Institut für Theoretische Physik
der Universität zu Köln

Prof. Dr. Simon Trebst
Michael Becker, Peter Bröcker

Computerphysik

Übungsblatt 6

SS 2013

Website: http://www.thp.uni-koeln.de/trebst/Lectures/2013-CompPhys.html
Abgabedatum: Montag, 3. Juni 2013 vor Beginn der Vorlesung

23. Visualisierung von Vektorfeldern

Programmiertechniken

Die **Visualisierung** von **Kraftfeldern und Strömungen** mittels kleiner Pfeile kennen Sie bereits aus den Vorlesungen der ersten Semester. Bei gegebenen Vektorfeld sind diese auch mit python leicht zu erstellen. Wir beginnen damit, dass wir eine skalare Funktion ϕ diskretisiert auf einem zweidimensional Gitter gegeben haben, wie es zum Beispiel in der Aufgabe 25 der Fall sein wird. Um aus dem **Potential** das Kraftfeld zu berechnen, müssen wir den **Gradienten** berechnen. Dazu benutzen wir die numpy-Funktion *gradient*

```
# define X, Y domain for scalar function phi
values = np.linspace(-.1, .1, 20)
X, Y = plt.meshgrid(values, values)
# discretization step
dh = 0.01
# calculation of gradient
(grady, gradx) = np.gradient(phi, dh)
```

Beachten Sie, dass die Ordnung der Rückgabewerte etwas unintuitiv ist. Bei der Visualisierung des Vektorfeldes der Aufgabe 25 sollten Sie auf dieses Detail deshalb achten.

Um den Gradienten zu visualisieren, verwenden wir die Funktion quiver (dt. Köcher) aus matplotlib. Ihr übergeben wir wie auch schon bei den 3d Plots auf dem 0. Übungsblatt den Definitions- sowie Bildbereich:

```
plt.quiver(X, Y, gradx, grady)
```

Wir wollen nun das elektrische Feld einer elektrischen Ladung im Ursprung zu berechnen. Wir erinnern uns aus den KTP Vorlesungen, dass das Potential proportional zu r^{-1} ist, wobei r der Abstand vom Ursprung sei. Wir wollen also ein entsprechendes Potential *phi* im oben bereits definierten Bereich definieren und es dabei als numpy Array darstellen. Etwa indem wir es über

```
phi = np.empty((len(values), len(values)))
```

initialisieren und dann mit den entsprechenden Werten füllen.

Wer gleich zur Lösung springen möchte, der schaue sich das Beispiel vector_fields.py an.

24. Heißer Draht

5 Punkte

In dieser Aufgabe wollen wir die eine der einfachsten partiellen Differentialgleichungen, die **Wärmeleitungsgleichung** in einer Dimension, untersuchen. Sie haben diese bereits in der Vorlesung als

$$\frac{\partial u}{\partial t} = \kappa \frac{\partial^2 u}{\partial x^2}$$

mit dem Wärmeleitungskoeffizienten κ kennengelernt. Für die Bearbeitung dieser Aufgabe setzen wir $\kappa = 1$. Physikalisch betrachten wir also zum Beispiel die zeitliche Entwicklung der Temperatur eines Drahtes.

Die einfachste Methode, diese Gleichung zu lösen, ist die **Euler-Methode**, die Sie bereits auf den vorherigen Arbeitsblättern kennengelernt haben. Um diese für die Wärmeleitungsgleichung zu implementieren, müssen Sie allerdings sowohl die Zeit- als auch die Ortsableitung diskretisieren. Den zeitlichen Diskretisierungsschritt nennen wir *dt* und den räumlichen *dx*. So ergibt sich also

$$\frac{u(x_j, t_{m+1}) - u(x_j, t_m)}{dt} = \frac{u(x_{j-1}, t_m) - 2u(x_j, t_m) + u(x_{j+1}, t_m)}{dx^2}$$

wobei x_j die Ortskoordinate des Gitterplatzes j und t_m den Zeitwert des Zeitschritts m bezeichnen. Wenn Sie diese Gleichung nach $u(x_j, t_{m+1})$ auflösen erhalten Sie ein Iterationsrezept um für jeden Zeitschritt eine neue Wärmeverteilung zu berechnen.

Für eine funktionierende Simulation fehlen noch Randbedingungen. Wir verwenden **Dirichlet-Bedingungen**, das heißt, wir geben die Konfiguration auf dem Rand vor. Diese sei 0 auf beiden Seiten. Sie müssen außerdem noch eine Startverteilung, d.h. eine Temperaturverteilung bei $t = 0$ angeben. Wir schlagen vor einen Draht der Länge $l = 1$ zu betrachten und die Verteilung

$$u(x, 0) = \sin(\pi x), \quad x \in (0, 1)$$

zu benutzen.

Implementieren Sie den oben beschriebenen Algorithmus. Untersuchen Sie dann, unter welchen Bedingungen für *dt* und *dx* der Algorithmus stabil ist. Am besten können Sie dies etwa mit Hilfe der Animationsfunktion vom vorherigen Übungsblatt tun, indem Sie die Verteilung nach jedem Iterationsschritt neu plotten.

Tipp: Sehr gut visualisieren lässt sich die Verteilung mit der Funktion *imshow*, die wir auf dem 0. Übungsblatt vorgestellt haben. Speichern Sie dazu die maximale Starttemperatur in einer Variable *initial_max*. Die aktuelle Temperaturverteilung nennen Sie z.B. *heat* und stellen diese mit dem Befehl

```
heat_plot = plt.imshow(heat / initial_max, aspect='auto', extent=[-.1, .1, 0, 1])
```

dar. Auch diese Art von Plot können Sie übrigens mit

```
heat_plot.set_data(heat / initial_max)
```

aktualisieren. Um *imshow* zu benutzen, müssen Sie *heat* folgendermaßen initialisieren:

```
heat = np.empty((steps, 1))
```

wobei *steps* die Anzahl der Diskretisierungsschritte ist.

25. Relaxen im Plattenkondensator *5 Punkte*

Den Plattenkondensator und sein **elektrisches Feld** haben Sie schon in der einführenden Vorlesung zur Experimentalphysik kennengelernt. Meist wird dabei die Annahme eines homogenen Feldes innerhalb des Kondensators gemacht, so dass die Feldlinien ein gerades Bündel innerhalb des Kondensators formen entsprechend der folgenden Abbildung.

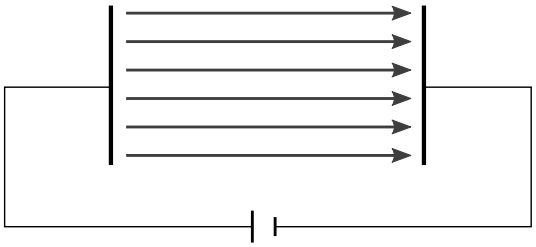

Abbildung 1: Ein Plattenkondensator mit elektrischen Feldlinien

In dieser Aufgabe wollen wir überprüfen, in welchem Maße diese Annahme zutreffend ist. Dazu wollen wir das Feld des Plattenkondensators aus der Lösung der **Laplace-Gleichung**

$$\Delta\phi = 0$$

numerisch exakt berechnen, etwa indem wir auf die in der Vorlesung vorgestellte **Relaxationsmethode** zurückgreifen. Die Randbedingungen seien dabei so formuliert, dass die beiden Kondensatorplatten auf einem Potential von $+1$ bzw. -1 und der Rand der Fläche auf einem Potential von 0 liegt. Um die letzte Randbedingung zu rechtfertigen, müssen die Platten ausreichend weit vom Rand entfernt sein. In der Praxis können Sie die in der folgenden Abbildung dargestellte Konfiguration verwenden.

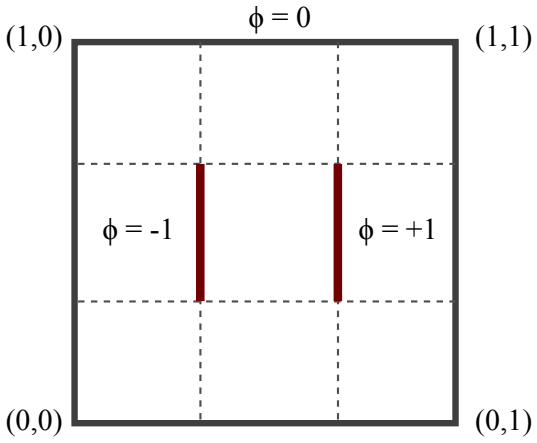

Abbildung 2: Startkonfiguration für die numerische Simulation

Implementieren Sie den in der Vorlesung vorgestellten Algorithmus für die Laplace-Gleichung mit den Randbedingungen aus Abbildung 2. Berechnen Sie dann das resultierende elektrische Feld und stellen Sie dieses dar, etwa wie in Aufgabe 23 beschrieben. Stimmt diese Feldkonfiguration mit der in Abbildung 1 überein? Ordnen Sie die Platten nun näher aneinander an. Wie ändert sich die Feldkonfiguration?

26. Steife Differentialgleichungen *optionale Anwesenheitsaufgabe*

In dieser Aufgabe lernen Sie **steife Differentialgleichungen** als einen neuen Typ von Differentialgleichungen kennen, die oft auftreten, wenn die Parameter der Differentialgleichungen mehrere Größenordnungen abdecken. Die sich ergebenden Anfangswertprobleme kann man numerisch oft nur mit großem Aufwand lösen, wenn man explizite Verfahren, beispielsweise forward-Euler, verwendet – implizite Verfahren wie das backward-Euler sind hingegen erweisen sich als deutlich effizienter. Hier lernen Sie die Hintergründe für die Stabilität impliziter Verfahren kennen und bekommen gleichzeitig einen Einblick in die Numerische Mathematik.

a) Betrachten Sie zunächst das folgende **Anfangswertproblem**:

$$\begin{aligned} u' &= 998u + 1998v, \\ v' &= -999u - 1999v, \end{aligned} \tag{1}$$

mit den Anfangsbedingungen

$$u(0) = 1, \quad v(0) = 0. \tag{2}$$

Lösen Sie dieses Problem mit verschiedenen Verfahren (Euler-Verfahren, Runge-Kutta 4. Ordnung) für $0 \leq t \leq 1$ und vergleichen Sie die Ergebnisse für verschiedene Schrittweiten $h \in \{0.01, 0.001, 0.0001\}$. Was fällt Ihnen auf, wenn Sie noch kleinere Schrittweiten verwenden?

Hinweis: Verwenden Sie die Implementierungen der Verfahren aus Ihren vorigen Übungen, um sich Arbeit zu ersparen! Bei sehr kleinen Schrittweiten h läuft das Programm etwas länger.

b) Vergleichen Sie nun die Numerik mit der **analytischen Lösung**. Substituieren Sie hier hierzu $u = 2y - z$ und $v = -y + z$ – das dann entstehende System von Differentialgleichungen lässt sich dann ohne weitere Rechnung direkt lösen. Plotten Sie nun die numerische Lösungen gegen die wahre Lösung des Anfangswertproblems. Kommentieren Sie die analytische Lösung und beschreiben Sie, wo Sie das Problem vermuten.

c) Steife Differentialgleichungen erfordern generisch sehr kleine **Schrittweiten**, um eine korrekte Lösung zu liefern. Um dies zu verstehen, vereinfachen wir das obige Problem und konzentrieren uns auf folgende Anfangswertproblem:

$$y' = -cy, \quad y(0) = 1, \tag{3}$$

wobei $c > 0$ konstant ist. Die offensichtliche Lösung $y(x) = e^{-cx}$ fällt exponentiell ab, so dass $y(x) \to 0$ für $x \to \infty$.

Das *explizite* Euler-Verfahren (oder forward-Euler-Verfahren), das Sie aus der Vorlesung kennen, nutzt folgende Diskretisierung der Differentialgleichung:

$$y_{n+1} = y_n + hy'_n = (1 - ch)y_n, \quad y_{n=0} = 1. \tag{4}$$

Lösen Sie diese Rekursionsgleichung auf und geben Sie die Lösung für y_n an. Für welche Schrittweiten h ist dieses Verfahren stabil, d. h. liefert die richtige Lösung für große x, und für welche h ist es instabil?

d) Ein Weg, die Instabilität des expliziten Euler-Verfahrens zu beheben, ist die Verwendung eines **impliziten Verfahren**, z. B. das implizite Euler-Verfahren (backward-Euler-Verfahren):

$$y_{n+1} = y_n + hy'_{n+1}, \quad y_{n=0} = 1. \tag{5}$$

Lösen Sie nun auch diese Rekursionsgleichung. Warum ist dieses Verfahren absolut stabil, liefert also unabhängig von der Wahl von h das richtige Ergebnis $y(x) \to 0$ für $x \to \infty$? Erklären Sie mit den gewonnenen Einsichten noch einmal die numerischen Lösungen des Anfangswertproblems (1, 2).

Die Ergebnisse dieses einfachen Anfangswertproblems lassen sich nun verallgemeinern, und damit erhält man rigide Stabilitätskriterien für die verschiedenen Verfahren zur Lösung von gewöhnlichen Differentialgleichungen – dies ist u. a. Gegenstand der Numerischen Mathematik.