

CFD-Programmier-Seminar

Projekt 3: Randbedingungen, Zeitdiskretisierung, CFL-Bedingung und Zustandsgleichung

Randbedingungen

Da man leider nur endliche Rechengebiete verwenden kann, stellt sich die Frage, was am Rand passieren muss, denn für die Berechnung der Flüsse über die Randkanten steht nur ein Zustand, nämlich der der inneren Zelle, zur Verfügung. Den Zustand auf der anderen Seite muss man basierend auf dem Typ des Rands über spezielle Randbedingungen vorgeben, von denen hier einige umgesetzt werden sollen.

In einem Finite-Volumen-Verfahren wird die Änderung des Inhalts einer Zelle als Summe der Integrale der Flüsse über die Seiten der Zelle berechnet. Für die Flussberechnung werden jeweils zwei Zustände, nämlich links und rechts der Seite, benötigt. An Randkanten ist jedoch nur eine Zelle vorhanden. Bei der Implementierung der Randbedingungen nutzen wir diesen Sachverhalt, indem wir dort jeweils eine sog. Ghostzelle einführen, deren Zustand wir so bestimmen, dass die Lösung des entsprechenden Riemannproblems über die Randkante genau dem entspricht, was wir dort physikalisch vorgeben wollen.

Wandrandbedingungen

Bei Wandrandbedingungen für die Eulergleichungen muss die Normalkomponente der Geschwindigkeit gespiegelt werden. Achtung! Bitte implementieren Sie die Wandrandbedingung für den CASE SLIPWALL. Der CASE WALL ist eine Navier-Stokes Wandrandbedingung. Da wir mit Eulergleichungen rechnen, trifft dies nicht auf unsere Probleme zu.

Zur Erklärung: Die Slipwall sorgt lediglich dafür, dass die Normalkomponente v_1 der Geschwindigkeit zu null wird. Die Tangentialkomponente v_2 bleibt erhalten. Da die Eulergleichungen keine Reibung berücksichtigen, ist dies auch korrekt. Die Navier-Stokes-Gleichungen hingegen berücksichtigen die Wandreibung und haben somit auch einen Einfluss auf die Tangentialkomponente.

Einströmrund (supersonisch)

Hier wird einfach nur ein Zustand vorgegeben, der vom Anwender definiert wurde (`Side%BC%pvar`)

Ausströmrund (supersonisch)

Hier wird der Zustand der inneren Zelle extrapoliert, d. h. er wird einfach kopiert.

Implementierung

Die Implementierung erfolgt in der Datei `Boundary.f90`. Dort sind drei Routinen umzusetzen:

- `Boundary(CONST, MESH, aSide, time, int_pvar, ghost_pvar, x)`:
ACHTUNG!!! In der `SetBCatSides(CONST, MESH)` lautet der Aufruf `Boundary(CONST, MESH, gSide, time, int_pvar, ghost_pvar, x)`. Diese Routine liefert für einen Zustand `int_pvar` einen Zustand `ghost_pvar`. Weitere Variablen sind:
 - `x`: Die Position, an der der Randzustand berechnet werden soll.
 - `aSide`: Zeiger auf die Seite der Ghostzelle. In der `SetBCatSides` wird `gSide` dafür an `Boundary` übergeben. Leider wird dadurch aus dem `gSide`-Pointer in der `SetBCatSides` ein namentlicher `aSide`-Pointer in der `Boundary`. Der `aSide`-Pointer in der `Boundary` hat jedoch nichts mit dem `aSide`-Pointer in der `SetBCatSides` zu tun. In der `SetBCatSides` ist `aSide` die interne Seite. In der `Boundary` ist `aSide` die Ghostseite! Der Normalenvektor in der `Boundary` für die Rotation sollte jedoch aus der Seite der eigentlichen Zelle, also `aSide%Connection`, entnommen werden.
 - `time`: Die Zeit, an der die Randbedingung gesetzt werden soll, `CONST*t + aElem%dt_loc`.
 - `CONST` und `MESH`: Alle Konstanten, werden einfach übergeben.

Der Zustand wird im globalen Koordinatensystem übergeben und muss auch im globalen Koordinatensystem zurück gegeben werden.

- `SetBCatSides(CONST, MESH)`:
Diese Routine setzt die Randbedingung am Mittelpunkt der Ghostseite (`aElem%bary + aSide%GP`). Hierfür ist eine Schleife über alle Randseiten zu implementieren: `MESH%BCSides(:)`. Das `BCSides`-Array enthält dabei die Ghostzellen. Das heißt, dass Sie bei der Implementierung den Zeiger `gSide` als Laufpointer für das `MESH%BCSides(:)`-Array verwenden sollten.

Definition der Randbedingungen im ini-File

Zur Einstellung der Randbedingungen müssen Sie im ini-File zwei Abschnitte berücksichtigen. Im Abschnitt `Mesh` müssen Sie angeben, welche Randbedingungen die einzelnen Randseiten haben. Hier definieren Sie also, wo das Gitter welche Randbedingungen hat. Dies müssen Sie im ini-File machen, wenn Sie ein kartesisches Gitter verwenden, wie es in den bisherigen Testfällen der Fall war. Wenn sie jedoch ein unstrukturiertes Gitter aus einem Gittergenerator verwenden, müssen Sie diese Information dem Gittergenerator mitteilen. In diesem Fall sieht der `Mesh`-Abschnitt im ini-File anders aus (s. ini-File der Keilprofilauflage). Sowohl für die Riemannprobleme als auch für den Gauss-Puls können Sie folgendes Setup verwenden, um ein abgeschlossenes Rechengebiet zu erzeugen. Dabei ist "101" der Typ der Wand-Randbedingung, die Sie implementiert haben.

```
1           ! Number of bottom boundary segments
101        ! BC Type of boundary segment
1           ! Number of right boundary segments
101        ! BC Type of boundary segment
1           ! Number of top boundary segments
101        ! BC Type of boundary segment
1           ! Number of left boundary segments
101        ! BC Type of boundary segment
```

Folgende Typen können Sie nun definieren:

- 101: Euler-Wand
- 301: supersonischer Einströmrand
- 401: supersonischer Ausströmrand
- 601: exakte Funktion
- 701: periodischer Rand

Jetzt müssen Sie noch dem Code mitteilen, welche Randbedingungen er kennen muss, um sie dem Gitter richtig zuweisen zu können. Dafür müssen Sie in der Boundaries-Section angeben, wie viele Boundaries Sie haben und von welchem Typ. Dies geschieht in den Riemannproblemen wie folgt:

```
Boundaries:
  2                ! number of boundaries
  101              ! BC type          (slipwall/symmetry)    !
  401              ! BC type          (outflow)              !
```

Diese Angaben müssen sowohl für kartesische Gitter, als auch für unstrukturierte Gitter (Gittergenerator!) gemacht werden. Ohne diese Angaben, wird der Code die BC, welche Sie im Gitter definiert haben, nicht verwenden können.

Rechnungen

Achten Sie bitte bei allen folgenden Beispielen darauf, dass im Parameterfile die Ordnung der Raum- und der Zeitdiskretisierung auf eins gesetzt wurde.

1. Rechnen Sie zuerst den Gauss-Puls mit Wandrandbedingungen. Das Rechengebiet ist somit abgeschlossen und es gibt keine Ausflussränder. Visualisieren Sie das Ergebnis in einer Animation. Verwenden Sie hierzu die Video-Funktionalität unter Visit, indem Sie beim Öffnen der Datei die “file grouping” Option auf “off” (statt “smart”) stellen und nur die Master-Datei (*_Master.cgns) auswählen. Nun können Sie den Timeslider in Visit verwenden (drücken Sie hierzu einfach auf die play-Taste).
2. Berechnen Sie nun das SOD-Problem mit Wandrandbedingungen. Dies entspricht in etwa dem, was im Stoßrohr-Windkanal am IAG passiert. Werten sie das Ergebniss aus¹. Verwenden Sie auch hier die Video-Funktionalität in Visit.
3. Um die supersonischen Randbedingungen zu überprüfen, rechnen Sie das Double-Mach-Reflection- und das Forward-Facing-Step-Beispiel mit einem Verfahren erster Ordnung in Raum und Zeit. Welche Phänomene treten auf?

¹Es werden Reflexionen an den Wänden entstehen. Die Verdünnung wird in den Stoß laufen und umgekehrt. Welches Phänomen (Stoß oder Verdünnung) läuft schneller?

Zeitschrittweite

Der maximal mögliche, stabile Zeitschritt einer Zelle i kann mit folgender Formel bestimmt werden:

$$\Delta t_i = CFL \frac{\text{Zellvolumen}}{\Lambda^x + \Lambda^y}$$

Die Werte im Nenner sind:

$$\Lambda^x = (|v_1| + c) \Delta S^x$$

$$\Lambda^y = (|v_2| + c) \Delta S^y$$

Die Werte ΔS^x und ΔS^y sind die Projektion der Zelle auf die x- bzw. y-Achse und stehen im Code bereits zur Verfügung.

Da für instationäre Probleme alle Zellen den gleichen Zeitschritt gehen müssen, bestimmt die Zelle mit dem kleinsten maximal möglichen Zeitschritt den globalen Zeitschritt:

$$\Delta t^{global} = \min_i (\Delta t_i)$$

Implementierung

Die Implementierung erfolgt in der Datei `CalcTimeStep.f90`. Der Zeitschritt jeder Zelle wird in `Elem%dt` gespeichert.

Zeitdiskretisierung

Das Raum-Zeit-Integral

$$\int_{t^n}^{t^n + \Delta t} \int_{L_{ij}} g(U_i(x, y, t), U_j(x, y, t)) ds dt$$

wird numerisch durch einfache Quadraturlformeln gelöst:

$$\Delta t \sum_{j=1}^{nSides} L_{i,j} g(U_i(x, y, t^n), U_j(x, y, t^n)).$$

Implementierung

Die Implementierung erfolgt in der Datei `CalcCFDFV.f90` in der Subroutine `CalcSTE`. Dort müssen die in `Side%Flux` gespeicherten Flussintegrale summiert und, wie oben angegeben, in der Zeit integriert werden.

Rechnungen

Achten Sie bei allen folgenden Beispielen darauf, dass im Parameterfile die Ordnung der Raum- und der Zeitdiskretisierung auf eins gesetzt wurde.

1. Berechnen Sie die Riemannprobleme (SOD, Toro1 und Toro3) mit unterschiedlichen CFL-Zahlen. Überprüfen Sie, wann die Rechnung instabil wird. Stellen Sie sicher, dass Ihr Ausgabeintervall größer als der entstandene Zeitschritt ist. Sonst verwendet der Code den Ausgabe-Zeitschritt und nicht den vorgegebenen Zeitschritt. Dies kann dazu führen, dass für große CFL-Zahlen die Rechnung noch stabil bleibt, weil nicht der CFL-Zeitschritt, sondern der Ausgabe-Zeitschritt verwendet wird.

2. Berechnen sie anhand des NACA2312-Profiles den Einfluss der CFL-Zahl auf die Rechenzeit. Stellen Sie sicher, dass sie im ini-File global time-stepping verwenden. Der Default ist leider noch auf local-timestepping gesetzt. Dies funktioniert jedoch in der derzeitigen Codeversion nicht.

Zustandsgleichung

Wir benötigen eine Umrechnung zwischen den sog. konservativen Variablen

$$\begin{pmatrix} \rho \\ \rho v_1 \\ \rho v_2 \\ E \end{pmatrix}$$

und den primitiven Variablen

$$\begin{pmatrix} \rho \\ v_1 \\ v_2 \\ p \end{pmatrix}.$$

Die Zustandsgleichung lautet:

$$p = (\gamma - 1)\left(E - \frac{1}{2}\rho v^2\right).$$

Im 2D-Fall ergibt sich:

$$\rho v^2 = \rho v_1^2 + \rho v_2^2.$$

Implementierung

Die Zustandsgleichung wird in der Datei `PrimCons.f90` in den Routinen `PrimCons` (primitiv auf konservativ) und `ConsPrim` (konservativ auf primitiv) umgesetzt.