Der Gaußpuls

Eine Anwendung für das Random Walk Verfahren

Anfängerpraktikum „Wissenschaftliches Rechnen“
Universität Heidelberg WS 2011/2012

Dominik Cebulla

24. Februar 2012

Inhaltsverzeichnis

1 Particle Tracking Verfahren **3**
1.1 Einführung und Motivation . . . 3
1.2 Der Random Walk . . . 4

2 Anwendung: Gaußpuls **5**
2.1 Motivation . . . 5
2.2 Der Gaußpuls . . . 5
2.2.1 Grundlagen . . . 5
2.2.2 Visualisierung . . . 5
2.2.3 Fehlermessung . . . 7

3 Implementierung **9**
3.1 Gesamtübersicht . . . 9
3.2 Die GaussianPulse Klasse . . . 10
3.2.1 Initialisierung . . . 10
3.2.2 Der Bewegung des Gaußpulses . . . 11
3.2.3 Fehlermessung (Implementierung) . . . 12
3.3 Ergänzungsmöglichkeiten . . . 14

Literatur **15**

1 Particle Tracking Verfahren

1.1 Einführung und Motivation

Particle Tracking Verfahren befassen sich – wie der Name schon sagt – mit der Bewegung von Partikeln. Anwendungen dafür sind zum Beispiel in der theoretischen Chemie und auch in der Hydrologie bzw. Geologie zu finden.

Als einführendes Beispiel betrachten wir folgendes Szenario: Wie in Abbildung 1 zu sehen besteht der Erdboden aus verschiedenen Schichten (z.B. Sand, Lehm, Erde), ganz unten befindet sich das Grundwasser. Angenommen an der Erdoberfläche befindet sich nutzbarer Boden. Nun kann es sein, dass man darauf Herbizide verteilt um der Unkrautbildung entgegenzuwirken. Diese Herbizide können aber in die Erde eindringen, im schlimmsten Fall ins Grundwasser gelangen und dieses somit verunreinigen. Mit der Zeit verringert sich die Konzentration der Herbizide im Erdboden und die Gefahr einer Grundwasserverschmutzung wird geringer. Man möchte also den Fluss des Herbizids im Erdboden simulieren um eine qualifizierte Aussage über die Verunreinigung treffen zu können. So einen Test in der Realität durchzuführen ist sehr aufwändig und dauert auch eine relativ lange Zeit. Deswegen möchte man das geschilderte Szenario simulieren und verwendet dafür ein Particle Tracking Verfahren. Wie bereits in diesem Szenario zu sehen können sich die Stoffe durchaus im flüssigen Zustand befinden; man „teilt“ die Flüssigkeit einfach in viele kleine Partikel und verwendet dann ein Particle Tracking Verfahren.

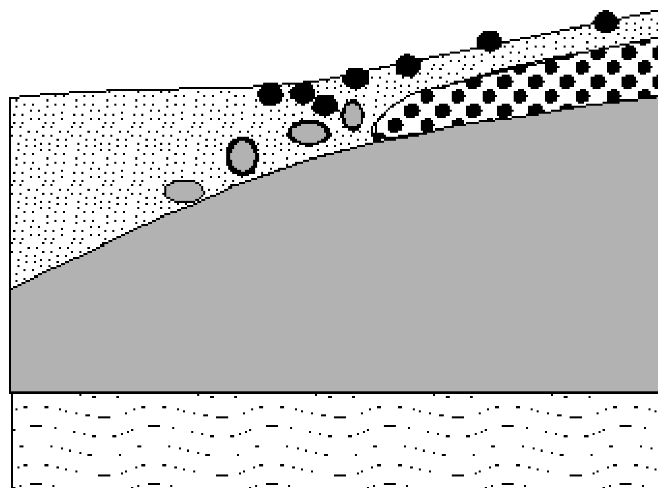


Abbildung 1: Schema eines möglichen Aufbaus von Erdschichten[1]

Wie bereits erwähnt gibt es viele verschiedene Particle Tracking Verfahren. Im nächsten Abschnitt wollen wir nun ein mögliches Verfahren – den Random Walk – vorstellen.

[1] http://www.boden.uni-bonn.de/bilder/schichtenfolge/, 20.02.2012

1.2 Der Random Walk

Ein mögliches Particle Tracking Verfahren ist der sogenannte Random Walk. Im Gegensatz zu vielen anderen Particle Tracking Verfahren verwendet man hier (wie der Name bereits sagt) Zufallskomponenten. Zugunsten einer einfacheren Darstellung werden wir nur die grundlegenden Gleichungen sowie kleine Erklärungen dazu auflisten, ohne die Theorie dahinter genauer zu beleuchten.

Angenommen wir kennen den Ort eines Particles $\mathbf{x}$ zum Zeitpunkt t. Dann kann man durch folgende Gleichung den neuen Ort des Partikels nach einem Zeitschrit Δt bestimmen:

$$\mathbf{x}(t+\Delta t) = \mathbf{x}(t) + \left[\mathbf{u}[x(t)] + \frac{\partial \mathbf{D}[x(t)]}{\partial x}\right]\Delta t + \sqrt{2\mathbf{D}[x(t)]\Delta t}\cdot \mathbf{Z} \tag{1}$$

Es folgt eine kurze Beschreibung über die verschiedenen Komponenten der Gleichung:

- $\mathbf{x}(t)$ der Ort eines Partikels zu einem Zeitpunkt t
- Δt ist die Größe unseres Zeitschritts
- $\mathbf{u}$ ist unser Geschwindigkeitsvektor, welcher die grundlegende Richtung des Partikels beschreibt (in Abhängigkeit vom Ort und der Zeit).
- $\mathbf{D}$ ist der Dispersionstensor (ebenfalls abhängig vom Ort des Partikels und der Zeit).
- $\mathbf{Z}$ ist ein Zufallsvektor mit normalverteilten Komponenten.

Man kann es sich also so vorstellen, dass auf einem Partikel zuerst die grundlegende „Richtungsverschiebung“ (u.a. angegeben durch den Geschwindigkeitsvektor $\mathbf{u}$) angewendet wird und man dann noch an dem Partikel „schaukelt“, d.h. eine (zufällige) Verschiebung, bzw. Streuung einbaut.
Wie oben angemerkt ist dies nur die Präsentation der Lösung ohne genauere Herleitungen angegeben zu haben. Für eine genauere Darstellung verweisen wir auf [Delay2005].

2 Anwendung: Gaußpuls

2.1 Motivation

Durch Gleichung 1 haben wir also ein Grundgerüst erhalten, wie man das Random Walk Verfahren implementieren könnte. Allerdings können wir bis jetzt noch nichts über die Qualität dieses Verfahrens sagen. Was wir brauchen ist ein Szenario, bei dem die Lösung des Verfahrens immer bekannt ist, um somit den Fehler unseres Verfahrens bestimmen zu können. Im folgenden Abschnitt wollen wir so ein Szenario einführen um eine qualifizierte Aussage über die Qualität des Random Walk Verfahrens machen zu können.

2.2 Der Gaußpuls

2.2.1 Grundlagen

In unserem Beispiel rotiert ein Gaußpuls in einem Windungsfeld (wie dieser aussieht werden wir in Abschnitt 2.2.2 sehen). Unser Raum sei $\Omega = (-0.5, 0.5) \times (-0.5, 0.5)$. Die Verteilung der Partikel folgt folgender Konzentrationsgleichung:

$$C(x,y,t) = \frac{2\sigma^2}{2\sigma^2 + 4Dt} \exp\left(-\frac{(\overline{x} - x_0)^2 + (\overline{y} - y_0)^2}{2\sigma^2 + 4Dt}\right) \tag{2}$$

mit $\overline{x} = x\cos(4t) + y\sin(4t)$ und $\overline{y} = -x\sin(4t) + y\cos(4t)$. Unser Windungsfeld entspricht der Funktion $F(x,y) = (-4y, 4x)^T$ (dies entspricht auch gleichzeitig unserem Geschwindigkeitsvektor $\mathbf{u}$ in Gleichung 1). Weiterhin gilt: $2\sigma^2 = 0.004$, $D = 10^{-4}$, $x_0 = -0.25$, sowie $y_0 = 0$. Unser Zeitintervall sei $t \in [0, \pi/4]$. Der Dispersionstensor $\mathbf{D}$ aus Gleichung 1 ist in unserem Fall also ein Skalar ($\Rightarrow \frac{\partial D[x(t)]}{\partial x} \equiv 0$).

Gleichung 2 gibt somit an, wie hoch die Konzentration von Partikeln an einem Ort (x, y) zu einem Zeitpunkt t ist.

2.2.2 Visualisierung

Bevor wir mit der eigentlichen Fehlermessung starten, wollen wir zuerst zeigen, wie sich der Gaußpuls in unserem Raum Ω bewegt.

Abbildung 2 zeigt den Puls zu verschiedenen Zeitpunkten t. Dabei ist die Anzahl der Partikel $T = 16.000$ und wir haben das Zeitintervall in 50 gleich große Zeitschritte zerlegt.

Grundsätzlich erkennt man, dass sich der Gaußpuls (der dadurch gekennzeichnet ist, dass die Konzentration von Partikeln exponentiell abnimmt, je weiter wir uns vom Mittelpunkt des Pulses entfernen) etwas ausbreitet, d.h. dass z.B. zum Zeitpunkt $t = 0$ die Konzentration im Mittelpunkt des Pulses leicht höher ist als zum Zeitpunkt $t = \pi/4$, aber dass auch der „Rand“ des Pulses zum Zeitpunkt $t = \pi/4$ etwas weiter vom Mittelpunkt entfernt ist, als zum Zeitpunkt $t = 0$ (theoretisch ist die Konzentration von Partikeln niemals 0, aber dennoch so klein, dass es unwahrscheinlich ist, dass ein Partikel weit weg vom Mittelpunkt des Pulses entfernt ist).

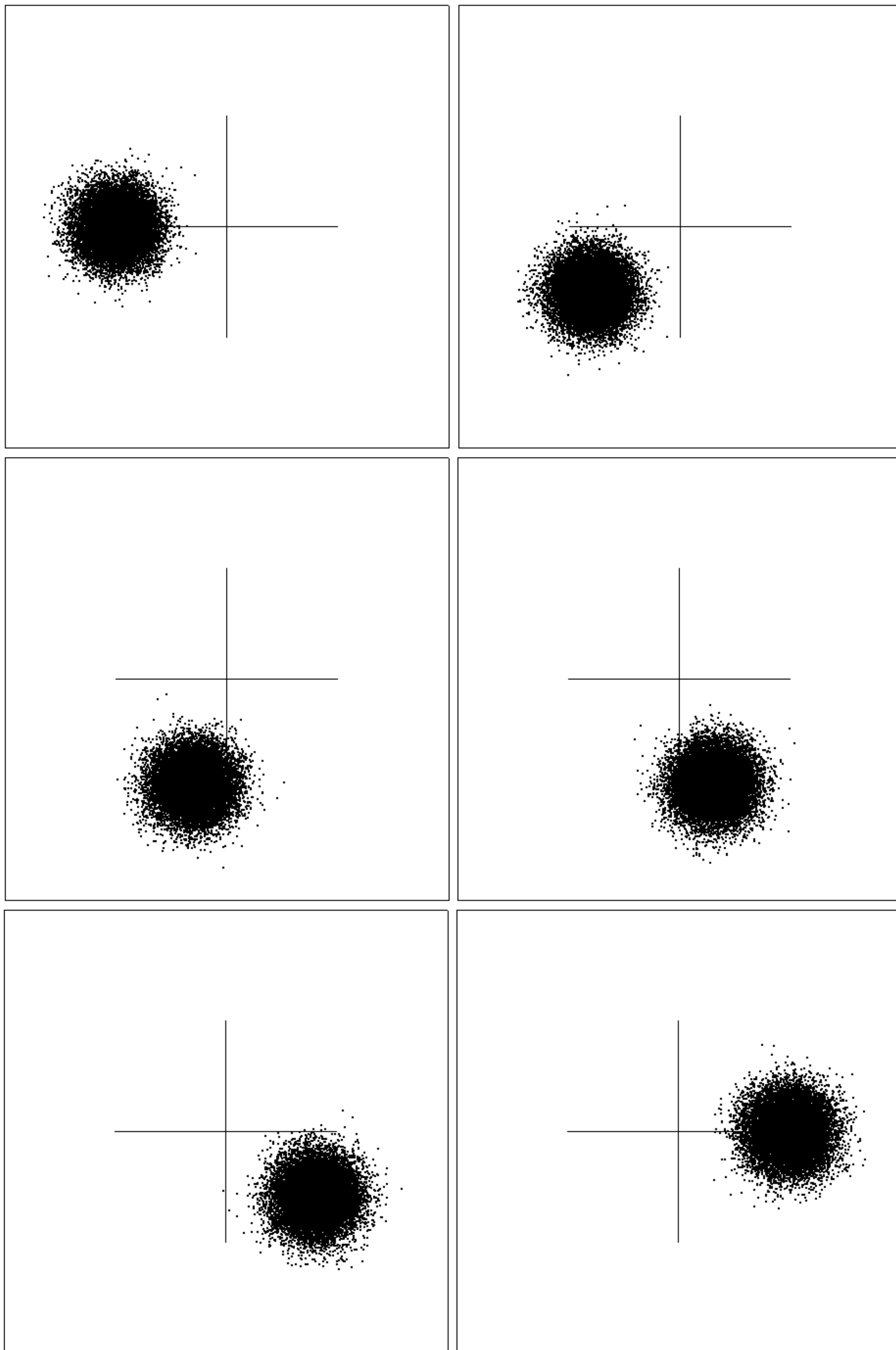


Abbildung 2: Der Gaußpuls zu verschiedenen Zeitpunkten (Leserichtung: links nach rechts; oben nach unten). Pro Frame vergehen 10 Zeitschritte.

2.2.3 Fehlermessung

Wir haben den Gaußpuls eingeführt, um damit den Fehler des Random Walk Verfahrens messen zu können. Um die Masse eines Partikels berechnen zu können benötigen wir die Gesamtmasse der Partikel (über den gesamten Raum); diese berechnen wir numerisch aus Gleichung 2 (die Gesamtmasse ist natürlich unabhängig von der Zeit, solange kein Partikel aus unserem Raum Ω „verschwindet"). Es gilt:

$$M = \int_{\Omega} C(x,y,t)\,dx\,dy = 0.0125664 \qquad \forall\, t \in [0, \pi/4] \tag{3}$$

Den Fehler des Verfahrens berechnen wir nun, indem wir ein Grid G mit m×m Zellen über unseren Raum Ω legen, in jeder einzelnen Zelle die Masse der Partikel berechnen und mit dem Integral über dem jeweiligen Teilraum vergleichen. Unser Programm berechnet den Fehler ξ folgendermaßen:

$$\xi = \left(\sum_{i,j=1}^{m} \left(\int_{\Omega_{ij}} C(x,y,t)\,dx\,dy - M_{ij} \right)^2 \right)^{1/2} \tag{4}$$

mit m wie oben beschrieben, Ω_{ij} bezeichnet den Raum, der durch Zelle G_{ij} beschrieben wird und über den wir integrieren und M_{ij} ist die Masse der Partikel in Zelle G_{ij}. Da wir uns nur für den Start- und Endfehler interessieren ist $t = 0$ oder $t = \pi/4$.

In folgenden Tabellen werden nun einige Messwerte aufgeführt; es gelten folgende Notationen: m ist (wie oben) die Anzahl der Zellen an einer Seite des Grids. TS ist die Anzahl an Zeitschritten (also die Aufteilung des Zeitintervalls $[0, \pi/4]$ in gleich große Teile) und T ist die Anzahl an Partikeln. Mit L_2-Norm bezeichnen wir den gemessenen (End–)Fehler.

Tabelle 1: Einige Messwerte bei denen zusätzlich die Rate bestimmt wurde. Der Random Walk ist ein Verfahren erster Ordnung, jedoch „springt" aus bisher noch ungeklärten Ursachen die Rate um den Wert 1, anstatt von unten gegen den Wert 1 zu konvergieren.

m	TS	T	L_2-Norm	Rate
8	10	256	$8.2928 \cdot 10^{-4}$	
16	20	1.000	$4.3295 \cdot 10^{-4}$	0.94
32	40	4.000	$2.0166 \cdot 10^{-4}$	1.10
64	80	16.000	$8.3180 \cdot 10^{-5}$	1.28
128	160	64.000	$5.1935 \cdot 10^{-5}$	0.68
256	320	256.000	$2.4815 \cdot 10^{-5}$	1.07
512	640	1.000.000	$1.2571 \cdot 10^{-5}$	0.98
1024	1280	4.000.000	$6.2464 \cdot 10^{-6}$	1.01

Tabelle 2: Messwerte der Fehlerberechnung. Zu beachten ist, dass das Verfahren sowohl für m = 128 und TS = 160 als auch für m = 256 und TS = 320 noch nicht konvergiert ist (dies erkennt man daran, dass sich die letzten gemessenen Werte noch relativ stark voneinander unterscheiden).

m	TS	T	L_2-Norm
8	10	256	$8.2928 \cdot 10^{-4}$
8	10	1.000	$7.5625 \cdot 10^{-4}$
8	10	4.000	$6.7631 \cdot 10^{-4}$
8	10	16.000	$7.1384 \cdot 10^{-4}$
8	10	64.000	$7.1966 \cdot 10^{-4}$
8	10	256.000	$7.2720 \cdot 10^{-4}$
8	10	1.000.000	$7.1668 \cdot 10^{-4}$
8	10	4.000.000	$7.2943 \cdot 10^{-4}$
16	20	256	$5.6084 \cdot 10^{-4}$
16	20	1.000	$4.3295 \cdot 10^{-4}$
16	20	4.000	$2.2015 \cdot 10^{-4}$
16	20	16.000	$1.6795 \cdot 10^{-4}$
16	20	64.000	$1.9387 \cdot 10^{-4}$
16	20	256.000	$1.8501 \cdot 10^{-4}$
16	20	1.000.000	$1.7367 \cdot 10^{-4}$
16	20	4.000.000	$1.7561 \cdot 10^{-4}$
32	40	256	$8.7131 \cdot 10^{-4}$
32	40	1.000	$5.0230 \cdot 10^{-4}$
32	40	4.000	$2.0166 \cdot 10^{-4}$
32	40	16.000	$1.0677 \cdot 10^{-4}$
32	40	64.000	$8.3215 \cdot 10^{-5}$
32	40	256.000	$6.0515 \cdot 10^{-5}$
32	40	1.000.000	$6.4160 \cdot 10^{-5}$
32	40	4.000.000	$6.0628 \cdot 10^{-5}$

m	TS	T	L_2-Norm
64	80	256	$7.7833 \cdot 10^{-4}$
64	80	1.000	$4.4302 \cdot 10^{-4}$
64	80	4.000	$1.9016 \cdot 10^{-4}$
64	80	16.000	$8.3180 \cdot 10^{-5}$
64	80	64.000	$5.3395 \cdot 10^{-5}$
64	80	256.000	$3.1005 \cdot 10^{-5}$
64	80	1.000.000	$2.0819 \cdot 10^{-5}$
64	80	4.000.000	$1.8526 \cdot 10^{-5}$
128	160	256	$7.8305 \cdot 10^{-4}$
128	160	1.000	$3.8916 \cdot 10^{-4}$
128	160	4.000	$1.9717 \cdot 10^{-4}$
128	160	16.000	$9.9138 \cdot 10^{-5}$
128	160	64.000	$5.1935 \cdot 10^{-5}$
128	160	256.000	$2.4531 \cdot 10^{-5}$
128	160	1.000.000	$1.3656 \cdot 10^{-5}$
128	160	4.000.000	$7.7382 \cdot 10^{-6}$
256	320	256	$7.6788 \cdot 10^{-4}$
256	320	1.000	$4.0192 \cdot 10^{-4}$
256	320	4.000	$1.9808 \cdot 10^{-4}$
256	320	16.000	$1.0128 \cdot 10^{-4}$
256	320	64.000	$4.9208 \cdot 10^{-5}$
256	320	256.000	$2.4815 \cdot 10^{-5}$
256	320	1.000.000	$1.2835 \cdot 10^{-5}$
256	320	4.000.000	$6.4676 \cdot 10^{-6}$

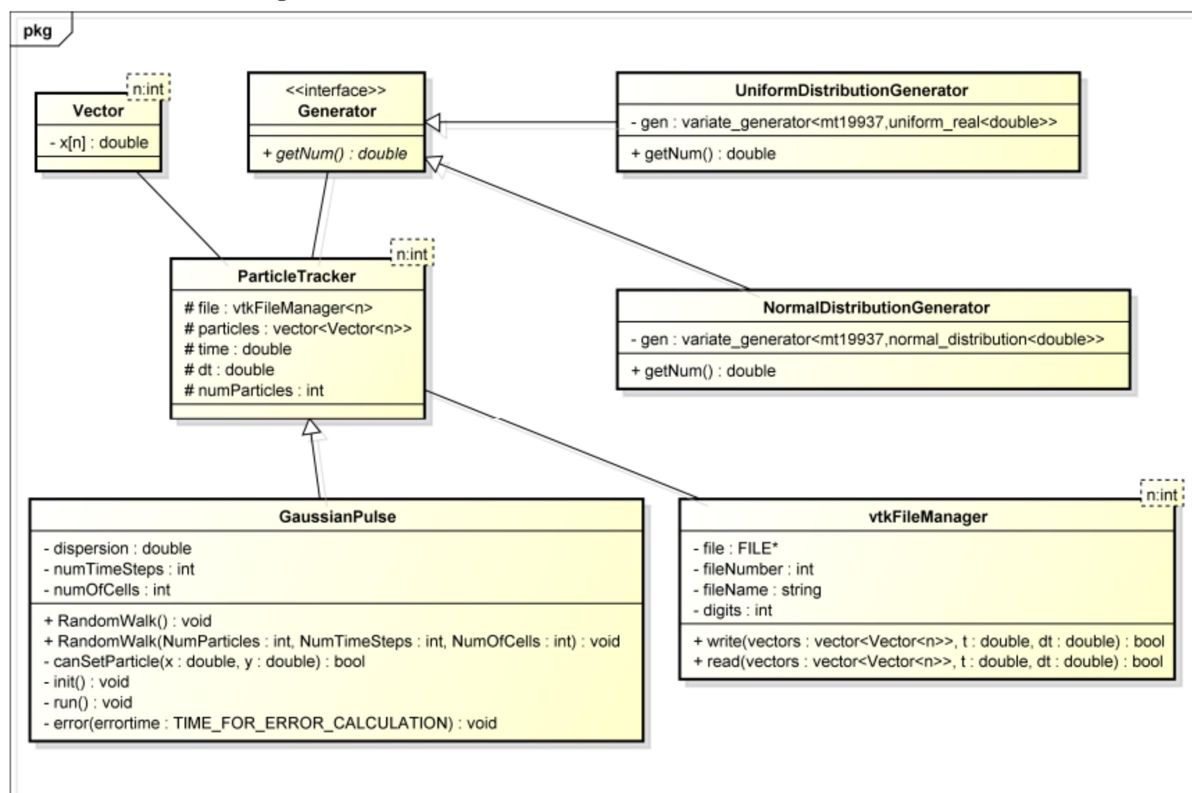
Aufgrund der sehr kleinen Fehler lässt sich abschließend sagen, dass der Random Walk ein gutes Particle Tracking Verfahren ist.

3 Implementierung

3.1 Gesamtübersicht

Das folgende Klassendiagramm (Abbildung 3) zeigt eine Übersicht über die verwendeten Klassen (mit Verebung und Entitäten). Der Übersichtlichkeit halber wurden Konstruktoren, Destruktoren, überladene Operatoren (sofern sie keine besondere Rolle spielen) sowie Get– und Setmethoden weggelassen.

Abbildung 3: Vereinfachte Übersicht über die verwendeten Klassen.



Im folgenden Abschnitt werden wir die GaussianPulse Klasse genauer beleuchten. Da die anderen Klassen größtenteils Hilfsmittel sind wollen wir sie hier an dieser Stelle kurz beschreiben, ohne uns in Details zu verlieren.

Vector: Diese (Template–)Klasse repräsentiert einen Vektor im $\mathbb{R}^n$. Neben den standardmäßigen Operatoren gibt es eine sog. *axpy*–Funktion, die effizient ein Multiply-Add der Form $y = ax + y$ berechnet, wobei x und y Vektoren sind und a ein Skalar ist. Außerdem gibt es eine Funktion, die einen n–dimensionalen Vektor mit zufälligen Komponenten erzeugt.

Generator: Diese Klasse ist eine Schnittstelle für Zufallsgeneratoren, die uns Gleitkommazahlen liefern. Für unser Programm benötigen wir zwei Klassen: den **NormalDistributionGenerator**, welcher normalverteilte Zufallszahlen liefert, sowie den **UniformDistributionGenerator**, welcher gleichverteilte Zufallszahlen liefert.

vtkFileManager: Diese Klasse benötigen wir um nach jedem Zeitschritt die neue Position der Partikel in eine Datei zu speichern. In unserem Fall verwenden wir das VTK File Format (für Details siehe: `http://www.vtk.org/VTK/img/file-formats.pdf`). Der Vorteil ist, dass man dieses Format mit jedem gewöhlichen Texteditor lesen kann (da die Koordinaten der Partikel in lesbarer Textform gespeichert werden) und vor allem, dass wir mit dem open-source Programm ParaView die Bewegung der Partikel visualisieren können. Die read-Methode wurde noch nicht umgesetzt; mit ihrer Hilfe könnte man eine Simulation abbrechen und später wieder vom letzten gespeicherten Punkt weiterlaufen lassen.

ParticleTracker: Die Particle Tracker Klasse vereint alle bis jetzt vorgestellten Klassen und dient als Grundgerüst für „richtige“ Simulationen (d.h. unter realistischen Bedingungen). Unser Gaußpuls ist eher ein Problem theoretischer Natur, um die Qualität des Random Walk Verfahrens zu bestimmen. Diese Klasse stellt bis auf einige Attribute und zugehörigen Get– und Setmethoden bis jetzt keine wirkliche Funktionalität bereit, könnte aber erweitert werden.

Nachdem nun die meisten Klassen vorgestellt wurden, wollen wir uns nun mit der eigentlichen Hauptklasse – der GaussianPulse Klasse – auseinandersetzen.

3.2 Die GaussianPulse Klasse

Die GaussianPulse Klasse muss verschiedene Aufgaben erfüllen: Zuerst müssen die Partikel im Raum Ω so platziert werden, dass sie (annähernd) die Konzentrationsgleichung (Gleichung 2) erfüllen. Dann beginnt das eigentliche Particle Tracking mit dem Random Walk (ein Verfahren 1. und 2. Ordnung werden später vorgestellt). Vor und nach dem Walk müssen auch noch die jeweiligen Fehler gemessen werden. Wie die oben genannten Aufgaben gelöst werden wird Thema folgender Abschnitte sein.

3.2.1 Initialisierung

Zuerst müssen wir es schaffen, dass wir die Partikel so im Raum verteilen, dass die Konzentrationsgleichung näherungsweise erfüllt wird. Dafür ziehen wir mit einem Zufallsgenerator (Gleichverteilung) Zahlen für die x- und y-Koordinaten und setzen diese in die Konzentrationsgleichung ein (t = 0). Sollte der Punkt (repräsentiert durch die gezogenen x- und y-Koordinaten) weit weg vom Maximum der Gleichung sein, so erhalten wir einen sehr kleinen Wert, andernfalls einen relativ großen. Damit die Gleichung erfüllt ist müssen wir also gewährleisten, dass Punkte nah am Maximum sehr wahrscheinlich gesetzt werden und weit entfernte Partikel nur sehr selten gesetzt werden. Dazu ziehen wir noch eine Zufallszahl r (mit einem Gleichverteilungsgenerator) und prüfen, ob r kleiner ist als der Funktionswert von $C(x, y, 0)$. Wenn ja, dann setzen wir ein neues Partikel mit Koordinaten x und y, ansonsten verwerfen wir diese Position. Dies wiederholen wir solange, bis wir die gewünschte Anzahl an Partikeln erreicht haben. Eine mögliche Implementierung in Pseudocode könnte etwa so aussehen:

```
function init()
  while (Anzahl gesetzter Partikel noch nicht erreicht) do
    x = UniformDistributionGenerator.getNum();
    y = UniformDistributionGenerator.getNum();

    if (canSetParticle(x, y) == true)
      Setze (d.h. speichere) neues Partikel mit Koord. (x, y);
      Erhoehe Anzahl gesetzter Partikel um 1;
  end do
endfunction
```

```
function canSetParticle(x, y)
  // Setze x, y in die Gleichung ein
  funcValue = C(x, y, 0);

  // Ziehe Zufallszahl
  r = UniformDistributionGenerator.getNum();

  if (r < funcValue)
    return true;
  else
    return false;
endfunction
```

Die init-Methode soll die gewünschte Anzahl an Partikel setzen und ruft dafür die Hilfsmethode canSetParticle auf, die berechnet, ob ein Partikel gesetzt wird oder nicht. Somit können wir mit diesem „Trick“ Gleichung 2 annähernd erfüllen (Messungen des Startfehlers zeigen, dass dieser tatsächlich sehr klein ist).

3.2.2 Der Bewegung des Gaußpulses

Jetzt ist es an der Zeit die Partikel zu bewegen. Zuerst verwendete unser ein Programm das explizite Euler Verfahren dafür:

```
for (alle Partikel p) do
  p = p + u(p) * dt + sqrt(2 * dispersion * dt) * Z
end do
```

Wie man sieht, ist das quasi eine 1:1 Implementierung von Gleichung 1. Die Werte der benötigten Komponenten (*u, dispersion, Z, ...*) entnimmt man der Beschreibung zu Gleichung 2. Dieses Verfahren ist jedoch recht ungenau. Ein besseres Verfahren ist das sogenannte Heun Verfahren. Dabei ist der Grundgedanke, einen Zeitschritt zu halbieren und das Geschwindigkeitsfeld (bei uns der Vektor u) einmal nach einem halben Schritt und

einmal nach einem ganzen Schritt auszuwerten. Der folgende Pseudocode sollte eventuelle Unklarheiten über die Funktionsweise klären.

```
for (alle Partikel p) do
  // Zuerst legen wir ein temporaeren Partikel an
  temp = p;
  // Der temporaere Partikel wird normal bewegt
  temp = temp + u(temp) * dt;

  // Hier geschieht die eigentlich Bewegung von p in 2 Schritten
  p = p + u(p) * dt/2 + u(temp) * dt/2;
  p = p + sqrt(2 * dispersion * dt) * Z;
end do
```

Man beachte, dass man bei der Bewegung von Partikel p das Geschwindigkeitsfeld einmal auf p und einmal auf das temporäre Partikel anwendet!

Interessant könnte auch noch die Zeit sein, die benötigt wird um ein Partikel p um einen Zeitschritt Δt zu bewegen. Diese betrug auf einem Intel Core 2 Duo P8700 mit jeweils 2.53 GHz Taktung durchschnittlich $2.1 \cdot 10^{-7} s$ (für diese Messung wurde das Programm ausschließlich auf einem Thread ausgeführt; normalerweise verwenden wir zur Bewegung der Partikel mehrere Threads).

3.2.3 Fehlermessung (Implementierung)

Die eigentliche Fehlermessung ist nicht wirklich schwer zu implementieren; der Code dafür ist aber doch relativ ausladend. Deswegen werden wir zuerst beschreiben, mit welchen Mitteln wir den Fehler berechnen und anschließend geben wir eine mögliche Implementierung an.

- Erstelle ein zweidimensionales Array A der Größe m×m ($m \in \mathbb{N}$) und „spanne“ damit unseren Raum Ω auf (die Seitenlänge einer Zelle ist also 1/m [cm]).
- Zähle – ausgehend von der Position der Partikel – die Anzahl der Partikel in jeder einzelnen Zelle A[i][j] und berechne daraus die Masse an Partikeln in jeder Zelle.
- Berechne nun numerisch das Integral über jede Zelle A[i][j]. Verwende dazu die Mittelpunktsregel (d.h. werte die Funktion $C(x, y, t)$ in der Mitte der Zelle aus und multipliziere diesen Wert zweimal mit der Seitenlänge einer Zelle). Um ein genaueres Ergebnis zu erreichen ist es sinnvoll, jede Zelle A[i][j] wieder in mehrere kleinere Zellen aufzuteilen und das Integral über jede kleine Zelle auszuwerten und diese aufeinander zu addieren.
- Bilde nun die Differenz zwischen gemessener Masse und „theoretischer Masse“ (für jede Zelle A[i][j]).
- Addiere nun das Quadrat der Differenz jeder Zelle A[i][j] zum Fehler hinzu und ziehe am Ende der Berechnung (wenn das Verfahren für alle A[i][j] durchgeführt wurde) die Quadratwurzel. Dadurch erhalten wir den Fehler in der gewünschten L_2-Norm.

Den Pseudocode wollen wir nicht vorenthalten, dieser berechnet allerdings nur den Fehler für t = 0. Für t = $\pi/4$ müsste man die Werte bei den Funktionsauswertungen minimal anpassen. Der Code mag zwar etwas kompliziert aussehen, aber im Prinzip macht er nichts anderes, als die oben genannten Stichpunkte abzuarbeiten.

```
function error()
  Mass = 0.0125664;
  error = 0.0;
  step = 1.0/numOfCells; // Laenge einer Zelle
  start = -0.5;
  integralValue = 0.0;
  numOfFineCells = 10; // Feinere Zellen fuer genauere Quadratur

  // Laenge einer feinen Zelle:
  fineCellSize = 1.0/(numOfCells * numOfFineCells);

  // In grid speichern wir die Anzahl an Partikeln in einer Zelle
  grid[numOfCells][numOfCells] = 0;
  // M speichert die Masse der Partikel in einer Zelle
  M[numOfCells][numOfCells] = 0.0;

  // Diese Schleife zaehlt die Partikel in der Zelle
  for (alle Partikel p) do
    i = (numOfCells * p.x + 1) - numOfCells/2;
    j = (numOfCells * p.y + 1) - numOfCells/2;

    grid[i][j] = grid[i][j] + 1;
  end do

  // i und j sind Schleifenzaehler fuer jede Zelle
  for i = 0 .. numOfCells do
    for j = 0 .. numOfCells do
      // Berechne Masse in Zelle grid[i][j]
      M[i][j] = grid[i][j] * Mass/numParticles;

      // k und l sind Zaehler fuer die feinen Zellen
      // in einer grossen Zelle
      for k = 0 .. numOfFineCells do
        for l = 0 .. numOfFineCells do
          // x und y sind die Punkte an denen wir die Funktion
          // zur Berechnung des Integrals auswerten
          x = start + i * step + k * fineCellSize + 0.25;
          y = start + j * step + l * fineCellSize;

          integralValue = integralValue + exp(- (x*x + y*y) /
```

```
                0.004) * fineCellSize * fineCellSize;
        end do
      end do

      M[i][j] = M[i][j] - integralValue;
      error = error + M[i][j] * M[i][j];

      integralValue = 0.0;
    end do
  end do
  return (sqrt(error));
endfunction
```

3.3 Ergänzungsmöglichkeiten

Abschließend wollen wir einige Dinge auflisten, die unserer Meinung nach das Programm benutzerfreundlicher, effizienter und auch funktionaler machen, die jeodch aufgrund fehlender Manpower nicht umgesetzt wurden.

- Der Gaußpuls ist nur ein theoretisches Problem und war vor allem dazu gedacht, die Qualität des Random Walk Verfahrens zu beurteilen. Schön wäre es, wenn man noch einen „richtigen“ Random Walk implementieren würde (also z.B. mit realen Daten und Bedingungen). Dafür wurde die ParticleTracker Klasse erstellt, die man erweitern müsste.

- Die Benutzerfreundlichkeit des Programms lässt noch zu wünschen übrig. Um z.B. die Anfangsparameter (Anzahl von Zeitschritten, Anzahl Partikel, . . .) zu ändern, musste man dies direkt im Code machen und diesen dann neu kompilieren. Um dieses Problem zu beheben könnte man die Parameter als „Konsolenparameter“ übergeben (oder über die Standardeingabe einlesen).

- Ein weiteres Problem ist, dass man auch nicht angeben kann, ob man die Partikel zu jedem Zeitschritt in Dateien schreiben will oder nicht. Es wäre aber deutlich benutzerfreundlicher, wenn man dies zu Programmstart wählen könnte.

Auch wenn es sicherlich noch mehr Punkte zu nennen gibt, so bleibt festzuhalten, dass wir durch unsere Fehlermessungen des Gaußpulses das Random Walk Verfahren als ein gutes Particle Tracking Verfahren identifizieren konnten.

Literatur

[Delay2005] Delay Frédérick, et al.: *Simulating Solute Transport in Porous or Fractured Formations Using Random Walk Particle Tracking: A Review*, in: *Vadose Zone Journal 4: p. 360 - 379*, Madison, WI, 2005

[Kitanidis1994] Kitanidis, Peter K.: *Particle-tracking equations for the solution of the advection-dispersion equation with variable coefficients*, in: *Water Resources Research, Vol. 30, No. 11, p. 3225-3227*, 1994