Institut für Strömungsmechanik
und Umweltphysik im Bauwesen

Übung 1

Statinäre Transportgleichung - Analytische und numerische Lösung mit Finiten Differenzen in 1D

Ausgabe: 18.04.2019

Abgabe: 16.05.2019 13:00 Uhr

Mit der Abgabe und der Unterschrift auf dieser Abgabe verpflichten Sie sich zur selbständigen Anmeldung im Modul Numerische Strömungsmechanik sobald die Online-Anmeldung freigegeben ist.

Anforderungen

- Arbeit in Gruppen zu 2-3 Personen
- Inhalte der Handreichungen und Übungen sind prüfungsrelevant
- Dieses Dokument ist auszufüllen und unterschrieben einzureichen
- Aufgaben mit * sind optional und führen zu Extra-Punkten in der Bewertung
- Versehen Sie jeden Programm-Code mit Namen und Matrikelnummern
- Alle Plots sind mit aussagekräftiger Achsenbeschriftung und Legende zu versehen
- Nutzen Sie immer die vorgegebenen Variablen- und Funktionsnamen
- Alle Programm-Codes müssen bei Abgabe gesendet werden als E-Mail an saemann@hydromech.uni-hannover.de

Eigenständigkeitserklärung

Hiermit versichere ich, dass die vorliegende Übung selbständig, ohne Hilfe Dritter bearbeitet wurde. Kein Programm-Code wurde aus fremden Übungen kopiert. Mit der Prüfung auf Plagiate erkläre ich mich einverstanden.

Nachname	Vorname	Matrikelnummer	Unterschrift

Übung 1: Stationäre Lösungen der Transportgleichung

1 Analytische Lösung der 1D Transportgleichung

Die 1D-Advektions-Diffusions Gleichung ist

$$\frac{\partial c}{\partial t} = D\frac{\partial^2 c}{\partial x^2} - v\frac{\partial c}{\partial x}. \tag{1}$$

Im stationären Fall lautet die Advektions-Diffusions-Gleichung

$$v\frac{\partial c}{\partial x} - D\frac{\partial^2 c}{\partial x^2} = 0 \tag{2}$$

Hier ist $c = c(x)$ die gesuchte Konzentration eines gelösten Stoffes. Dieser wird mit der Advektionsgeschwindigkeit v und dem Diffusionskoeffizienten D transportiert.

Diese Gleichung soll nun für die Randbedingungen $c(\hat{x} = 0) = 0$ und $c(\hat{x} = 1) = 1$ auf einem Gebiet der Länge L analytisch gelöst werden. Dazu existiert die analytische Lösung

$$c(\hat{x}) = \frac{\exp(Pe \cdot \hat{x}) - 1}{\exp(Pe) - 1}. \tag{3}$$

1. Erstellen Sie das Matlab Script `analytisch.m`, welches die analytische Lösung für die Peclet Zahlen $Pe = (-10, -2, 0, 2, 10)$ plottet.
   - Definieren Sie den Vektor $\hat{x}$ basierend auf der Knotenanzahl n (frei wählbar).
   - Die analytische Lösung `c_ana` soll als anonyme Funktion `f_ana=@(x,Pe)` berechnet werden. (Anonyme Funktionen im Tutorial Kapitel 5.3)
   - Stellen Sie die Lösungen für alle Peclet Zahlen mit Hilfe einer Schleife in einen gemeinsamen Plot dar.
   - Speichern Sie das Bild als Datei `anaLoesung.png`.
2. Was fällt für die Lösung von Pe=0 auf und wie lässt sich dies erklären?

2 Numerische Lösung der stationären Transportgleichung

Die dimensionslose stationäre Advektions-Diffusions-Gleichung

$$Pe\frac{\partial c}{\partial \hat{x}} - \frac{\partial^2 c}{\partial \hat{x}^2} = 0. \tag{4}$$

soll für die Randbedingungen $c(\hat{x}=0)=0$ und $c(\hat{x}=1)=1$ numerisch gelöst werden. Die Näherungslösung soll mit der Finite-Differenzen-Methode berechnet werden.

Wir betrachten die Lösung in Abhängigkeit der Gitter-Peclet-Zahl. Diese ist definiert als

$$Pe_G = \frac{v\Delta x}{D} = \frac{vL}{D}\Delta\hat{x} = Pe\Delta\hat{x}. \tag{5}$$

Die Gitter-Peclet-Zahl vergleicht die advektive ($t_a = \Delta x/v$) und diffusive Zeitskala ($t_d = \Delta x^2/D$) über die Länge einer Gitterzelle.

2.1 Matlabfunktionen für spezielle Matrizen

Im folgenden sollen Funktionen zum Erzeugen von speziellen Matrizen programmiert werden, die in den folgenden Übungen benötigt werden. Verwenden Sie in diesem Abschnitt die folgenden Matlab-Befehle: `zeros`, `ones`, `eye`, `diag`, `disp`, `if`.

Bei Matrizen in Matlab kann man auf die einzelnen Elemente in der Matrix zugreifen, indem man diese über die Zeile und die Spalte adressiert. Beispiel:

```
A=magic(5)        % beliebige Matrix als Beispiel
A(2,3)            % Ausgabe des Wertes aus Zeile 2, Spalte 3
n = 4
A(n, 3)           % Ausgabe des Wertes aus Zeile n (hier 4), Spalte 3
A(2,3) = 5        % Zeile 2, Spalte 3 mit dem Wert 5 überschreiben
A(end, 3) = 7     % letzte Zeile, dritte Spalte mit dem Wert 7 überschreiben
A(:, end) = 1     % alle Zeilen der letzten Spalte mit 1 überschreiben
```

1. Erstellen Sie die Funktion `[A] = tridiag(n, u1, h, o1)`. Diese Funktion erzeugt eine Matrix A der Größe n×n. Auf der Hauptdiagonalen steht der Wert h, auf der unteren ersten Nebendiagionalen der Wert u1 und auf der oberen ersten Nebendiagionalen der Wert o1. Alle restlichen Einträge sind Null. Wenn n kleiner als drei ist, soll eine Fehlermeldung im Kommandofenster ausgegeben werden und A = 0 zurückgegeben werden.

Beispiel:

```
>> A = tridiag(5, 1, 2, 3)
A =
     2     3     0     0     0
     1     2     3     0     0
     0     1     2     3     0
     0     0     1     2     3
     0     0     0     1     2
```

2. Erstellen Sie die Funktion `A = tridiagzyk(n, u1, h, o1)`. Diese Funktion erzeugt eine Matrix A der Größe n×n. Auf der Hauptdiagonalen steht der Wert h, auf der unteren ersten Nebendiagionalen der Wert u1 und auf der oberen ersten Nebendiagionalen der Wert o1. In der rechten oberen Ecke steht der Wert der unteren Nebendiagonale. In der linken unteren Ecke steht der Wert der oberen Nebendiagonalen. Alle restlichen Einträge sind Null. Wenn n kleiner als drei ist, soll eine Fehlermeldung im Kommandofenster ausgegeben werden und A = 0 zurückgegeben werden.

Beispiel:

```
>> A = tridiagzyk(5, 1, 2, 3)
A =
     2     3     0     0     1
     1     2     3     0     0
     0     1     2     3     0
     0     0     1     2     3
     3     0     0     1     2
```

3. Erstellen Sie die Funktion `A = pentadiag(n, u2, u1, h, o1, o2)`. Diese Funktion erzeugt eine Matrix A der Größe n×n. Auf der Hauptdiagonalen steht der Wert h, auf der unteren ersten Nebendiagionalen der Wert u1 und auf der unteren zweiten Nebendiagonalen der Wert u2. Alle restlichen Einträge sind Null. Wenn n kleiner als fünf ist, soll eine Fehlermeldung im Kommandofenster ausgegeben werden und A = 0 zurückgegeben werden.

Beispiel:

```
>> A = pentadiag(7, 1, 2, 3, 4, 5)
A =
     3     4     5     0     0     0     0
     2     3     4     5     0     0     0
     1     2     3     4     5     0     0
     0     1     2     3     4     5     0
     0     0     1     2     3     4     5
     0     0     0     1     2     3     4
     0     0     0     0     1     2     3
```

4. Erstellen Sie die Funktion `A = pentadiagzyk(n, u2, u1, h, o1, o2)`. Diese Funktion erzeugt eine Matrix A der Größe n×n. Auf der Hauptdiagonalen steht der Wert h, auf der unteren ersten Nebendiagionalen der Wert u1 und auf der unteren zweiten Nebendiagonalen der Wert u2. Weiterhin sind Einträge der unteren Nebendiagonalen in der oberen rechten Ecke (Siehe Beispiel) zu finden. Auf der ersten oberen Nebendiagonalen steht der Wert o1 und auf der zweiten oberen Nebendiagonalen der Wert o2. In der unteren linken Ecke (Siehe Beispiel) sind ebenfalls Einträge der oberen Nebendiagonalen zu finden. Alle restlichen Einträge sind Null. Wenn n kleiner als fünf ist, soll eine Fehlermeldung im Kommandofenster ausgegeben werden und A = 0 zurückgegeben werden.

Beispiel:

```
>> A = pentadiagzyk(7, 1, 2, 3, 4, 5)
A =
     3     4     5     0     0     1     2
     2     3     4     5     0     0     1
     1     2     3     4     5     0     0
     0     1     2     3     4     5     0
     0     0     1     2     3     4     5
     5     0     0     1     2     3     4
     4     5     0     0     1     2     3
```

5*. Die vorhergehenden Beispiele erzeugen vollbesetzte Matrizen. Das bedeutet, dass auch die Nullen gespeichert werden. Da für große dünnbesetze Matrizen viele überflüssige Nullen gespeichert werden müssen, gibt es hier als Alternative das Sparse-Format, um solche Matrizen platzsparend zu speichern. Hier werden nur Matrixeinträge gespeichert, die sich von Null unterscheiden. Der Vorteil ist, dass große Matrizen im Sparse-Format weniger Speicher benötigen und das Matrix-Vektor-Multiplikationen schneller berechnet werden können.

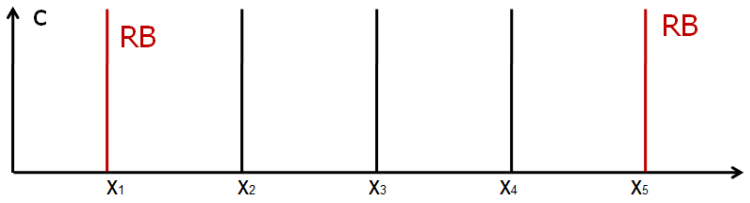
Erstellen Sie eine Sparse-Versionen der Funktion `pentadiag`. Sie können dazu die Funktionen `sparse`, `speye` und `spdiags` verwenden. Nennen Sie die Funktion `sppentadiag`. Eine Sparse-

Matrix kann mit dem Befehl `full` als vollbesetze Matrix angezeigt werden. Hinweis: Erzeugen Sie die Matrizen direkt als Sparse-Matrix und nicht über eine vollbesetzte Matrix!

6*. Untersuchen Sie, wie groß die Matrizen, die Sie mit den Funktionen `pentadiag` und `sppentadiag` erzeugen können, maximal sein dürfen, bevor Matlab anzeigt, dass nicht ausreichend Speicher vorhanden ist. Achtung: Bei diesem Versuch kann der Computer abstürzen.

2.2 Diskretisierung mit zentralen Differenzen

Das Gebiet wird mit fünf äquidistanten Knoten dargestellt. Davon sind die zwei Knoten $x_1 = 0$ und $x_5 = 1$ die Randknoten.



1. An welchen Knoten sind Werte bekannt? An welchen Knoten sind die Werte unbekannt? Wieviele unbekannte Knoten gibt es?

2. Schreiben Sie für jeden Knoten die diskretisierte Gleichung auf. Nähern Sie dazu die erste und zweite Ableitung in Gleichung (4) mit zentralen Differenzen.

3. Formen Sie die Gleichung so um, dass die folgende Matrix-Vektor-Gleichung entsteht.

$$\{Pe_G \mathbf{P}_c - \mathbf{K}\}\,\vec{c} = r\vec{h}s \tag{6}$$

Hier enthält $\mathbf{P}_c \in \mathbb{R}^{n\times n}$ die Diskretisierung des Advektionsterms mit zentralen Differenzen. $\mathbf{K} \in \mathbb{R}^{n\times n}$ enthält die Diskretisierung des Diffusionsterms und $r\vec{h}s \in \mathbb{R}^{n\times 1}$ die Randbedingungen. n ist die Anzahl aller Knoten. Der Vektor $\vec{c}$ ist definiert als

$$\vec{c} = \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \end{pmatrix} \tag{7}$$

und der Rechte-Seite-Vektor $\vec{rhs}$ enthält in diesem Fall nur die Werte der Dirichlet-Ränder $\vec{c_d}$

$$\vec{rhs} = \vec{c_d} = \begin{pmatrix} c_L \\ 0 \\ 0 \\ 0 \\ c_R \end{pmatrix} \tag{8}$$

Definieren Sie die Matrizen $\mathbf{P}_c$ und $\mathbf{K}$.

$$\mathbf{P}_c = \begin{bmatrix} & & & & \\ & & & & \\ & & & & \\ & & & & \\ & & & & \end{bmatrix} \tag{9}$$

$$\mathbf{K} = \begin{bmatrix} & & & & \\ & & & & \\ & & & & \\ & & & & \\ & & & & \end{bmatrix} \tag{10}$$

4. Das Gleichungssystem (6) kann jetzt in der Form

$$\mathbf{A}\overrightarrow{c} = \overrightarrow{rhs} \tag{11}$$

dargestellt werden. Definieren Sie die Matrix $\mathbf{A}$

$$\mathbf{A^{n\times n}} = \begin{bmatrix} & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \\ & & \end{bmatrix} \tag{12}$$

5. Welche Einträge müssen in der ersten und letzten Zeile von $\mathbf{A}$ stehen, damit die Randbedingungen richtig einbezogen werden? Berücksichtigen Sie dabei die Regeln der Matrix-Vektor-Multiplikation, also z.B.

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} a_{11}\cdot b_1 + a_{12}\cdot b_2 + a_{13}\cdot b_3 \\ a_{21}\cdot b_1 + a_{22}\cdot b_2 + a_{23}\cdot b_3 \end{pmatrix}$$

2.3 Implementierung der numerischen Lösung in Matlab

Diese $n \times n$ Matrix und der Rechte-Seite-Vektor reichen aus, um das Problem der Konzentrationsverteilung auf allen Knoten zu lösen.

1. Erstellen Sie die Funktion `[c x PeG] = ADEstationaer(n, Pe, cL, cR)`, welche die Näherungslösung c für das hier in Abschnitt 2 beschriebene Problem mit dem zentralen Diskretisierungsschema berechnet.

Eingabeparameter:	n	Anzahl Knoten
	Pe	Peclet Zahl Pe
	cL	Randbedingung links
	cR	Randbedingung rechts

Rückgabeparameter	c	Numerische Lösung $\vec{c}$, beinhaltet c_L und c_R
	x	Vektor mit n Knotenkoordinaten
	PeG	Gitter Peclet Zahl als Skalar

   Innerhalb der Funktion sieht der Ablauf folgendermaßen aus:

   (a) Gittergenerierung und Berechnung benötigter Parameter (nur aus den Eingabeparametern)

   (b) Erstellung des Gleichungssystems $\mathbf{A}$ und $\overrightarrow{rhs}$ (Tipp: Nutzen Sie die `tridiag` Funktion zur Generierung der Matrix)

   (c) Lösung des Gleichungssystems (11)

2. Kopieren Sie das Script `analytisch.m` als `stationaer.m` und erweitern Sie es. Das Script soll die numerische zusammen mit der analytischen Lösung in einem Plot darstellen. Plotten Sie wieder die Liste an Peclet Zahlen Pe=(-10,-2,0,2,10) für die Anzahl von n=11 Knoten.

   - Nutzen Sie für jede Peclet Zahl einen eigenen Subplot (Matlab Kommando: `subplot`).
   - Speichern Sie die Ausgabe als Bild `numLoesung.png`.

   Das Skript sollte jetzt die berechnete Näherungslösung zusammen mit der analytischen Lösung für eine Liste von Peclet-Zahlen graphisch ausgeben.

2.4 Ein allgemeines Diskretisierungsschema

Ein allgemeines Schema zur Diskretisierung der stationären Advektions-Gleichung (4) lautet

$$Pe_G \left[\alpha \left(c_i - c_{i-1}\right) + \left(1 - \alpha\right)\left(c_{i+1} - c_i\right)\right] - \left(c_{i-1} - 2c_i + c_{i+1}\right) = 0 \tag{13}$$

Mit der Wahl von α wird das Differenzenschema für den Advektionsterm bestimmt:

$\alpha = 0$	vorwärts
$\alpha = 0.5$	zentral
$\alpha = 1$	rückwärts

Mit der im vorhergehenden Abschnitt eingeführten Matrix-Vektor-Schreibweise kann das Gleichungssystem umformuliert werden:

$$\mathbf{A}\overrightarrow{c} = \overrightarrow{rhs} \tag{14}$$

mit

$$\mathbf{A} = Pe_G \left[\alpha \mathbf{P}_b + (1-\alpha)\,\mathbf{P}_f\right] - \mathbf{K} \tag{15}$$

und

$$\overrightarrow{rhs} = \vec{c}_d \tag{16}$$

Hier enthält $\mathbf{P}_b$ die Diskretisierung des Advektionsterms mit Rückwärtsdifferenzen. $\mathbf{P}_f$ enthält die Diskretisierung mit Vorwärtsdifferenzen.

1. Definieren Sie die Matrizen $\mathbf{P}_b$ und $\mathbf{P}_f$.

$$\mathbf{P}_b = \begin{bmatrix} & & \\ & & \\ & & \\ & & \\ & & \end{bmatrix}$$

$$\mathbf{P}_f = \begin{bmatrix} & & \\ & & \\ & & \\ & & \\ & & \end{bmatrix}$$

2. Kopieren Sie die Funktion `ADEstationaer` als `ADEstationaer2`, und zwar so, dass α als zusätzlicher Eingabeparameter genutzt wird. Dieses Script soll die Matrix $\mathbf{A}$ mit dem allgemeinen Diskretisierungsschema (Gleichung 15) statt mit zentralen Differenzen berechnen.

3. Erstellen Sie das Script `stationaer2.m`,welches die analytische Lösung sowie die numerische Lösung für die Parameterliste $\alpha = [0,\ 0.5,\ 1]$ plottet.

   - Realisieren Sie dies mit einer Schleife über die Parameterliste α.
   - Fügen Sie eine Legende ('analytisch', 'vorwärts', 'rückwärts', 'zentral') hinzu.
   - Verwenden Sie für jede Peclet-Zahl einen eigenen Sub-Plot mit eigenem `title` der Peclet Zahl und der gitter-Peclet Zahl. Hilfreich: `subplot` oder `hold on`.
   - Speichern Sie die Plots als Bild `allgemeinSchema.png` ab.

   Tipp: Mit `num2str` lassen sichZahlen in Zeichenketten umwandeln. Zeichenketten wie Vektoren mit den eckigen Klammern `[]` zusammensetzen. Beispiel: `title(['Pe_G = ' num2str(PeG)])`.

2.5 Stabilitätsanalyse

1. Stellen Sie jetzt im Skript `stationaer2.m` 11 Gitterknoten ein. Untersuchen Sie die Lösungen für die Gitter-Peclet-Zahlen 1.0, 1.5, 2.0 und 3.0. Welche Lösung erscheint Ihnen für welche Gitter-Peclet Zahl am besten? Ist die Vorwärtsdifferenz irgendwann gut?

P_{eG}	$\alpha = 0$	$\alpha = 0.5$	$\alpha = 1$
1.0			
1.5			
2.0			
3.0			

2. Gibt es ein Schema, das die analytische Lösung trifft? Welches Schema überschätzt und welches Schema unterschätzt den advektiven Transport? Können Sie sich vorstellen, warum das so ist?

P_{eG}	$\alpha = 0$	$\alpha = 0.5$	$\alpha = 1$
1.0			
1.5			
2.0			
3.0			

3. * Überlegen Sie sich eine Lösung, die eine bessere Näherung als die zentrale Differenz und die Rückwärtsdifferenz ergibt. Definieren Sie dazu den Parameter α in Abhängigkeit der Gitter-Peclet Zahl so, dass sich für eine kleine Gitter-Peclet Zahl (stabil mit zentraler Differenz) der Parameter $\alpha = 0.5$ ergibt (zentrale Differenz) und für große Gitter-Peclet Zahlen (instabil mit zentraler Differenz) der Parameter $\alpha = 1$ ergibt. Berechnen Sie die entsprechende Lösung ebenfalls in Ihrem Matlab-Skript und geben Sie die Lösung mit im Plot aus. Die Funktion $\alpha(Pe_G)$ soll kontinuierlich sein. Sprünge in der Funktion sind als Lösung nicht erlaubt.

2.6 Konvergenzanalyse

1. Vergleichen Sie die analytische und die numerische Lösung. Berechnen Sie die Abweichung e_1 als geometrischen Abstand zwischen der numerischen und der analytischen Lösung an den Punkten x_i:

$$e_1 = \sqrt{\sum_{i=1}^{3} \left(c_{ana}(x_i) - c_{num}|_{x_i}\right)^2}, \quad x_i = (0.25,\, 0.5,\, 0.75) \tag{17}$$

   - Nutzen Sie eine Peclet Zahl von $Pe = 1$.
   - Nutzen Sie vorwärts, rückwärts und zentrale Differenzen.
   - Untersuchen Sie die Abweichung e_1 in Abhängigkeit der Gitterweite Δx.
   - Nutzen Sie `norm` um e_1 zu bestimmen.
   - Wählen Sie dazu die Gitterweite jeweils so, dass die Gitterknoten exakt auf den Punkten x_i liegen. Beginnen Sie mit einer Gitterweite von $\Delta x = 0.25$ und halbieren Sie die Gitterweite aus der vorhergehenden Rechnung, um die neue Gitterweite zu erhalten.
   - Plotten Sie die Abweichung e_1 in Abhängigkeit der Gitterweite Δx für vier Werte in ein Diagramm.
   - Verwenden Sie für das Diagramm `plot`, `semilogx`, `semilogy` oder `loglog`.
   - Speichern Sie die Grafik als `konvergenz.png`.

2. Welche Gitterweite würden Sie empfehlen zu nutzen? Begründen Sie die Entscheidung.

3. In der Realität ist die analytische Lösung meist nicht bekannt. Wie können Sie dennoch eine gute Lösung bestimmen?

4. * Setzen Sie Ihre Antwort aus der vorherigen Frage um. Erstellen Sie dazu das Script `konvergenz2`. Berechnen Sie die Abweichung e_2 an den gleichen Knoten wie zuvor. Stellen Sie die Abweichung über die Gitterweite dar. Speichern Sie die Ausgabe als `konvergenz2.png`.

2.7 Taylor-Entwicklung und Finite Differenzen

1. Leiten Sie das zentrale finite Differenzen-Schema für die erste Ortsableitung $\partial f/\partial x$ und ein beliebiges finite Differenzen-Schema für die zweite Ortsableitung $\partial^2 f/\partial x^2$ mit der Taylor-Entwicklung her.

2. Welche Abbruchfehler haben die von Ihnen hergeleiteten Schemata? Unterstreichen Sie den entsprechenden Term in Ihrer Taylorentwicklung.