

CFD-Programmier-Seminar

Projekt 1: Riemannlöser

Aufgabenstellung

In diesem Projekt soll Erfahrung mit dem Herzstück eines Finite-Volumen Codes, dem Riemannlöser, gewonnen werden. Ein fertiges CFD-Codegerüst wird zur Verfügung gestellt, in das alle Riemannlöser hineinprogrammiert werden können. Fertig programmierte Versionen liegen als Bibliotheken bereit.

Zum Verständnis der Grundidee der Flussberechnung in einem FV-Code sollen verschiedene Flussfunktionen programmiert werden und mit diesen dann an Vergleichsrechnungen durchgeführt werden. Als Testprobleme dienen hierbei verschiedene 1D-Stossrohrprobleme aus der Literatur.

Insbesondere sind hier Genauigkeit, Rechenzeit und eventuell auftretende numerische Probleme von Interesse.

Folgende Aufgaben sind zu lösen:

- Programmierung verschiedener Flussfunktionen
- Validierung der selbst programmierten Flussfunktionen anhand eines Testbeispiels
- Vergleich der Flussfunktionen (genaue Aufgabenstellung siehe unten)

Lehrziele

- Verständnis der Rolle der Flussfunktionen in einem FV-Code
- Erlernen der Grundeigenschaften und Unterscheidungsmerkmale verschiedener klassischer Flussfunktionen in der Anwendung
- Interpretation und Bewertung von Ergebnissen
- Einführung in den FV-Code *cfdfv*
- Berechnung einfacher Testbeispiele
- Visualisierung eindimensionaler Daten mit Visit

Benötigte Software

- Linux
- Fortran Compiler
- beliebiger Texteditor unter Linux (z. B. *gedit*)
- Visit

Benötigte thermodynamische Beziehungen

Energie:

$$E = \rho\epsilon + \frac{1}{2}\rho\bar{v}^2$$

mit

$$\bar{v} = \sqrt{v_1^2 + v_2^2}$$

Zustandsgleichung für ein ideales Gas in kalorischer Form (ϵ = spez. innere Energie):

$$\epsilon = \epsilon(\rho, p) = \frac{p}{(\gamma - 1)\rho}$$

Schallgeschwindigkeit:

$$c = \sqrt{\frac{\gamma p}{\rho}}$$

Enthalpie:

$$H = \frac{E + p}{\rho}$$

1 Aufgabenteil 1: Programmierung der Flussberechnung mit dem exakten Riemannlöser von Godunov

1.1 Hintergrund

Grundidee eines Finite-Volumen-Codes ist es, das Rechengebiet in Gitterzellen zu unterteilen, in denen jeweils die integralen Mittelwerte der physikalischen Erhaltungsgrößen Masse, Impuls und Energie gespeichert werden. Die Eulergleichungen in ihrer Flussformulierung geben uns an, wie sich diese Mittelwerte durch Ein- und Ausfließen über den Rand ändern:

$$u_t + f_1(u)_x = 0$$

mit

$$u = \begin{pmatrix} \rho \\ \rho v_1 \\ \rho v_2 \\ E \end{pmatrix}, \quad f_1 = \begin{pmatrix} \rho v_1 \\ \rho v_1^2 + p \\ \rho v_1 v_2 \\ v_1(E + p) \end{pmatrix}$$

Die Bedeutung des Riemannlösers wird in den folgenden Bildern verdeutlicht. Betrachten wir zunächst die allgemeine Ausgangssituation, nämlich zwei benachbarte Zellen mit unterschiedlichem Zustand, über deren Grenze wir den Fluss berechnen wollen:

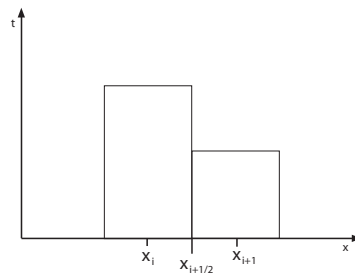


Abbildung 1: Anfangsbedingung

Wie wir es bereits kennen gelernt haben, liegt hier eine Unstetigkeit vor, ein lokales Riemannproblem, dessen Lösung in unserem Fall folgendermaßen aussehen wird:

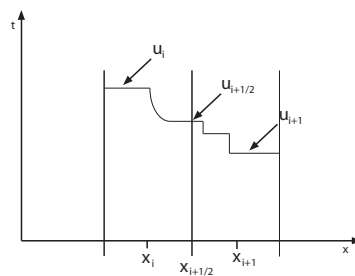


Abbildung 2: Dichte zum Zeitpunkt $t + \Delta t$

Das obige Bild stellt die Lösung des Riemannproblems nach einer Zeit Δt dar. Links und rechts sind die alten Zustände u_i bzw. u_{i+1} markiert. Es ist gut sichtbar, wie sie links von der Verdünnung und rechts vom Stoß begrenzt werden. Für die Flussberechnung im FV-Verfahren ist allerdings die Struktur, wie sie sich hier darstellt egal. Von zentraler Bedeutung ist einzig und alleine der Zustand auf der Zellgrenze, nämlich $u_{i+1/2}$. Dies ist nämlich genau der Zustand, der sich sofort an der Zellgrenze einstellen wird und auch während des gesamten Zeitschritts konstant bleibt. Kennen wir diesen Zustand, können wir den exakten Fluss auf dieser Zellgrenze angeben, nämlich

$$g_{i+1/2}^{god} = f_1(u_{i+1/2}) = f_1(u_{RP}(0; u_{ij}, u_{i+1,j}))$$

Der exakte Riemannlöser liefert uns aber genau diesen Zustand für gegebene Anfangsbedingungen von Dichte, Geschwindigkeit und Druck, womit wir nun, vorausgesetzt, daß wir die Lösung des Riemannproblems bestimmen können. Letzteres nehmen wir als gegeben an. Abgesehen vom immensen Aufwand, die exakte Lösung des Riemannproblems zu bestimmen, liefert uns diese Herangehensweise eine sehr einfache Methode zur Flussberechnung.

Eine Schwierigkeit stellt sich in unserem Fall aber noch, nämlich die dritte Gleichung der Eulergleichungen, wie oben angegeben. Der Riemannlöser kann nur 1D-Probleme lösen, eine analytische Lösung von 2D-Problemen ist nicht möglich. Wir müssen uns also noch Gedanken darüber machen, wie mit der Lösung dieser Gleichung zu verfahren ist. Die Lösung gestaltet sich jedoch denkbar einfach, denn die y-Geschwindigkeit v_2 können wir einfach als konstant während des Zeitschritts annehmen, wir können sie also als eine sog. *passive skalare Größe* behandeln, die einfach nur mit der Strömung transportiert wird. Woher bekommen wir aber nun den benötigten Wert für die Flussberechnung her? Wir haben v_{2l} und v_{2r} gegeben. Die Information, welchen Wert wir verwenden müssen erhalten wir aus der Riemannlösung: Ist die Geschwindigkeit an der Kante positiv, so bewegt sich die Strömung nach rechts und wir müssen v_{2l} verwenden, ist sie negativ, bewegt sich die Strömung nach links und an der Zellkante wird $v_{2_{i+1/2}}$ gleich v_{2r} sein.

1.2 Genaue Aufgabenstellung

- Öffnen Sie eine Shell und kopieren Sie die Datei *Flux_godunov.f90* aus dem Verzeichnis *LeereFiles* in das Hauptverzeichnis und öffnen Sie die Datei im Texteditor:

```
cd ~ /CFDFV
cp . /LeereFiles/Flux_godunov.f90 .
gedit Flux_godunov.f90 &
```

- Editieren Sie *Flux_godunov.f90* und programmieren Sie die Routine so, daß sie unter Verwendung des exakten Riemannlösers g^{god} berechnet.
- Wenn Sie Ihre Änderungen beendet haben, speichern Sie die Datei ab und verlassen das Editorfenster, indem Sie auf die Shell klicken.

- Kompilieren Sie das Programm mit folgender Eingabe:

```
make
```

Sollten Sie Fehlermeldungen vom Compiler erhalten, so korrigieren Sie diese im Editorfenster, speichern die korrigierte Datei und wiederholen das Kompilieren.

- Sie erhalten so ein nun ausführbares Programm namens *cfdfv*

2 Aufgabenteil 2: Validierung der selbst programmierten Flussberechnung

2.1 Hintergrund

Die selbst programmierte Flussberechnung soll nun anhand eines klassischen Testbeispiels, dem sog. *SOD-Testfall* validiert werden. Dieser Testfall stellt ein klassisches Stoßrohrproblem dar, bei dem ein Stoßrohr durch eine undurchlässige Membran in zwei Teile geteilt ist. Die linke Kammer wird aufgepumpt, so dass ein Überdruck entsteht, während die rechte Kammer evakuiert wird, um dort einen starken Unterdruck gegenüber der linken Kammer zu erzeugen:

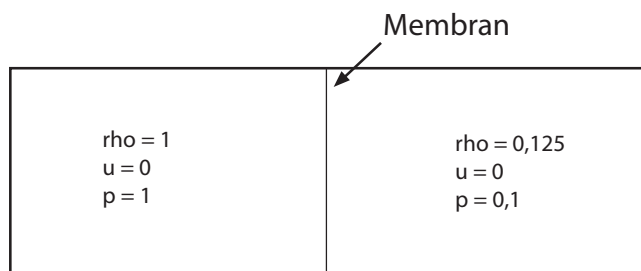


Abbildung 3: Anfangsbedingungen des SOD-Testfalls

Anmerkung: Die Größen sind zur Vermeidung von Rundungs- und Auslöschungsproblemen auf 1 normiert. Die Geschwindigkeit in *y*-Richtung ist hier natürlich 0, da wir ein reines 1D-Problem betrachten.

Zum Zeitpunkt $t = 0$ wird die Membran durchstoßen. Die sich einstellende Lösung ist in Abbildung 4 schematisch dargestellt.

Bei der Dichte ρ sehen wir alle entstehenden Phänomene am Besten: Rechts läuft ein Stoß in das noch ruhende Medium hinein, in der Mitte folgt eine Kontaktunstetigkeit dem Stoß und nach links breitet sich ein Verdünnungsfächer aus. Vergleicht man den Dichteverlauf mit dem Druckverlauf, macht man eine interessante Feststellung: Der Stoß und die Verdünnung sind dort genauso zu sehen wie auch bei der Dichte, die Kontaktunstetigkeit ist jedoch verschwunden. Dies liegt daran, daß über eine Kontaktunstetigkeit die Bedingung der Druckgleichheit gilt. Bei der Geschwindigkeit haben wir ein ähnliches Verhalten - beim Stoß kann eine sprunghafte Änderung der Geschwindigkeit beobachtet werden, am Verdünnungsfächer ist eine kontinuierliche

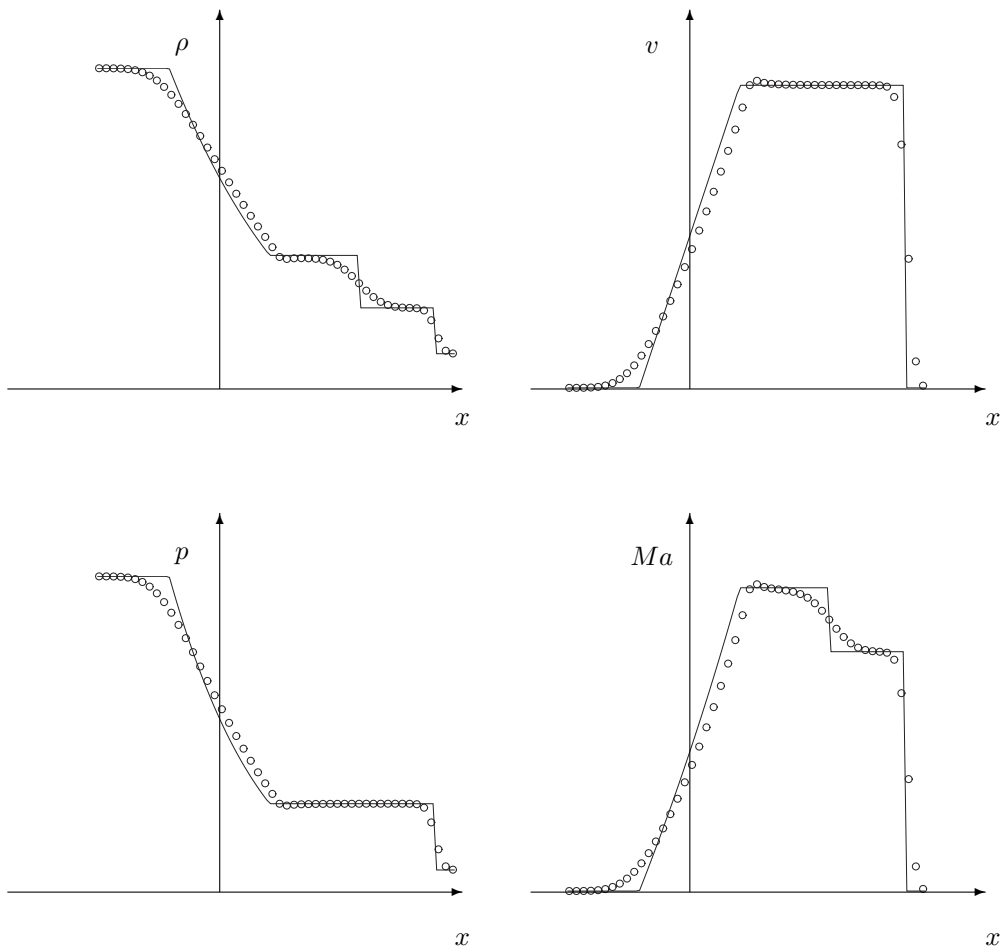


Abbildung 4: Exakte (durchgezogen) und numerische Lösung (Punkte) des SOD-Testfalls

Änderung zu sehen und die Kontaktunstetigkeit ist wieder nicht erkennbar. Dies liegt an der zweiten Bedingung über die Kontaktunstetigkeit, nämlich der, dass die Normalgeschwindigkeiten gleich sein müssen.

2.2 Genaue Aufgabenstellung

- Öffnen Sie eine Shell, falls Sie noch keine offen haben, gehen Sie ins Projektverzeichnis und wechseln Sie in das Verzeichnis *Calc/RiemannProblems*::

```
cd ~ /CFDFV
cd Calc/RiemannProblems
```

- Starten Sie nun das Programm mit folgendem Aufruf:

```
../..cfdfv_sod.ini
```

- War die Ausführung nicht erfolgreich, so liegt wahrscheinlich noch ein Programmierfehler vor. Gehen Sie in diesem Fall zurück zum letzten Arbeitspunkt und überprüfen Sie die Datei *Flux_godunov.f90* auf Fehler.
- War die Ausführung erfolgreich, starten Sie das Visualisierungstool *Visit*:

```
visit_&
```

Hinweis: Alle Dateinamen, die auf *_ex-xxxx.curve* enden, stellen die exakte Lösung dar und dienen der Validierung der numerischen Lösung. Dies gilt allerdings nur für 1D-Probleme, für 2D-Probleme gibt es im Programm *cfdv* keine exakten Lösungen. Die Zahl am Ende des Dateinamens gibt den Ausgabezeitpunkt an. Wählen Sie unbedingt den gleichen Ausgabezeitpunkt für alle gleichzeitig angezeigten Dateien, wenn Sie die Ergebnisse vergleichen wollen.

Laden Sie die numerische und die exakte Lösung und vergleichen Sie die Ergebnisse.

3 Aufgabenteil 3: Programmierung der Riemannlöser

Nun sollen verschiedene Flussfunktionen selbst programmiert werden. Das Vorgehen ist hierbei identisch wie in Aufgabenteil 2: Kopieren Sie sich die entsprechende Rumpfdati aus dem Verzeichnis *LeereFiles* und programmieren Sie die Flussberechnung in die Datei. Hier noch ein paar Hinweise zum Vorgehen:

- Wechseln Sie ins Projektverzeichnis: `cd ~ /CFDFV`
- kopieren Sie die entsprechende Rumpfdati in das Verzeichnis `cp LeereFiles/Flux_XXXX.f90`. Die entsprechenden Dateinamen sind:
 - Roe-Verfahren: `Flux_roe.f90`
 - HLLE-Verfahren: `Flux_hlle.f90`
 - Lax-Friedrichs-Verfahren: `Flux_lax_friedrichs.f90`
 - Steger-Warming-Verfahren: `Flux_steger_warming.f90`

Achten Sie darauf, immer nur eine Datei in das Projektverzeichnis zu holen - eine noch nicht bearbeitete Rumpfdati in diesem Verzeichnis wird das Kompilieren unmöglich machen.

- Programmieren Sie die Flussberechnung nach einer der unten stehenden Anleitungen
- Kompilieren Sie das Programm und testen Sie es wie in Aufgabenteil 2 beschrieben.

3.1 Godunov-Typ Verfahren

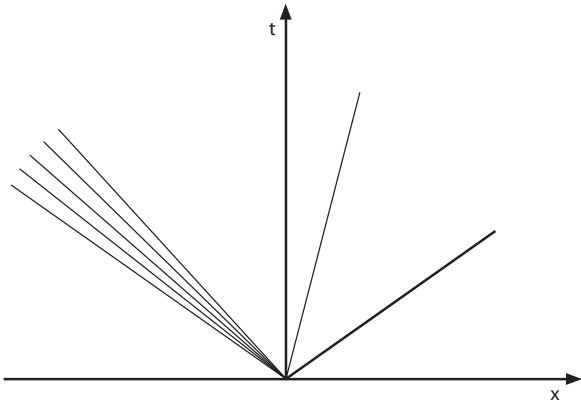
Grundprinzip: Bestimmung der Flüsse durch (approximatives) Lösen eines Riemann-Problems an den Zellgrenzen.

Die Flußdifferenz am Rand eines Gitterintervalls wird in die Anteile nach rechts und links zerlegt und geeignet approximiert. Man nennt diese Verfahren daher auch Flußdifferenzen-Splitting Verfahren.

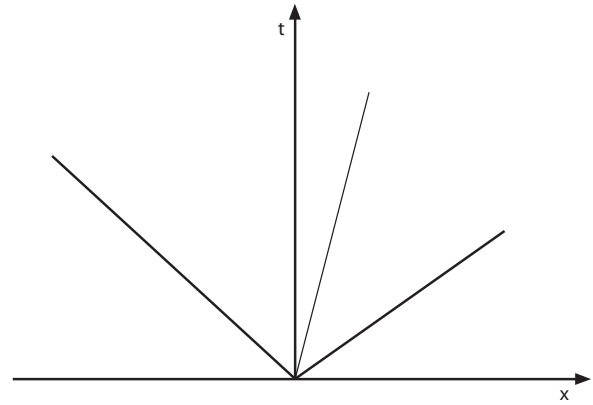
3.1.1 Roe-Verfahren

Hintergrund

Phil Roe hat in den 80er Jahren einen Riemannlöser auf Basis der Linearisierung der Eulergleichungen vorgestellt. Die Jacobimatrix der Eulergleichungen wird mittels der sog. Roe-Mittelwerte linearisiert, womit nur noch das linearisierte Riemannproblem zu lösen ist, was sich als sehr einfach gestaltet. Die Auswirkungen auf das lokale Riemannproblem an der Zellgrenze sind am Besten im x-t-Diagramm zu zeigen:



(a) Das Riemannproblem der Eulergleichungen



(b) Das Riemannproblem der linearisierten Eulergleichungen

Es ist recht deutlich zu sehen, dass der Verdünnungsfächer durch einen „Verdünnungsstoß“ ersetzt wird. Dies ist zwar physikalisch falsch, da Verdünnungsstöße die Entropiebedingung verletzen, uns geht es aber lediglich um den Fluss über die Zellgrenze, also kann der Fehler, den wir machen an sich keine große Auswirkung haben. Das einzige Problem, das auftreten kann, ist wenn der Verdünnungsstoß direkt auf der Zellgrenze liegt, die Ausbreitungsgeschwindigkeit also null ist. Dies ist genau dann der Fall, wenn $u = c$ ist, da sich die Verdünnungswelle mit der Geschwindigkeit $u - c$ ausbreitet. Ein solcher Fall läßt sich sehr leicht an der Machzahl ablesen, die unter diesen Umständen nämlich genau $M = 1$ ist. Dieses Problem tritt dann auf, wenn in einer globalen Verdünnung dieser Durchgang stattfindet, man spricht vom sog. *Sonic Glitch* Problem.

Umsetzung im Code

Lösung des exakten Riemann-Problems für die linearisierte Erhaltungsgleichung

$$u_t + A_{lr}u_x = 0$$

Dabei muß die Matrix A_{lr} folgende 3 Bedingungen erfüllen:

1. $A_{lr}(u, u) = A(u)$ (mit $A(u) = \partial f_1(u)/\partial u$ Jacobi-Matrix)
2. $A_{lr}(u, u)$ ist diagonalisierbar (Hyperbolizität)
3. $A_{lr}(u, u)$ besitzt die Mittelwerteigenschaft $f_1(u_r) - f_1(u_l) = A_{lr}(u_r - u_l)$

Die Matrix $A_{lr} = A(\bar{u})$ mit den Mittelwerten

$$\bar{v}_1 = \frac{\sqrt{\rho_r}v_{1r} + \sqrt{\rho_l}v_{1l}}{\sqrt{\rho_r} + \sqrt{\rho_l}}, \bar{v}_2 = \frac{\sqrt{\rho_r}v_{2r} + \sqrt{\rho_l}v_{2l}}{\sqrt{\rho_r} + \sqrt{\rho_l}}, \bar{H} = \frac{\sqrt{\rho_r}H_r + \sqrt{\rho_l}H_l}{\sqrt{\rho_r} + \sqrt{\rho_l}}$$

$$\bar{c}^2 = (\gamma - 1) \left(\bar{H} - \frac{1}{2}\bar{v}^2 \right) \quad H = \frac{E + p}{\rho}$$

$$\bar{v}^2 = \bar{v}_1^2 + \bar{v}_2^2$$

erfüllt diese Bedingungen. Die Eigenwerte der Matrix A_{lr} sind

$$a_1 = \bar{v}_1 - \bar{c}, a_2 = a_3 = \bar{v}_1, a_4 = \bar{v}_1 + \bar{c},$$

die Eigenvektoren

$$r_1 = \begin{pmatrix} 1 \\ \bar{v}_1 - \bar{c} \\ \bar{v}_2 \\ \bar{H} - \bar{v}_1 \bar{c} \end{pmatrix}, \quad r_2 = \begin{pmatrix} 1 \\ \bar{v}_1 \\ \bar{v}_2 \\ \frac{1}{2} (\bar{v}_1^2 + \bar{v}_2^2) \end{pmatrix}, \quad r_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \\ \bar{v}_2 \end{pmatrix}, \quad r_4 = \begin{pmatrix} 1 \\ \bar{v}_1 + \bar{c} \\ \bar{v}_2 \\ \bar{H} + \bar{v}_1 \bar{c} \end{pmatrix}.$$

Mit den Bezeichnungen

$$\begin{aligned} \Delta\rho &= \rho_r - \rho_l & \Delta m_1 &= \rho_r v_{1r} - \rho_l v_{1l} & \Delta m_2 &= \rho_r v_{2r} - \rho_l v_{2l} \\ \Delta E &= E_r - E_l & \overline{\Delta E} &= \Delta E - (\Delta m_2 - \bar{v}_2 \Delta\rho) \bar{v}_2 \end{aligned}$$

erhält man aus der Bedingung $u_r - u_l = \sum_{i=1}^4 \gamma_i r_i$ die Koeffizienten

$$\begin{aligned} \gamma_2 &= -\frac{\gamma - 1}{\bar{c}^2} [\Delta\rho (\bar{v}_1^2 - \bar{H}) - \bar{v}_1 \Delta m_1 + \overline{\Delta E}] \\ \gamma_1 &= -\frac{1}{2\bar{c}} [\Delta m_1 - \Delta\rho (\bar{v}_1 + \bar{c})] - \frac{1}{2} \gamma_2 \\ \gamma_4 &= \Delta\rho - \gamma_1 - \gamma_2 \\ \gamma_3 &= \Delta m_2 - \bar{v}_2 \Delta\rho \end{aligned}$$

Damit erhält man den Roe-Fluß als

$$g_{i+1/2,j} = \frac{1}{2} (f_1(u_{i+1,j}) + f_1(u_{i,j})) - \frac{1}{2} \sum_{k=1}^4 \gamma_k |a_k| r_k$$

Das Roe-Verfahren läuft dann folgendermaßen ab:

- Berechne die Roe-Mittelwerte $\bar{v}_1, \bar{v}_2, \bar{H}$ und \bar{c}
- Berechne die gemittelten Eigenwerte a_1, a_2, a_3, a_4
- Berechne die gemittelten Rechtseigenvektoren r_1, r_2, r_3, r_4
- Berechne die Faktoren $\gamma_1, \gamma_2, \gamma_3, \gamma_4$
- Berechne den Roe-Fluß $g_{i+1/2,j}$

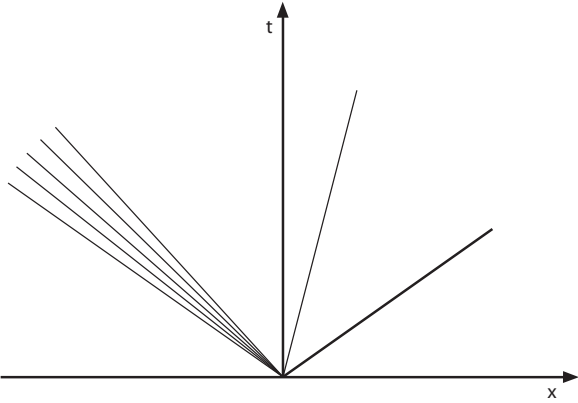
Eigenschaften des Roe-Verfahrens sind

- Ein einzelner Stoß oder eine einzelne Kontaktunstetigkeit wird exakt aufgelöst
- Unterschätzen der Wellengeschwindigkeit bei Verdünnungen
- Bei starken Verdünnungen kann die Konsistenz mit der Entropiebedingung verletzt sein (Entropy-Fix notwendig)

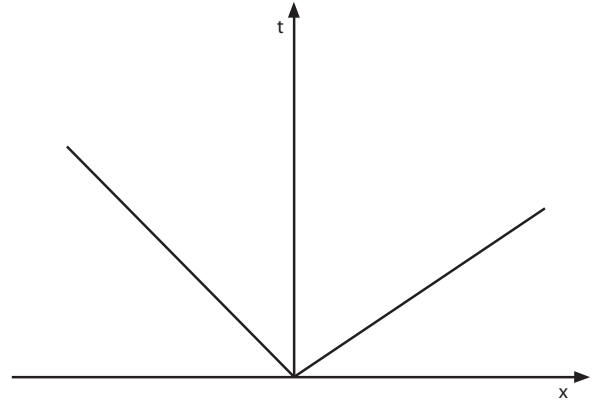
3.1.2 HLLE-Verfahren

Hintergrund

Der HLLE-Riemannlöser vereinfacht das lokale Riemannproblem noch weiter als der von Roe. Hier wird die Kontaktunstetigkeit zwischen den beiden äußeren Wellen vollkommen ignoriert, so daß sich nur noch ein mittlerer Zustand ergibt:



(c) Das Riemannproblem der Eulergleichungen



(d) Das Riemannproblem beim HLLE-Riemannlöser

Zwar stellt dies eine starke Vereinfachung gegenüber der Realität dar, jedoch ist es trotzdem oft ausreichend, das lokale Riemannproblem auf diese Art und Weise zu lösen - global ergibt sich dennoch eine gute Näherungslösung. Aufgrund seines Konstruktionsprinzips sind beim HLLE-Verfahren Schwächen vor allem in der Auflösung der Kontaktunstetigkeit festzustellen. Vorteil des Verfahrens ist seine Robustheit und die einfache Anwendbarkeit auf andere Gleichungen. Die Linearisierung von Roe ist in vielen Bereichen nicht anwendbar, z.B. bei Strömungen von realen Gasen. In diesen Fällen stellt das HLLE-Verfahren eine sehr gute Möglichkeit der Flussberechnung dar.

Umsetzung im Code

Approximative Lösung des Riemann-Problems: nur ein mittlerer Zustand

$$g_{i+1/2,j} = \frac{a_r^+ f_1(u_{ij}) - a_l^- f_1(u_{i+1,j})}{a_r^+ - a_l^-} + \frac{a_r^+ a_l^-}{a_r^+ - a_l^-} (u_{i+1,j} - u_{ij})$$

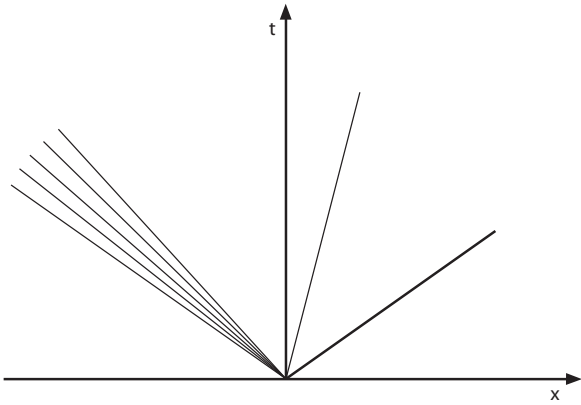
mit den Signalgeschwindigkeiten $a_r^+ = \max \{0, v_{1r} + c_r, \bar{v}_1 + \bar{c}\}$, $a_l^- = \min \{0, v_{1l} - c_l, \bar{v}_1 - \bar{c}\}$.

\bar{v}_1 und \bar{c} sind die Roe-Mittelwerte aus 3.1.1.

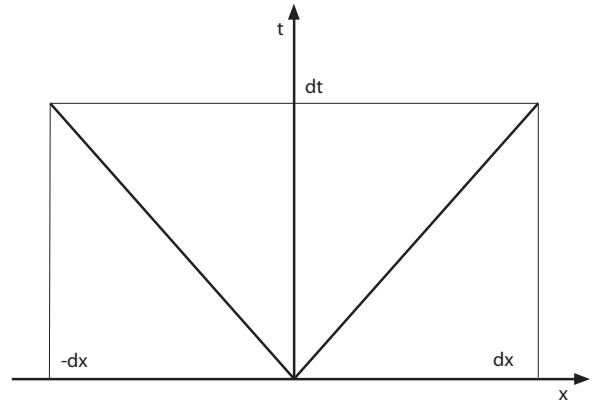
3.2 Lax-Friedrichs Verfahren

Hintergrund

Das Lax-Friedrichs-Verfahren ist die einfachste und grösste Variante eines HLL-Verfahrens. Es ist an sich ein „altes“ Verfahren, das ursprünglich im Finite-Differenzen-Kontext entwickelt wurde und erst später Einzug in die FV-Verfahren erhalten hat.



(e) Das Riemannproblem der Eulergleichungen



(f) Das Riemannproblem beim Lax-Friedrichs-Verfahren

Man erkennt sofort die Ähnlichkeit zum HLLE-Verfahren, allerdings ist es hier so, dass nicht in jeder Zelle die Ausbreitungsgeschwindigkeit jeder individuellen Welle bestimmt wird, sondern es wird angenommen, dass die Ausbreitungsgeschwindigkeit einer jeden Welle so ist, daß sie innerhalb eines Zeitschritts Δt genau den Rand der Zelle erreicht. Dadurch hat jede betrachtete Welle im Rechengebiet die gleiche Ausbreitungsgeschwindigkeit, was natürlich eine noch größerer Vereinfachung darstellt als beim HLLE-Verfahren, wo zumindest für jede individuelle Welle eine Ausbreitungsgeschwindigkeit berechnet wird.

Umsetzung im Code

Wähle $a_l = -\frac{\Delta x}{\Delta t}$, $a_r = \frac{\Delta x}{\Delta t}$. Dann ergibt sich

$$g_{i+1/2,j} = \frac{1}{2} (f_1(u_{i+1,j}) + f_1(u_{i,j})) - \frac{\Delta x}{2\Delta t} (u_r - u_l)$$

Für das Lax-Friedrichs Verfahren muß die CFL-Bedingung

$$\frac{\Delta x}{\Delta t} > \max \{|v - c|, |v + c|\}$$

eingehalten werden.

Im Code wird tatsächlich eine lokale Form des Lax-Friedrichs Verfahrens implementiert: der **Rusanov-Fluss**.

$$\lambda_{max} = \max ((|v_1| + c)_L, (|v_1| + c)_R)$$

$$g_{i+1/2,j} = \frac{1}{2} (f_1(u_{i+1,j}) + f_1(u_{i,j})) - \frac{\lambda_{max}}{2} (u_r - u_l)$$

3.3 Flußvektor-Splitting Verfahren

Aufteilung der Flüsse in nach rechts und nach links laufende Anteile:

$$f_1(u) = f_1^+(u) + f_1^-(u)$$

Der numerische Fluß setzt sich dann aus beiden Anteilen wieder zusammen.

$$g_{i+1-2,j} = f_1^+(u_{ij}) + f_1^-(u_{i+1,j}).$$

3.3.1 Steger-Warming

Hintergrund

Das Steger-Warming-Verfahren ist ein Vertreter der Flussvektorsplittingverfahren. Flußvektorsplittingverfahren gehen auf die Idee der Upwind-Verfahren zurück, die bei der Diskretisierung die Richtung der Informationsausbreitung berücksichtigen. Die Verfahren sind generell einfacher als die Godunov-Typ Verfahren, was die Verfahren schnell und effizient macht, natürlich mit gewissen Einbußen in der Genauigkeit im Vergleich z.B. zum Roe-Verfahren. Flussvektorsplittingverfahren lassen sich aber hervorragend bei impliziten Zeitdiskretisierungen einsetzen, weshalb sie sich gerade hierbei großer Beliebtheit erfreuen.

Das Steger-Warming-Verfahren wird konstruiert, indem die Jacobi-Matrix der Eulergleichungen in einen positiven und einen negativen Anteil aufgespalten wird:

$$A^+ = R\Lambda^+R^{-1} \quad A^- = R\Lambda^-R^{-1},$$

wobei Λ^+ und Λ^- Nullmatrizen mit den positiven bzw. negativen Eigenwerten auf der Hauptdiagonalen sind.

Umsetzung im Code

Grundidee: Diagonalisierung der Jacobi-Matrix $A(u)$ und Aufteilung der Diagonalmatrix in eine positive und eine negative. Die positive kommt von links und läuft nach rechts, die negative umgekehrt. Mit den Eigenwerten $a_1 = v_1 - c, a_2 = a_3 = v_1, a_4 = v_1 + c$ ergibt sich

$$f_1^\pm = \begin{pmatrix} f_{1,1}^\pm \\ f_{1,2}^\pm \\ f_{1,3}^\pm \\ f_{1,4}^\pm \end{pmatrix} = \begin{pmatrix} \frac{\rho}{2\gamma} (2(\gamma-1)a_2^\pm + a_1^\pm + a_4^\pm) \\ f_{1,1}^\pm v_1 + (a_4^\pm - a_1^\pm) \frac{\rho c}{2\gamma} \\ f_{1,1}^\pm v_2 \\ f_{1,1}^\pm \frac{v_1^2 + v_2^2}{2} + (a_4^\pm - a_1^\pm) \frac{\rho c v_1}{2\gamma} + (a_4^\pm + a_1^\pm) \frac{\rho c^2}{2\gamma(\gamma-1)} \end{pmatrix}.$$

Dabei bezeichnet a_i^\pm den positiven Eigenwert $a_{l,i}$ oder den negativen Eigenwert $a_{r,i}$, also

$$a_i^+ = \max(a_{l,i}, 0) = \begin{cases} a_{l,i} & \text{falls } a_{l,i} > 0 \\ 0 & \text{sonst} \end{cases} \\ a_i^- = \min(a_{r,i}, 0) = \begin{cases} a_{r,i} & \text{falls } a_{r,i} < 0 \\ 0 & \text{sonst} \end{cases}.$$

Die übrigen Variablen sind gemäß der obigen Definition des Flusses zu wählen, also für f^+ die Werte von links, $\rho_l, v_{1l}, v_{2l}, c_l$, für f^- die von rechts, $\rho_r, v_{1r}, v_{2r}, c_r$.

4 Aufgabenteil 4: Vergleich verschiedener Flussfunktionen

4.1 Hintergrund

Die Bestimmung der exakten Lösung eines Riemannproblems an einer Zellgrenze stellt einen großen Rechenaufwand dar, der für numerische Rechnungen nicht praktikabel ist. Es wurden daher approximative Riemannlöser mit reduziertem Aufwand entwickelt. Diese Gruppe von Verfahren bezeichnet man als Godunov-Typ oder Flussdifferenzen-Splitting-Verfahren. Eine weitere Klasse von Flussfunktionen stellen die sog. Flußvektor-Splitting-Verfahren dar, die auf die Lösung des Riemannproblems verzichten. Sie basieren auf der Zerlegung des Flussvektors in einen nach rechts und einen nach links laufenden Anteil. In dieser Aufgabe sollen verschiedene klassische Vertreter dieser Verfahren angewendet und miteinander verglichen werden.

4.2 Genaue Aufgabenstellung

4.2.1 Auflösungsvermögen der Riemannlöser

- Berechnen Sie das Sod-Problem mit folgenden Riemannlösern:
 - Godunov (1)
 - Roe (2)
 - HLLE (3)
 - Lax-Friedrichs (5)
 - Steger-Warming (6)

Gehen Sie dabei folgendermaßen vor:

- Öffnen Sie eine Shell
- Gehen Sie ins Verzeichnis für Rechnungen:
`cd ~/CFDFV/Calc/RiemannProblems`
 und öffnen Sie die Datei `sod.ini`:
`gedit sod.ini &`
- Der Inhalt der Datei ist:


```
! SOD Testcase
!-----!
Mesh:
  1           ! Mesh type 0=UNStructured, 1=CARTesian   !
 100         ! imax
```

```

1          ! jmax
0.         ! xMin
0.         ! yMin
1.         ! xMax
0.01      ! yMax
1          ! Number of bottom boundary segments
101       ! BC Type of boundary segment
1          ! Number of right boundary segments
401      ! BC Type of boundary segment
1          ! Number of top boundary segments
101     ! BC Type of boundary segment
1          ! Number of left boundary segments
401     ! BC Type of boundary segment
!-----!
!-----!
Const:
1          ! Eqn type: Euler=1          !
1.4       ! Gamma                      !
15000    ! Maximum iteration number   !
0.25     ! Final simulation time      !
!-----!
Discretization:
0.99     ! CFL number                    !
1         ! Flux function                !
1         ! Order of temporal discretisation !
1         ! Order of spatial discretisation !
1         ! Limiter                      !
! 10.    ! Constant for Venkatakrisnan's Limiter !
0         ! stationary/transient (1/0) problem !
0         ! local/global (1,0) timestepping !
!-----!
InitialCondition:
2         ! exact function                !
5         ! 1D Riemann Problem           !
0.5      ! Interface                    !
1.0      ! rho_l                        !
0.0      ! u_l                          !
1.0      ! p_l                          !
0.125    ! rho_r                        !
0.0      ! u_r                          !
0.1      ! p_r                          !
!-----!
Boundaries:
2         ! number of boundaries          !
101      ! BC type                      (slipwall/symmetry) !
401      ! BC type                      (outflow)          !
4        ! dmr                          !

```

```

!-----!
FileIO:
sod                ! Filename for data output          !
  0.25             ! Timeinterval for data output          !
 100000000         ! Iteration Interval for data output    !
  2                ! Visual Program                        !
  1                ! compute errors (1:yes/0:no)          !
!-----!
ERRORS:
!-----!
!File should be finished with a END statement:
END

```

Im oberen Teil (Mesh:) wird das Rechengitter beschrieben. Wir haben für unseren 1D-Fall eine Ausdehnung von 100 Zellen in x-Richtung und eine Ausdehnung von einer Zelle in y-Richtung. Die Länge des Gebiets ist auf 1 normiert. Diese Werte sollen zunächst so beibehalten werden.

Im Teil `Discretization`: werden alle Einstellungen für das Verfahren vorgenommen. In unserem Fall ist die Zeile

```

1                ! Flux function          !

```

interessant. Hier können wir die zu verwendende Flussfunktion einstellen. In der obigen Aufstellung ist hinter jedem Riemannlöser in Klammern die Zahl eingegeben, die man ins Ini-File eintragen muss, um mit dem entsprechenden Verfahren zu rechnen.

Im Teil `FileIO`: geht es um die Dateiausgabe der Ergebnisse. Hier ist vor allem eine Zeile interessant:

```

Sod                ! Filename for data output          !

```

Hier wird der Dateiname angegeben, den die Ausgabedateien haben sollen. Das Programm hängt noch die Numerierung für die Ausgabezeitpunkte bzw. die Kennzeichnung der exakten Lösung an. Um verschiedene Riemannlöser vergleichen zu können, sollten die Ausgabedateien auch entsprechend anders heißen. Verwenden Sie mit jedem neuen Riemannlöser auch einen neuen Dateinamen, z.B. `Sod_Godunov` für das Godunov-Verfahren oder z.B. `Sod_Roe` für das Roe-Verfahren.

- Speichern Sie die Datei und starten Sie das Programm:
`../..cfdfv_sod.ini`
- Wiederholen Sie dieses Vorgehen für jeden Riemannlöser.
- Starten Sie dann *Visit* und visualisieren Sie die Ergebnisse. Gehen Sie dabei genauso vor, wie in Aufgabenteil 2. Sie können alle Ergebnisse gleichzeitig darstellen, indem Sie auch die unteren Felder für weitere Ausgabedateien nutzen. Visualisieren Sie immer nur die Dateien mit der Endung `-0001.curve`. Die exakte Lösung ist bei allen Lösungen gleich. Achten Sie darauf, daß Sie die exakte Lösung nur im obersten Feld laden. Welche exakte Lösung Sie dabei nehmen spielt keine Rolle. Sollte eine numerische Lösung (gepunktet) doch mit der exakten Lösung übereinstimmen, kontrollieren Sie, ob nicht ein `_ex` im Dateinamen enthalten ist.

- Vergleichen Sie nun die folgende Eigenschaften der Ergebnisse für jeden Riemannlöser. Fertigen Sie hierzu eine Tabelle an:

- Auflösung des Stoßes
Wie viele Punkte benötigt die Lösung ungefähr, um den Stoß aufzulösen?
 - Auflösung der Kontaktunstetigkeit
Wie viele Punkte benötigt die Lösung ungefähr, um die Kontaktunstetigkeit aufzulösen?
 - Auflösung der Verdünnung
Wie sind Beginn und Ende des Verdünnungsfächers aufgelöst?
- Wiederholen Sie das Vorgehen nun noch für die Testfälle Toro 1 und Toro 3.
Beachten Sie beim Vergleich der Ergebnisse des Testfalls Toro 1 nur den Verdünnungsfächer - hier tritt ein Fehler auf, der beim Sod-Problem nicht zu beobachten war. Woran könnte er liegen und warum tritt er nicht bei jedem Riemannlöser auf? **Hinweis:** Lassen Sie sich die Machzahl (bzw. Geschwindigkeit) im Verdünnungsfächer anzeigen.
Beim Testfall Toro 3 ist vor allem die Höhe des Plateaus zwischen Stoß und Kontaktunstetigkeit entscheidend. Vergleichen Sie diese für alle Flussfunktionen.
 - Advanced: Warum wird der Stoß besser dargestellt als die Kontaktunstetigkeit?

4.2.2 Geschwindigkeit der Riemannlöser

Sie haben nun die Riemannlöser im Hinblick auf Genauigkeit verglichen und wichtige Stärken und Schwächen der einzelnen Verfahren kennengelernt. Im zweiten Teil der Aufgabe wenden wir uns nun dem anderen wichtigen Kriterium bei CFD-Rechnungen zu, nämlich der Geschwindigkeit. Hierzu berechnen wir wieder den Sod-Testfall mit allen Riemannlösern und vergleichen die Rechenzeit. Ihnen wird vielleicht aufgefallen sein, daß das Programm die Rechenzeit am Ende einer Rechnung ausgibt. Da aber die Probleme so klein sind, sodass die Gefahr besteht, dass die Rechenzeiten nicht aussagekräftig sind, müssen wir die Probleme künstlich vergrößern, indem wir die Anzahl der Gitterpunkte erhöhen.

Gehen Sie dazu folgendermaßen vor:

- Öffnen Sie wieder die Datei *Sod.ini*: `gedit sod.ini &`
- Ändern Sie in der ersten Zeile nach *Mesh*:

```
100           ! Number of cells in x-direction           !,
```

die Zellenzahl auf **1000**.
- Ändern Sie den Riemannlöser zunächst wieder auf das Verfahren von Godunov (1).
- Um Ihre zuvor erzeugten Daten nicht zu löschen, ändern Sie noch unter *FileIO* den Ausgabedateinamen auf *Sod_Time*. Behalten Sie den Dateinamen für alle weiteren Rechnungen bei. Eine Visualisierung der Ergebnisse ist hier nicht erforderlich - es geht ausschließlich um die Rechenzeit!

- Speichern Sie die Datei und führen das Programm aus. Die Berechnung sollte nun merklich länger dauern. Notieren Sie die Rechenzeit, und wiederholen Sie das Vorgehen für alle anderen Riemannlöser. Welcher Riemannlöser ist der schnellste und welcher ist der langsamste?