();
119       param.cin->key = "cin";
120       param.cin->type = TYPE_STRING;
121       param.cin->required = NO;
122       param.cin->gisprompt = "old,raster,raster";
123       param.cin->description = _("concentration sources and sinks in [g/m^3]");
124
125       param.cs = G_define_option();
126       param.cs->key = "cs";
127       param.cs->type = TYPE_STRING;
128       param.cs->required = YES;
129       param.cs->gisprompt = "old,raster,raster";
130       param.cs->description = _("concentration sources and sinks in [g/m^3]");
131
132       param.r = G_define_option();
133       param.r->key = "R";
134       param.r->type = TYPE_STRING;
135       param.r->required = YES;
136       param.r->gisprompt = "old,raster,raster";
137       param.r->description = _("Retardation factor [m/s]");
138
139       param.nf = G_define_option();
140       param.nf->key = "nf";
141       param.nf->type = TYPE_STRING;
142       param.nf->required = YES;
143       param.nf->gisprompt = "old,raster,raster";
144       param.nf->description = _("Effective porosity");
145
146       param.top = G_define_option();
147       param.top->key = "top";
148       param.top->type = TYPE_STRING;
149       param.top->required = YES;
150       param.top->gisprompt = "old,raster,raster";
151       param.top->description = _("Top surface of the aquifer in [m]");
152
153       param.bottom = G_define_option();
154       param.bottom->key = "bottom";
155       param.bottom->type = TYPE_STRING;
156       param.bottom->required = YES;
157       param.bottom->gisprompt = "old,raster,raster";
158       param.bottom->description = _("Bottom surface of the aquifer in [m]");
159
160       param.output = G_define_option();
161       param.output->key = "output";
162       param.output->type = TYPE_STRING;
163       param.output->required = YES;
164       param.output->gisprompt = "new,raster,raster";
```

```
    param.output->description =
_
("The result of the numericalsolute transport calculation will be written to this map.");

    param.vector = G_define_option();
    param.vector->key = "velocity";
    param.vector->type = TYPE_STRING;
    param.vector->required = NO;
    param.vector->gisprompt = "new,raster,raster";
    param.vector->description =
_
("Calculate the groundwater distance velocity vector field and write the x, and y components to maps named name_[xy]. name is the basename fo


    param.type = G_define_option();
    param.type->key = "type";
    param.type->type = TYPE_STRING;
    param.type->required = NO;
    param.type->answer = "confined";
    param.type->options = "confined,unconfined";
    param.type->description =
_("Which type of groundwater flow? confined or unconfined.");

    param.dt = N_define_standard_option(N_OPT_CALC_TIME);
    param.maxit = N_define_standard_option(N_OPT_MAX_ITERATIONS);
    param.error = N_define_standard_option(N_OPT_ITERATION_ERROR);
    param.solver = N_define_standard_option(N_OPT_SOLVER_UNSYMM);
    param.sor = N_define_standard_option(N_OPT_SOR_VALUE);

    param.al = G_define_option();
    param.al->key = "al";
    param.al->type = TYPE_DOUBLE;
    param.al->required = NO;
    param.al->answer = "0.0";
    param.al->description =
_("The longditudinal dispersivity length.");

    param.at = G_define_option();
    param.at->key = "at";
    param.at->type = TYPE_DOUBLE;
    param.at->required = NO;
    param.at->answer = "0.0";
    param.at->description =
_("The transversal dispersivity length.");


    param.sparse = G_define_flag();
    param.sparse->key = 's';
    param.sparse->description =
_
("Use a sparse linear equation system, only available with iterative solvers");

    param.cfl = G_define_flag();
    param.cfl->key = 'c';
    param.cfl->description =
_
("Use the Courant-Friedrichs-Lewy criteria for time step calculation");


}

/* ************************************************************************* */
/* Main function ********************************************************** */
/* ************************************************************************* */
int main(int argc, char *argv[])
{
    struct GModule *module = NULL;
    N_solute_transport_data2d *data = NULL;
    N_geom_data *geom = NULL;
    N_les *les = NULL;
    N_les_callback_2d *call = NULL;
    struct Cell_head region;
    double error, sor;
    char *solver;
    int x, y, stat, i, maxit = 1;
    N_array_2d *xcomp = NULL;
    N_array_2d *ycomp = NULL;
    N_array_2d *hc_x = NULL;
    N_array_2d *hc_y = NULL;
    N_array_2d *phead = NULL;
    char *buff;

    double time_step, cfl, length, time_loops, time_sum;

    /* Initialize GRASS */
    G_gisinit(argv[0]);

    module = G_define_module();
    module->keywords = _("raster");
    module->description =
```

```
255     _
256     ("Numerical calculation program for transient, confined and unconfined solute transport in two dimensions");
257
258       /* Get parameters from user */
259       set_params();
260
261       if (G_parser(argc, argv))
262     exit(EXIT_FAILURE);
263
264       /*Set the maximum iterations */
265       sscanf(param.maxit->answer, "%i", &(maxit));
266       /*Set the calculation error break criteria */
267       sscanf(param.error->answer, "%lf", &(error));
268       sscanf(param.sor->answer, "%lf", &(sor));
269       /*Set the solver */
270       solver = param.solver->answer;
271
272       if (strcmp(solver, N_SOLVER_DIRECT_LU) == 0 && param.sparse->answer)
273     G_fatal_error(_
274             ("The direct LU solver do not work with sparse matrices"));
275       if (strcmp(solver, N_SOLVER_DIRECT_GAUSS) == 0 && param.sparse->answer)
276     G_fatal_error(_
277             ("The direct Gauss solver do not work with sparse matrices"));
278
279
280       /*get the current region */
281       G_get_set_window(&region);
282
283       /*allocate the geometry structure for geometry and area calculation */
284       geom = N_init_geom_data_2d(&region, geom);
285
286       /*Set the function callback to the groundwater flow function */
287       call = N_alloc_les_callback_2d();
288       N_set_les_callback_2d_func(call, (*N_callback_solute_transport_2d));  /*solute_transport 2d */
289
290       /*Allocate the groundwater flow data structure */
291       data = N_alloc_solute_transport_data2d(geom->cols, geom->rows);
292
293       /*the dispersivity lengths*/
294       sscanf(param.al->answer, "%lf", &(data->al));
295       sscanf(param.at->answer, "%lf", &(data->at));
296
297       /* set the groundwater type */
298       if (param.type->answer) {
299     if (strncmp("unconfined", param.type->answer, 10) == 0) {
300         ;
301     }
302     else {
303         ;
304     }
305       }
306
307       /*Set the calculation time */
308       sscanf(param.dt->answer, "%lf", &(data->dt));
309
310       /*read all input maps into the memory and take care of the
311        * null values.*/
312       N_read_rast_to_array_2d(param.c->answer, data->c);
313       N_convert_array_2d_null_to_zero(data->c);
314       N_read_rast_to_array_2d(param.c->answer, data->c_start);
315       N_convert_array_2d_null_to_zero(data->c_start);
316       N_read_rast_to_array_2d(param.status->answer, data->status);
317       N_convert_array_2d_null_to_zero(data->status);
318       N_read_rast_to_array_2d(param.diff_x->answer, data->diff_x);
319       N_convert_array_2d_null_to_zero(data->diff_x);
320       N_read_rast_to_array_2d(param.diff_y->answer, data->diff_y);
321       N_convert_array_2d_null_to_zero(data->diff_y);
322       N_read_rast_to_array_2d(param.q->answer, data->q);
323       N_convert_array_2d_null_to_zero(data->q);
324       N_read_rast_to_array_2d(param.nf->answer, data->nf);
325       N_convert_array_2d_null_to_zero(data->nf);
326       N_read_rast_to_array_2d(param.cs->answer, data->cs);
327       N_convert_array_2d_null_to_zero(data->cs);
328       N_read_rast_to_array_2d(param.top->answer, data->top);
329       N_convert_array_2d_null_to_zero(data->top);
330       N_read_rast_to_array_2d(param.bottom->answer, data->bottom);
331       N_convert_array_2d_null_to_zero(data->bottom);
332       N_read_rast_to_array_2d(param.r->answer, data->R);
333       N_convert_array_2d_null_to_zero(data->R);
334
335       if(param.cin->answer) {
336         N_read_rast_to_array_2d(param.cin->answer, data->cin);
337         N_convert_array_2d_null_to_zero(data->cin);
338       }
339
340       /*initiate the values for velocity calculation*/
341       hc_x = N_alloc_array_2d(geom->cols, geom->rows, 1, DCELL_TYPE);
342       hc_x = N_read_rast_to_array_2d(param.hc_x->answer, hc_x);
343       N_convert_array_2d_null_to_zero(hc_x);
344       hc_y = N_alloc_array_2d(geom->cols, geom->rows, 1, DCELL_TYPE);
```

```
345       hc_y = N_read_rast_to_array_2d(param.hc_y->answer, hc_y);
346       N_convert_array_2d_null_to_zero(hc_y);
347       phead = N_alloc_array_2d(geom->cols, geom->rows, 1, DCELL_TYPE);
348       phead = N_read_rast_to_array_2d(param.phead->answer, phead);
349       N_convert_array_2d_null_to_zero(phead);
350
351       /* Set the inactive values to zero, to assure a no flow boundary */
352       for (y = 0; y < geom->rows; y++) {
353     for (x = 0; x < geom->cols; x++) {
354         stat = (int)N_get_array_2d_d_value(data->status, x, y);
355         if (stat == N_CELL_INACTIVE) {  /*only inactive cells */
356       N_put_array_2d_d_value(data->diff_x, x, y, 0);
357       N_put_array_2d_d_value(data->diff_y, x, y, 0);
358       N_put_array_2d_d_value(data->cs, x, y, 0);
359       N_put_array_2d_d_value(data->q, x, y, 0);
360         }
361     }
362       }
363
364       /*compute the velocities */
365       N_math_array_2d(hc_x, data->nf, hc_x, N_ARRAY_DIV);
366       N_math_array_2d(hc_y, data->nf, hc_y, N_ARRAY_DIV);
367       N_compute_gradient_field_2d(phead, hc_x, hc_y, geom, data->grad);
368
369       N_print_gradient_field_2d_info(data->grad);
370
371       /*Now compute the dispersivity tensor*/
372       N_calc_solute_transport_disptensor_2d(data);
373
374       /*************************************/
375       /*the Courant-Friedrichs-Lewy criteria */
376       /*Compute the correct time step */
377       if (geom->dx > geom->dy)
378     length = geom->dx;
379       else
380     length = geom->dy;
381
382       if (fabs(data->grad->max) > fabs(data->grad->min)) {
383     cfl = (double)data->dt * fabs(data->grad->max) / length;
384     time_step = 1*length / fabs(data->grad->max);
385       }
386       else {
387     cfl = (double)data->dt * fabs(data->grad->min) / length;
388     time_step = 1*length / fabs(data->grad->min);
389       }
390
391       G_message("The Courant-Friedrichs-Lewy criteria is %g it should be within [0:1]", cfl);
392       G_message("The largest stable time step is %g", time_step);
393
394       /*Set the number of inner loops and the time step*/
395       if (data->dt > time_step && param.cfl->answer) {
396     /*safe the user time step */
397     time_sum = data->dt;
398     time_loops = data->dt / time_step;
399     time_loops = floor(time_loops) + 1;
400     data->dt = data->dt / time_loops;
401     G_message("Number of inner loops is %g", time_loops);
402     G_message("Time step for each loop %g", data->dt);
403       }
404       else {
405           if(data->dt > time_step)
406     G_warning("The time step is to large: %gs. The largest time step should be of size %gs.", data->dt, time_step);
407
408     time_loops = 1;
409       }
410
411       N_free_array_2d(phead);
412       N_free_array_2d(hc_x);
413       N_free_array_2d(hc_y);
414
415        /*Compute for each time step*/
416       for (i = 0; i < time_loops; i++) {
417           G_message("Time step %i with time sum %g", i + 1, (i + 1)*data->dt);
418
419     /*assemble the linear equation system  and solve it */
420     les = create_solve_les(geom, data, call, solver, maxit, error, sor);
421
422     /* copy the result into the c array for output */
423     copy_result(data->status, data->c_start, les->x, &region, data->c);
424
425           if (les)
426         N_free_les(les);
427
428     /*Set the start array*/
429     N_copy_array_2d(data->c, data->c_start);
430
431     /*Set the transmission boundary*/
432     N_calc_solute_transport_transmission_2d(data);
433
434       }
```

```
435
436       /*write the result to the output file */
437       N_write_array_2d_to_rast(data->c, param.output->answer);
438
439       /*Compute the the velocity field if required and write the result into three rast maps */
440       if (param.vector->answer) {
441     xcomp = N_alloc_array_2d(geom->cols, geom->rows, 1, DCELL_TYPE);
442     ycomp = N_alloc_array_2d(geom->cols, geom->rows, 1, DCELL_TYPE);
443
444     N_compute_gradient_field_components_2d(data->grad, xcomp, ycomp);
445
446     G_asprintf(&buff, "%s_x", param.vector->answer);
447     N_write_array_2d_to_rast(xcomp, buff);
448     G_asprintf(&buff, "%s_y", param.vector->answer);
449     N_write_array_2d_to_rast(ycomp, buff);
450     if (buff)
451         G_free(buff);
452
453     if (xcomp)
454         N_free_array_2d(xcomp);
455     if (ycomp)
456         N_free_array_2d(ycomp);
457       }
458
459
460       if (data)
461     N_free_solute_transport_data2d(data);
462       if (geom)
463     N_free_geom_data(geom);
464       if (call)
465     G_free(call);
466
467       return (EXIT_SUCCESS);
468   }
469
470
471   /* ************************************************************************** */
472   /* this function copies the result from the x vector to a N_array_2d array * */
473   /* ************************************************************************** */
474   void
475   copy_result(N_array_2d * status, N_array_2d * c_start, double *result,
476         struct Cell_head *region, N_array_2d * target)
477   {
478       int y, x, rows, cols, count, stat;
479       double d1 = 0;
480       DCELL val;
481
482       rows = region->rows;
483       cols = region->cols;
484
485       count = 0;
486       for (y = 0; y < rows; y++) {
487     G_percent(y, rows - 1, 10);
488     for (x = 0; x < cols; x++) {
489         stat = (int)N_get_array_2d_d_value(status, x, y);
490         if (stat == N_CELL_ACTIVE) {  /*only active cells */
491       d1 = result[count];
492       val = (DCELL) d1;
493       count++;
494         }
495         else if (stat == N_CELL_DIRICHLET || stat == N_CELL_TRANSMISSION) { /*dirichlet cells */
496       d1 = N_get_array_2d_d_value(c_start, x, y);
497       val = (DCELL) d1;
498         }
499         else {
500       G_set_null_value(&val, 1, DCELL_TYPE);
501         }
502         N_put_array_2d_d_value(target, x, y, val);
503     }
504       }
505
506       return;
507   }
508
509   /* ************************************************************* */
510   /* ***** create and solve the linear equation system ************ */
511   /* ************************************************************* */
512   N_les *create_solve_les(N_geom_data * geom, N_solute_transport_data2d * data,
513         N_les_callback_2d * call, const char *solver, int maxit,
514         double error, double sor)
515   {
516
517       N_les *les;
518
519       /*assemble the linear equation system */
520       if (param.sparse->answer)
521     les =
522         N_assemble_les_2d(N_SPARSE_LES, geom, data->status, data->c,
523               (void *)data, call);
524       else
```

```
525     les =
526         N_assemble_les_2d(N_NORMAL_LES, geom, data->status, data->c,
527               (void *)data, call);
528
529       /*solve the equation system */
530       if (strcmp(solver, N_SOLVER_ITERATIVE_JACOBI) == 0)
531     N_solver_jacobi(les, maxit, sor, error);
532
533       if (strcmp(solver, N_SOLVER_ITERATIVE_SOR) == 0)
534     N_solver_SOR(les, maxit, sor, error);
535
536       if (strcmp(solver, N_SOLVER_ITERATIVE_BICGSTAB) == 0)
537     N_solver_bicgstab(les, maxit, error);
538
539       if (strcmp(solver, N_SOLVER_DIRECT_LU) == 0)
540     N_solver_lu(les);
541
542       if (strcmp(solver, N_SOLVER_DIRECT_GAUSS) == 0)
543     N_solver_gauss(les);
544
545       if (les == NULL)
546     G_fatal_error(_
547             ("Could not create and solve the linear equation system"));
548
549       return les;
550   }
```

D Quelltext der Shell Scripte

D.1 Validierung des Strömungsmodells Beispiel 6.1

```
#!/bin/sh
# Shellscript to verify r.gwflow calculation, this calculation is based on
# the example at page 133 of the following book:
# author = "Kinzelbach, W. and Rausch, R.",
# title = "Grundwassermodellierung",
# publisher = "Gebr{\"u}der Borntraeger (Berlin, Stuttgart)",
# year = "1995"
#
# set the region
g.region res=100 n=700 s=0 w=0 e=700

r.mapcalc "phead=50"
r.mapcalc "status=if(col() == 1 || col() == 7 , 2, 1)"
r.mapcalc "well=if((row() == 4 && col() == 4), -0.1, 0)"
r.mapcalc "hydcond=0.0005"
r.mapcalc "recharge=0"
r.mapcalc "top_conf=20"
r.mapcalc "bottom=0"
r.mapcalc "syield=0.0001"
r.mapcalc "null=0.0"

#First compute the steady state groundwater flow
r.gwflow --o solver=gauss top=top_conf bottom=bottom phead=phead\
 status=status hc_x=hydcond hc_y=hydcond q=well s=syield\
 r=reacharge output=gwresult_conf dt=500 type=confined

count=500

while [ `expr $count \< 10000` -eq 1 ] ; do
  r.gwflow --o solver=gauss top=top_conf bottom=bottom phead=gwresult_conf\
     status=status hc_x=hydcond hc_y=hydcond q=well s=syield\
     r=reacharge output=gwresult_conf dt=500 type=confined
  count=`expr $count + 500`
done

#create the visualization
d.mon start=x0
d.mon select=x0
d.erase
d.rast gwresult_conf
d.rast.num gwresult_conf dp=2
d.barscale at=1,10 &
echo "Groundwater flow 10.000s" | d.text size=6 color=black

export GRASS_WIDTH=640
export GRASS_HEIGHT=480

#export as png and convert into eps and pdf
export GRASS_TRUECOLOR=TRUE
export GRASS_PNGFILE=valid_calc_7x7.png
d.mon start=PNG
d.mon select=PNG
d.rast gwresult_conf
d.rast.num gwresult_conf dp=2
d.barscale at=1,10 &
echo "Groundwater flow 10.000s" | d.text size=6 color=black
d.mon stop=PNG
convert valid_calc_7x7.png valid_calc_7x7.eps
convert valid_calc_7x7.png valid_calc_7x7.pdf
```

D.2 Validierung des Strömungsmodells Beispiel 6.10

```
#!/bin/sh
# Shellscript to verify r.gwflow calculation, this calculation is based on
# the example at page 167 of the following book:
# author = "Kinzelbach, W. and Rausch, R.",
# title = "Grundwassermodellierung",
# publisher = "Gebr{\"u}der Borntraeger (Berlin, Stuttgart)",
# year = "1995"
#
# set the region
g.region res=50 n=950 s=0 w=0 e=2000

r.mapcalc "phead= if(row() == 19, 5, 3)"
r.mapcalc "status=if((col() == 1 && row() == 13) ||\
                     (col() == 1 && row() == 14) ||\
          (col() == 2 && row() == 13) ||\
          (col() == 2 && row() == 14) ||\
          (row() == 19), 2, 1)"

r.mapcalc "well=0.0"
r.mapcalc "hydcond=0.001"
r.mapcalc "reacharge=0.000000006"
r.mapcalc "top=20"
r.mapcalc "bottom=0"
r.mapcalc "syield=0.001"
r.mapcalc "null=0.0"

#compute a steady state groundwater flow
r.gwflow --o -s solver=cg top=top bottom=bottom phead=phead \
 status=status hc_x=hydcond hc_y=hydcond q=well s=syield \
 r=reacharge output=gwresult dt=864000000000 type=unconfined

# create contour lines
r.contour input=gwresult output=gwresult_contour step=0.2 --o
#create flow lines
r.flow elevin=gwresult flout=gwresult_flow skip=3 --o

#create the visualization
d.mon start=x0
d.mon select=x0
d.erase
d.rast gwresult
d.vect gwresult_flow render=1 color=grey
d.vect gwresult_contour render=1 color=black display=attr,shape attrcol=level lsize=16 lcolor=black
d.legend at=8,12,15,85 map=gwresult
d.barscale at=1,10 &
echo "Groundwater flow steady state" | d.text size=6 color=black

export GRASS_WIDTH=640
export GRASS_HEIGHT=480
#export as png and convert into eps and pdf
export GRASS_TRUECOLOR=TRUE
export GRASS_PNGFILE=Excavation_pit.png
d.mon start=PNG
d.mon select=PNG
d.rast gwresult
d.vect gwresult_flow render=1 color=grey
d.vect gwresult_contour render=1 color=black display=attr,shape attrcol=level lsize=16 lcolor=black
d.legend at=8,12,15,85 map=gwresult
d.barscale at=1,10 &
echo "Groundwater flow steady state" | d.text size=6 color=black
d.mon stop=PNG
convert Excavation_pit.png Excavation_pit.eps
convert Excavation_pit.png Excavation_pit.pdf
```

D.3 Validierung des Transportmodells Beispiel 1.1 und 1.2

```
#!/bin/sh
# Shellscript to verify r.solute.transport calculation, this calculation is based on
# the example 1.1 and 1.2 at page 179 of the following book:
# title = "Str{\"o}mungs und Transportmodellierung",
# author = "Lege, T. and Kolditz, O. and Zielke W.",
# publisher = "Springer (Berlin; Heidelberg; New York; Barcelona;
#                        Hongkong; London; Mailand; Paris; Singapur; Tokio)",
# edition = "2. Auflage",
# year = "1996",
# series = "Handbuch zur Erkundung des Untergrundes von Deponien und Altlasten"
#
# set the region
g.region res=50 res3=1 t=10 b=0 n=1000 s=0 w=0 e=2000
r.mapcalc "phead_1=if(col() == 1 , 276, 50)"
r.mapcalc "status_1=if(col() == 1 || col() == 40 , 2, 1)"
#r.mapcalc "well_1=if((row() == 10 && col() == 10), 0.00281, 0)"
```

```
r.mapcalc "well_1=0.0"
r.mapcalc "hydcond_1=0.0001"
r.mapcalc "reacharge_1=0"
r.mapcalc "top_conf_1=25"
r.mapcalc "bottom_1=0"
r.mapcalc "poros_1=1.0"
r.mapcalc "poros_2=0.17"
r.mapcalc "syield_1=0.0001"
r.mapcalc "null_1=0.0"

#generate the transport data
r.mapcalc "c_1=if(col() == 10 && row() == 10 , 0.0, 0.0)"
r.mapcalc "cs_1=if(col() == 10 && row() == 10 , 0.001388888888, 0.0)"
r.mapcalc "cin_1=if(col() == 30 && row() == 10 , 0.0, 0.0)"
r.mapcalc "tstatus_1=if(col() == 1 || col() == 40 , 3, 1)"
r.mapcalc "diff_1=0.0"
r.mapcalc "R_1=1.0"

AL=100
AT=10
export GRASS_WIDTH=640
export GRASS_HEIGHT=480

#First compute a steady state groundwater flow
r.gwflow --o -s solver=cg top=top_conf_1 bottom=bottom_1 phead=phead_1\
 status=status_1 hc_x=hydcond_1 hc_y=hydcond_1 \
 q=well_1 s=syield_1 r=reacharge_1 output=gwresult_conf_1\
 dt=8640000000000 type=confined

r.colors map=gwresult_conf_1 color=gyr

# create contour lines
r.contour input=gwresult_conf_1 output=gwresult_conf_contour_1 step=10 --o

#create a new monitor
d.mon start=x2
d.mon select=x2
d.erase
d.rast gwresult_conf_1
d.rast.num -f map=gwresult_conf_1 grid_color=gray
#d.vect gwresult_conf_contour_1 color=black display=shape,attr attrcol=level lcolor=black
d.legend -f at=8,12,15,85 map=gwresult_conf_1 labelnum=5 color=black thin=500
d.barscale at=1,10
echo "Groundwater flow steady state" | d.text size=6 color=black
d.out.png output=seguin_verify_gwflow.png
convert seguin_verify_gwflow.png seguin_verify_gwflow.eps
convert seguin_verify_gwflow.png seguin_verify_gwflow.pdf

###
### compute with low velocity
###

r.solute.transport --o -cs maxit=100 solver=bicgstab top=top_conf_1 bottom=bottom_1\
 phead=gwresult_conf_1 status=tstatus_1 hc_x=hydcond_1 hc_y=hydcond_1\
 R=R_1 cs=cs_1 q=well_1 nf=poros_1 output=stresult_conf_1 dt=86400000\
 diff_x=diff_1 diff_y=diff_1 cin=cin_1 c=c_1 al=$AL at=$AT velocity=stresult_conf_vel_1

eval `r.univar stresult_conf_1 -g`

r.mapcalc "cont_map = stresult_conf_1/$max"
r.mapcalc "stresult_conf_vel_1_x = stresult_conf_vel_1_x *24.0 * 3600.0"
r.info stresult_conf_vel_1_x
r.contour input=cont_map output=stresult_conf_contour_1 step=0.1 --o

#create a new monitor
d.mon start=x0
d.mon select=x0
d.erase
d.rast cont_map
d.vect stresult_conf_contour_1 color=grey display=shape,attr attrcol=level
d.legend at=8,12,15,85 map=cont_map labelnum=5 color=black thin=500
d.barscale at=1,10
echo "Solute transport 1000d al=$AL at=$AT" | d.text size=6 color=black
d.out.png output=seguin_verify_a.png
convert seguin_verify_a.png seguin_verify_a.eps
convert seguin_verify_a.png seguin_verify_a.pdf

###
### compute with high velocity
###

r.solute.transport --o -cs maxit=100 solver=bicgstab top=top_conf_1 bottom=bottom_1\
 phead=gwresult_conf_1 status=tstatus_1 hc_x=hydcond_1 hc_y=hydcond_1\
 R=R_1 cs=cs_1 q=well_1 nf=poros_2 output=stresult_conf_2 dt=86400000\
 diff_x=diff_1 diff_y=diff_1 cin=cin_1 c=c_1 al=$AL at=$AT velocity=stresult_conf_vel_2

eval `r.univar stresult_conf_2 -g`

r.mapcalc "cont_map_a = stresult_conf_2/$max"
r.mapcalc "stresult_conf_vel_2_x = stresult_conf_vel_2_x *24.0 * 3600.0"
```

```
r.info stresult_conf_vel_2_x
r.contour input=cont_map_a output=stresult_conf_contour_2 step=0.1 --o

#create a new monitor
d.mon start=x1
d.mon select=x1
d.erase
d.rast cont_map_a
d.vect stresult_conf_contour_2 color=grey display=shape,attr attrcol=level
d.legend at=8,12,15,85 map=cont_map_a labelnum=5 color=black thin=500
d.barscale at=1,10
echo "Solute transport 1000d al=$AL at=$AT" | d.text size=6 color=black
d.out.png output=seguin_verify_b.png
convert seguin_verify_b.png seguin_verify_b.eps
convert seguin_verify_b.png seguin_verify_b.pdf
```

D.4 Validierung des Transportmodells Beispiel 2.1 und 2.2

```
#!/bin/sh
# Shellscript to verify r.solute.transport calculation, this calculation is based on
# the example 2.1 and 2.2 at page 181 of the following book:
# title = "Str{\"o}mungs und Transportmodellierung",
# author = "Lege, T. and Kolditz, O. and Zielke W.",
# publisher = "Springer (Berlin; Heidelberg; New York; Barcelona;
#                         Hongkong; London; Mailand; Paris; Singapur; Tokio)",
# edition = "2. Auflage",
# year = "1996",
# series = "Handbuch zur Erkundung des Untergrundes von Deponien und Altlasten"
#
# set the region
g.region res=50 res3=1 t=10 b=0 n=1000 s=0 w=0 e=2000
r.mapcalc "phead_1=if(col() == 1 , 276, 50)"
r.mapcalc "status_1=if(col() == 1 || col() == 40 , 2, 1)"
r.mapcalc "well_1=if((row() == 10 && col() == 10), 0.02818287, 0)"
#r.mapcalc "well_1=0.0"
r.mapcalc "hydcond_1=0.0001"
r.mapcalc "reacharge_1=0"
r.mapcalc "top_conf_1=25"
r.mapcalc "bottom_1=0"
r.mapcalc "poros_1=0.17"
r.mapcalc "syield_1=0.0001"
r.mapcalc "null_1=0.0"

#generate the transport data
r.mapcalc "c_1=if(col() == 10 && row() == 10 , 0.0, 0.0)"
r.mapcalc "cs_1=if(col() == 10 && row() == 10 , 0.0, 0.0)"
r.mapcalc "cin_1=if(col() == 10 && row() == 10 , 0.0000231481, 0.0)"
r.mapcalc "tstatus_1=if(col() == 1 || col() == 40 , 3, 1)"
r.mapcalc "diff_1=0.0"
r.mapcalc "R_1=1.0"

AL=50
AT=5
export GRASS_WIDTH=640
export GRASS_HEIGHT=480

#First compute a steady state groundwater flow
r.gwflow --o -s solver=cg top=top_conf_1 bottom=bottom_1 phead=phead_1\
 status=status_1 hc_x=hydcond_1 hc_y=hydcond_1 \
 q=well_1 s=syield_1 r=reacharge_1 output=gwresult_conf_1\
 dt=8640000000000 type=confined

r.colors map=gwresult_conf_1 color=gyr

# create contour lines
r.contour input=gwresult_conf_1 output=gwresult_conf_contour_1 step=10 --o

#create a new monitor
d.mon start=x2
d.mon select=x2
d.erase
d.rast gwresult_conf_1
#d.rast.num -f map=gwresult_conf_1 grid_color=gray
d.vect gwresult_conf_contour_1 color=black display=shape,attr attrcol=level lcolor=black
d.legend -f at=8,12,15,85 map=gwresult_conf_1 labelnum=5 color=black thin=500
d.barscale at=1,10
echo "Groundwater flow steady state" | d.text size=6 color=black
d.out.png output=seguin_verify_gwflow_well.png
convert seguin_verify_gwflow_well.png seguin_verify_gwflow_well.eps
convert seguin_verify_gwflow_well.png seguin_verify_gwflow_well.pdf

###
```

```
### compute with low velocity
###

r.solute.transport --o -cs error=0.000000000000001 maxit=1000 solver=bicgstab top=top_conf_1 bottom=bottom_1\
 phead=gwresult_conf_1 status=tstatus_1 hc_x=hydcond_1 hc_y=hydcond_1\
 R=R_1 cs=cs_1 q=well_1 nf=poros_1 output=stresult_conf_1 dt=21600000\
 diff_x=diff_1 diff_y=diff_1 cin=cin_1 c=c_1 al=$AL at=$AT velocity=stresult_conf_vel_1

eval `r.univar stresult_conf_1 -g`

r.mapcalc "cont_map = stresult_conf_1/$max"
r.mapcalc "stresult_conf_vel_1_x = stresult_conf_vel_1_x *24.0 * 3600.0"
r.info stresult_conf_vel_1_x
r.contour input=cont_map output=stresult_conf_contour_1 step=0.1 --o

#create a new monitor
d.mon start=x0
d.mon select=x0
d.erase
d.rast cont_map
d.vect stresult_conf_contour_1 color=grey display=shape,attr attrcol=level
d.legend at=8,12,15,85 map=cont_map labelnum=5 color=black thin=500
d.barscale at=1,10
echo "Solute transport 250d al=$AL at=$AT" | d.text size=6 color=black
d.out.png output=seguin_verify_c.png
convert seguin_verify_c.png seguin_verify_c.eps
convert seguin_verify_c.png seguin_verify_c.pdf

###
### compute different dispersion
###

AL=10
AT=1

r.solute.transport --o -cs error=0.000000000000001 maxit=1000 solver=bicgstab top=top_conf_1 bottom=bottom_1\
 phead=gwresult_conf_1 status=tstatus_1 hc_x=hydcond_1 hc_y=hydcond_1\
 R=R_1 cs=cs_1 q=well_1 nf=poros_1 output=stresult_conf_2 dt=21600000\
 diff_x=diff_1 diff_y=diff_1 cin=cin_1 c=c_1 al=$AL at=$AT velocity=stresult_conf_vel_2

eval `r.univar stresult_conf_2 -g`

r.mapcalc "cont_map_a = stresult_conf_2/$max"
r.mapcalc "stresult_conf_vel_2_x = stresult_conf_vel_2_x *24.0 * 3600.0"
r.info stresult_conf_vel_2_x
r.contour input=cont_map_a output=stresult_conf_contour_2 step=0.1 --o

#create a new monitor
d.mon start=x1
d.mon select=x1
d.erase
d.rast cont_map_a
d.vect stresult_conf_contour_2 color=grey display=shape,attr attrcol=level
d.legend at=8,12,15,85 map=cont_map_a labelnum=5 color=black thin=500
d.barscale at=1,10
echo "Solute transport 250d al=$AL at=$AT" | d.text size=6 color=black
d.out.png output=seguin_verify_d.png
convert seguin_verify_d.png seguin_verify_d.eps
convert seguin_verify_d.png seguin_verify_d.pdf
```

D.5 Punktueller Stoffeintrag in ein inhomogenes Strömungsfeld

```
#!/bin/sh
# set the region accordingly
g.region res=1 res3=1 t=10 b=0 n=100 s=0 w=0 e=200
r.mapcalc "phead=if(col() == 1 , 50, 40)"
r.mapcalc "phead=if(col() ==200  , 45.0 + row()/40.0, phead)"
r.mapcalc "status=if(col() == 1 || col() == 200 , 2, 1)"
r.mapcalc "well=if((row() == 50 && col() == 175) || (row() == 10 && col() == 135) , -0.001, 0)"
r.mapcalc "hydcond=0.00005"
r.mapcalc "reacharge=0.000000006"
r.mapcalc "top_conf=20"
r.mapcalc "top_unconf=60"
r.mapcalc "bottom=0"
r.mapcalc "poros=0.17"
r.mapcalc "syield=0.0001"
r.mapcalc "null=0.0"


#First compute a steady state groundwater flow
r.gwflow --o -s solver=cg top=top_conf bottom=bottom phead=phead status=status hc_x=hydcond hc_y=hydcond \
```

```
20    q=well s=syield r=reacharge output=gwresult_conf dt=8640000000000 type=confined velocity=gwresult_conf
21
22  r.colors map=gwresult_conf color=gyr
23
24  # compute the velocity field
25  r.mapcalc "velocity=(abs(gwresult_conf_x) + abs(gwresult_conf_y))"
26
27  #create the flow direction vectors
28  r.flow elevin=gwresult_conf flout=gwresult_conf_flow skip=15 --o
29  # create contour lines
30  r.contour input=gwresult_conf output=gwresult_conf_contour step=1 --o
31
32  #show the groundwater stream
33  d.mon start=x0
34  d.erase
35  d.rast gwresult_conf
36  d.vect gwresult_conf_flow color=grey
37  d.vect gwresult_conf_contour color=black display=shape,attr attrcol=level lcolor=black lsize=14
38  d.legend -f at=8,12,15,85 map=gwresult_conf
39  d.barscale at=1,10
40  echo "Groundwater flow steady state" | d.text size=6 color=black
41  d.out.png res=1 output=test_gwflow.png
42  convert test_gwflow.png test_gwflow.eps
43  convert test_gwflow.png test_gwflow.pdf
44
45  #show the velocity field
46  d.mon start=x1
47  d.erase
48  d.rast velocity
49  #d.vect gwresult_conf_flow color=grey
50  d.vect gwresult_conf_contour color=black display=shape,attr attrcol=level lcolor=black lsize=14
51  d.legend labelnum=3 at=8,12,15,85 map=velocity
52  d.barscale at=1,10 &
53  echo "Groundwater flow velocity field" | d.text size=6 color=black
54  d.out.png res=1 output=test_gwflow_velocity.png
55  convert test_gwflow_velocity.png test_gwflow_velocity.eps
56  convert test_gwflow_velocity.png test_gwflow_velocity.pdf
57
58  #generate the transport data
59  r.mapcalc "c=if(col() == 15 && row() == 75 , 20.0, 0.0)"
60  r.mapcalc "cs=if(col() == 15 && row() == 75 , 0.0, 0.0)"
61  r.mapcalc "tstatus=if(col() == 1, 2, 1)"
62  r.mapcalc "tstatus=if(col() == 200, 3, tstatus)"
63  r.mapcalc "diff=0.0000000001"
64  r.mapcalc "R=1.0"
65
66  TIME=356
67
68  ##################################################
69  compute_transport()
70  {
71  days=0
72  count=0
73
74  #copy map
75  g.copy rast=c,stresult_conf --o
76  eval `r.univar stresult_conf -g`
77
78  #create a new monitor
79  d.mon start=x2
80  d.mon select=x2
81  d.erase
82  d.rast stresult_conf
83  d.vect gwresult_conf_flow color=grey
84  d.vect gwresult_conf_contour color=black display=shape,attr attrcol=level lcolor=black lsize=14
85  d.legend at=8,12,15,85 map=stresult_conf
86  d.barscale at=1,10
87  echo "Solute transport timestep: 0d" | d.text size=6 color=black
88  echo "Mass balance: $sum" | d.text size=4 color=black at=1,13
89  d.out.png res=1 output=/tmp/test_0000${count}_${AL}_${AT}.png
90
91  #compute the transport
92  while true ; do
93
94    r.solute.transport --o -cs solver=bicgstab top=top_conf bottom=bottom phead=gwresult_conf status=tstatus \
95      hc_x=hydcond hc_y=hydcond R=R cs=cs q=well nf=poros output=stresult_conf dt=86400 diff_x=diff diff_y=diff \
96      c=stresult_conf al=$AL at=$AT
97
98    days=`expr $days + 1`
99    count=`expr $count + 1`
100
101   eval `r.univar stresult_conf -g`
102
103   #show the map
104   d.erase
105   d.rast stresult_conf
106   d.vect gwresult_conf_flow color=grey
107   d.vect gwresult_conf_contour color=black display=shape,attr attrcol=level lcolor=black lsize=14
108   d.legend at=8,12,15,85 map=stresult_conf
109   d.barscale at=1,10
```

```
110     echo "Solute transport timestep: ${days}d" | d.text size=6 color=black
111     echo "Mass balance: $sum" | d.text size=4 color=black at=1,13
112     #safe the display
113     if [ `expr $count \< 10` -eq 1 ] ; then
114       d.out.png res=1 output=/tmp/test_0000${count}_${AL}_${AT}.png
115     elif [ `expr $count \< 100` -eq 1 ] ; then
116       d.out.png res=1 output=/tmp/test_000${count}_${AL}_${AT}.png
117     elif [ `expr $count \< 1000` -eq 1 ] ; then
118       d.out.png res=1 output=/tmp/test_00${count}_${AL}_${AT}.png
119     elif [ `expr $count \< 10000` -eq 1 ] ; then
120       d.out.png res=1 output=/tmp/test_0${count}_${AL}_${AT}.png
121     fi
122
123     #exit after 356 days of calculation
124     if [ `expr $count = ${TIME}` -eq 1 ] ; then
125       break
126     fi
127     done
128   }
129
130   ################################################
131   #               1. computation                 #
132   ################################################
133   AL=0.1
134   AT=0.01
135   compute_transport
```

D.6 Beispiel Script für die zeitliche Kopplung mit variablen Randbedingungen

```
1   #!/bin/sh
2   # set the region accordingly
3   g.region res=1 res3=1 t=10 b=0 n=100 s=0 w=0 e=200
4   # create groundwater flow data
5   r.mapcalc "phead=if(col() == 1 , 50, 45)"
6   #r.mapcalc "phead=if(col() ==200  , 45 + row()/40, phead)"
7   r.mapcalc "status=if(col() == 1 || col() == 200 , 2, 1)"
8   r.mapcalc "well=if((row() == 50 && col() == 175) || (row() == 10 && col() == 135) , -0.005, 0)"
9   r.mapcalc "hydcond=0.00005"
10  r.mapcalc "recharge=0"
11  r.mapcalc "top_conf=20"
12  r.mapcalc "top_unconf=60"
13  r.mapcalc "bottom=0"
14  r.mapcalc "poros=0.17"
15  r.mapcalc "syield=0.1"
16  r.mapcalc "null=0.0"
17
18  #generate the transport data
19  r.mapcalc "c=if(col() == 15 && row() == 75 , 500.0, 0.0)"
20  r.mapcalc "cs=if(col() == 15 && row() == 75 , 0.0, 0.0)"
21  r.mapcalc "tstatus=if(col() == 1 || col() == 200 , 3, 1)"
22  r.mapcalc "diff=0.0000001"
23  r.mapcalc "R=1.0"
24
25  #dispersivity lengths
26  AL="0.1"
27  AT="0.01"
28
29  #First compute a steady state groundwater flow
30  r.gwflow --o -s solver=cg top=top_conf bottom=bottom phead=phead status=status hc_x=hydcond hc_y=hydcond \
31    q=null s=syield r=recharge output=gwresult_conf dt=8640000000000 type=unconfined
32
33  #Second initiate the solute transport
34  r.solute.transport --o -cs solver=bicgstab top=top_conf bottom=bottom phead=gwresult_conf status=tstatus \
35    hc_x=hydcond hc_y=hydcond R=R cs=cs q=well nf=poros output=stresult_conf dt=30 diff_x=diff diff_y=diff \
36    c=c  al=$AL at=$AT
37
38  #create the flow direction vectors
39  r.flow elevin=gwresult_conf flout=gwresult_conf_flow skip=15 --o
40  # create contour lines
41  r.contour input=gwresult_conf output=gwresult_conf_contour step=0.5 --o
42
43  #start a monitor
44  d.mon start=x0
45  d.mon select=x0
46  d.erase
47  d.rast gwresult_conf
48  d.vect gwresult_conf_flow
49  d.vect gwresult_conf_contour color=grey
50  d.legend at=7,11,15,85 map=gwresult_conf
51  d.barscale at=1,10 &
52  echo "Groundwater flow steady state" | d.text size=6 color=black
```

```
53
54  #start a monitor
55  d.mon start=x1
56  d.mon select=x1
57  d.erase
58  d.rast gwresult_conf
59  d.vect gwresult_conf_flow
60  d.vect gwresult_conf_contour color=grey
61  d.legend at=7,11,15,85 map=gwresult_conf
62  d.barscale at=1,10 &
63  echo "Groundwater flow with wells" | d.text size=6 color=black
64
65
66
67  #create a new monitor
68  d.mon start=x2
69  d.mon select=x2
70  d.erase
71  d.rast stresult_conf
72  d.vect gwresult_conf_flow
73  d.vect gwresult_conf_contour color=grey
74  d.legend at=8,12,15,85 map=stresult_conf
75  d.barscale at=1,10
76  echo "Solute transport timestep: 0d" | d.text size=6 color=black
77  d.out.png res=1 output=/tmp/test_0000$count.png
78
79  days=0
80  count=0
81  SINK=well
82
83  #compute in an infinite loop the transport
84  while true ; do
85
86
87  #First compute groundwater flow
88  r.gwflow --o -s solver=cg top=top_conf bottom=bottom phead=gwresult_conf status=status hc_x=hydcond \
89   hc_y=hydcond q=$SINK s=syield r=reacharge output=gwresult_conf dt=86400 type=unconfined
90
91  #create the flow direction vectors
92  r.flow elevin=gwresult_conf flout=gwresult_conf_flow skip=15 --o
93  # create contour lines
94  r.contour input=gwresult_conf output=gwresult_conf_contour step=0.5 --o
95
96  #Second compute soulte transport
97  r.solute.transport --o -cs solver=bicgstab top=top_conf bottom=bottom phead=gwresult_conf status=tstatus \
98    hc_x=hydcond hc_y=hydcond R=R cs=cs q=well nf=poros output=stresult_conf dt=86400 diff_x=diff diff_y=diff \
99    c=stresult_conf al=$AL at=$AT
100
101 days=`expr $days + 1`
102 count=`expr $count + 1`
103
104 if [ `expr $count \< 76` -eq 1 ] || [ `expr $count \> 151` -eq 1 ]; then
105 SINK=well
106 else
107 SINK=null
108 fi
109
110 #select a monitor
111 d.mon select=x1
112 d.erase
113 d.rast gwresult_conf
114 d.vect gwresult_conf_flow
115 d.vect gwresult_conf_contour color=grey
116 d.legend at=7,11,15,85 map=gwresult_conf
117 d.barscale at=1,10 &
118 echo "Groundwater flow timestep: ${days}d" | d.text size=6 color=black
119 if [ `expr $count \< 10` -eq 1 ] ; then
120   d.out.png res=1 output=/tmp/gwflow_0000$count.png
121 elif [ `expr $count \< 100` -eq 1 ] ; then
122   d.out.png res=1 output=/tmp/gwflow_000$count.png
123 elif [ `expr $count \< 1000` -eq 1 ] ; then
124   d.out.png res=1 output=/tmp/gwflow_00$count.png
125 elif [ `expr $count \< 10000` -eq 1 ] ; then
126   d.out.png res=1 output=/tmp/gwflow_0$count.png
127 fi
128
129 eval `r.univar stresult_conf -g`
130
131 #show the map
132 d.mon select=x2
133 d.erase
134 d.rast stresult_conf
135 d.vect gwresult_conf_flow
136 d.vect gwresult_conf_contour color=grey
137 d.legend at=7,11,15,85 map=stresult_conf
138 d.barscale at=1,10
139 echo "Solute transport timestep: ${days}d" | d.text size=6 color=black
140 echo "Quantity: $sum min-max: $min - $max" | d.text size=4 color=black at=1,12
141 if [ `expr $count \< 10` -eq 1 ] ; then
142   d.out.png res=1 output=/tmp/transport_0000$count.png
```

```
143   elif [ `expr $count \< 100` -eq 1 ] ; then
144     d.out.png res=1 output=/tmp/transport_000$count.png
145   elif [ `expr $count \< 1000` -eq 1 ] ; then
146     d.out.png res=1 output=/tmp/transport_00$count.png
147   elif [ `expr $count \< 10000` -eq 1 ] ; then
148     d.out.png res=1 output=/tmp/transport_0$count.png
149   fi
150
151   done
```

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die eingereichte Diplomarbeit selbstständig und ohne fremde Hilfe verfaßt, keine anderen als die von mir angegebenen Quellen und Hilfsmittel benutzt und die den benutzten Werken wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Sören Gebbert
Matrikelnummer 174894