Institut für Angewandte Mathematik der Universität Bonn

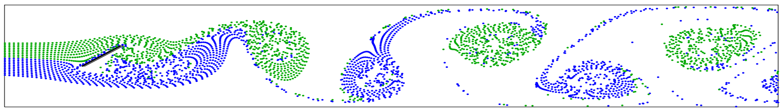
Abteilung Wissenschaftliches Rechnen und Numerische Simulation

Professor Dr. Michael Griebel

Praktikum Sommersemester 2001

Numerische Simulation und Visualisierung

- Leitung: *Prof. Dr. M. Griebel*, Zi. 604, Tel. 73 3437
- Mitarbeiter: *F. Koster*, Wegelerstr. 4, Tel. 73 3173



Blatt 2

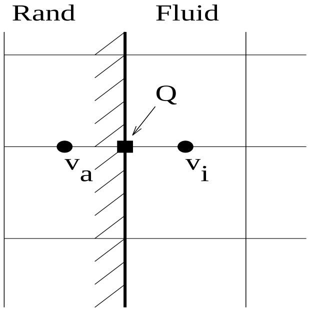
1. Behandlung weiterer Randbedingungen

Im zweiten Teil des Praktikums soll eine Reihe weiterer Beispiele inkompressibler Strömungen betrachtet werden. Dazu geben wir die Diskretisierung der restlichen 3 Randbedingungstypen an und gehen auf die Behandlung von nicht–rechteckigen Gebieten ein.
Schließlich kann jedes Problem durch eine spezielle Eingabedatei beschrieben werden. Das Programm soll dann mit dem Aufruf "`a.out` *Inputfile*" gestartet werden und in Abhängigkeit von dem Parameter *Inputfile* die richtige Eingabedatei einlesen.

Rutschbedingungen:
Bei Rutschbedingungen soll die Geschwindigkeit senkrecht zur Wand Null sein, ebenso wie die Normalableitung der Geschwindigkeit parallel zur Wand. Da in unserem rechteckigen Gebiet, das mit Hilfe des "staggered grid" diskretisiert wurde, die diskreten Geschwindigkeiten senkrecht zum Rand genau auf dem Rand liegen, können wir hier wie bei den Haftbedingungen

$$u_{0,j} = 0, \quad u_{imax,j} = 0, \quad j = 1, \ldots, jmax; \qquad v_{i,0} = 0, \quad v_{i,jmax} = 0, \quad i = 1, \ldots, imax \quad (1)$$

setzen.



Die Normalableitung der tangentialen Geschwindigkeit $\partial v/\partial n$ im Randpunkt Q kann durch den Ausdruck $(v_i - v_a)/\delta x$ diskretisiert werden, so daß aus der Forderung $\partial v/\partial n = 0$

$$v_a = v_i$$

folgt.

Wir erhalten so die weiteren Randbedingungen

$$\begin{array}{lll} v_{0,j} = v_{1,j}, & v_{imax+1,j} = v_{imax,j}, & j = 1, \ldots, jmax; \\ u_{i,0} = u_{i,1}, & u_{i,jmax+1} = u_{i,jmax}, & i = 1, \ldots, imax. \end{array} \tag{2}$$

Ausströmbedingung:
Bei der Ausströmbedingung werden am Rand die Normalableitungen beider Geschwindigkeitskomponenten auf Null gesetzt, was bedeutet, daß sich die gesamte Geschwindigkeit in der Richtung senkrecht zum Rand nicht verändern soll. Dies kann im diskreten Fall realisiert werden, indem die Geschwindigkeiten am Rand jeweils gleich den benachbarten Geschwindigkeiten im Gebiet gesetzt werden, d.h.

$$\begin{array}{lll} u_{0,j} = u_{1,j}, & u_{imax,j} = u_{imax-1,j}, & \multirow{2}{*}{$j = 1, \ldots, jmax;$} \\ v_{0,j} = v_{1,j}, & v_{imax+1,j} = v_{imax,j}, & \\[1ex] u_{i,0} = u_{i,1}, & u_{i,jmax+1} = u_{i,jmax}, & \multirow{2}{*}{$i = 1, \ldots, imax.$} \\ v_{i,0} = v_{i,1}, & v_{i,jmax} = v_{i,jmax-1}, & \end{array} \tag{3}$$

Einströmbedingung:
Bei der Einströmbedingung sind die Geschwindigkeitswerte am Rand fest vorgegeben. Da diese aber von Beispiel zu Beispiel unterschiedliche Werte annehmen sollen und auch nicht immer am ganzen Rand konstant sein werden, können wir diese Werte nicht über die Eingabedatei einlesen. Statt dessen setzen wir diese Randwerte je nach Problem in einer Prozedur "`SETSPECBCOND`", die jeweils unmittelbar nach der Prozedur "`SETBCOND`" aufgerufen werden soll.

Die Randbedingungen für p, F und G sind bei allen beschriebenen Randbedingungen die gleichen wie bei den Haftbedingungen auf Blatt 1 (Formeln (16) und (17)).

Das Programm soll nun so erweitert werden, daß wir an den 4 Seiten unseres Gebietes jeweils genau eine Randbedingung vorgeben können. Dazu führen wir die Integer–Parameter `wl` für den linken, `wr` für den rechten, `wt` für den oberen und `wb` für den unteren Rand ein, deren Belegung in der Prozedur `READ_PARAMETER` aus der Eingabedatei mit eingelesen werden soll. Dabei können die Parameter `wl, wr, wt` und `wb` die Werte

- 1 für Haftbedingungen
- 2 für Rutschbedingungen
- 3 für Ausströmbedingungen

annehmen. Bei Einströmbedingungen müssen die Geschwindigkeiten mit fest vorgegebenen Werten belegt werden (z.B. parabolisches oder konstantes Einströmprofil), so daß die Belegung der Geschwindigkeiten am Rand je nach Problem in einer eigenen Prozedur erfolgen muß und nicht durch die Vorgabe der Flags `wl,wr,wt` und `wb` allgemein beschrieben werden kann. Dazu führen wir einerseits den Parameter `char* problem` ein, der das zu berechnende

Problem kennzeichnen soll und eine Prozedur `SETSPECBCOND`, in der spezielle Randbedingungen gesetzt werden können.

Aufgabenstellung:

1. Ergänzen Sie die Prozedur `READ_PARAMETER` durch das Einlesen von `problem` und der Randwert–Flags `wl,wr,wt` und `wb`.

2. Ergänzen sie die Prozedur `SETBCOND` so, daß je nach Belegung der Randwert–Flags `wl, wr, wt` und `wb` mit Hilfe einer `if`– oder `switch`–Anweisung die richtigen Randwerte für u und v gemäß den obigen Formeln gesetzt werden.

3. Schreiben sie (in `boundary.c`) die Prozedur
`void SETSPECBCOND (REAL **U,REAL **V,int imax,int jmax,char *problem)`,
in der, abhängig vom jeweiligen Problem (gekennzeichnet durch `problem`), die Einströmbedingungen implementiert werden sollen. Bei der "Driven Cavity" muß hier die spezielle Bedingung $u = 1$ am oberen Rand aufgenommen werden. Für weitere Probleme folgt eine genaue Beschreibung im nächsten Abschnitt. Außerdem wäre es hier möglich, an einem Teil einer Kante des Gesamtgebietes andere Randbedingungen als durch `wl,wr,wt,wb` vorgegeben zu setzen (z.B. ein kleiner Ausströmbereich in einer Haftwand).

2. Behandlung allgemeiner Geometrien

Bisher haben wir die Simulation von Strömungen in rechteckigen Gebieten Ω beschrieben. Durch eine einfache Erweiterung ist es möglich, Strömungen näherungsweise auch in beliebigen zweidimensionalen Geometrien zu behandeln. Dazu betten wir das durchströmte Gebiet Ω in ein möglichst kleines rechteckiges Gebiet $\mathcal{R}$ ein, das wir, wie in Blatt 1 beschrieben, mit einem Gitter überdecken. Die Zellen von $\mathcal{R}$ werden dann eindeutig in *Fluidzellen*, (Zellen die ganz oder größtenteils in Ω liegen) und *Hinderniszellen* (Zellen die ganz oder größtenteils im Hindernis $\mathcal{H} := \mathcal{R} \setminus \Omega$ liegen) unterteilt, wobei die diskreten Navier–Stokes–Gleichungen dann nur noch in den Fluidzellen gelöst werden. Die Zellen $(i,j), i \in \{0, imax+1\}$ oder $j \in \{0, jmax+1\}$ der künstlich eingeführten Randschicht zählen auch zu den Hinderniszellen. Weiterhin bezeichnen wir diejenigen Hinderniszellen, die mit mindestens einer Fluidzelle eine Kante (*Randkante*) gemeinsam haben, als *Randzellen.* Ein Gebiet Ω mit beliebigem krummen Rand wird so durch ein Gebiet $\tilde{\Omega}$ approximiert, dessen Rand durch die Menge der Randkanten, die auf Gitterlinien verlaufen, spezifiziert wird (siehe Abbildung 1).

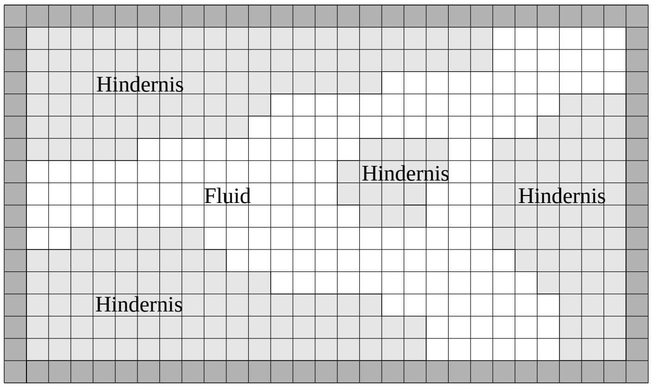


Bild 1 Einbettung eines beliebigen Gebietes in ein rechteckiges Grundgebiet

In Fluidzellen, die an Hinderniszellen angrenzen, benötigen wir dann zur Berechnung von F und G nach (9) und (10) von Blatt 1 die folgenden Randwerte:

- Werte der normalen Geschwindigkeitskomponente an Randkanten und
- Werte der tangentialen Geschwindigkeitskomponente an Kanten zwischen zwei Randzellen.

Dabei wollen wir uns hier auf die Implementierung der Haftbedingungen an den inneren Randkanten beschränken. In dem in Abbildung 2 gezeigten Beispiel sind die Randkanten mit einem Quadrat und die Kanten zwischen zwei Randzellen mit einem Kreis gekennzeichnet. Außerdem werden zur Berechnung des Drucks in den Mittelpunkten dem Rand benachbarter Fluidzellen die F– und G–Werte an Randkanten und die Druckwerte in den Mittelpunkten der Randzellen (Kreuz in Abbildung 2) gebraucht.

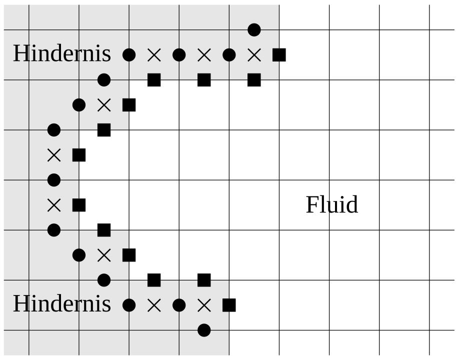


Bild 2 Benötigte Randwerte

Zur Kennzeichnung der Zellen verwenden wir ein Integer–Feld `int **FLAG`, das zunächst folgende Belegungen haben kann:

- `C_F` für eine Fluidzelle und

- `C_B` für eine Hinderniszelle.

Die Makros `C_B` und `C_F` stehen dabei für beliebig zu wählende Integerkonstanten.
Zur Bestimmung der Randwerte muß nun noch die Menge der Randzellen weiter aufgeschlüsselt werden. Dazu geben wir in dem Flagfeld an, in welcher der vier „Himmelsrichtungen“ sich Fluidzellen befinden. So bezeichne z.B. das Makro `B_O` eine Randzelle, deren rechte (östliche) Nachbarzelle zu $\tilde{\Omega}$ gehört, oder `B_SW` eine Randzelle, deren untere (südliche) und linke (westliche) Nachbarzelle zu $\tilde{\Omega}$ gehört. Dabei wollen wir annehmen, daß die Diskretisierung so fein ist, daß nur Randzellen mit einer benachbarten Fluidzelle[1] (*Kantenzellen*) oder mit zwei benachbarten, über Eck liegenden Fluidzellen (*Eckzellen*) vorkommen. Randzellen mit zwei gegenüberliegenden oder gar drei oder vier benachbarten Fluidzellen seien ausgeschlossen (*verbotene Randzellen*), da dies zu nicht mehr eindeutig bestimmbaren Randwerten führen würde.
Entsprechend den Formeln (14), (15), (16) und (17) von Blatt 1 setzen wir zum Beispiel für eine nördliche Kantenzelle (i,j) mit Flag `B_N`

$$\begin{array}{lll} v_{i,j} = 0, & u_{i-1,j} = -u_{i-1,j+1}, & u_{i,j} = -u_{i,j+1}, \\ G_{i,j} = v_{i,j}, & p_{i,j} = p_{i,j+1} & \end{array} \tag{4}$$

oder für eine westliche Kantenzelle (i,j) mit Flag `B_W`

$$\begin{array}{lll} u_{i-1,j} = 0, & v_{i,j-1} = -v_{i-1,j-1}, & v_{i,j} = -v_{i-1,j}, \\ F_{i-1,j} = u_{i-1,j} & p_{i,j} = p_{i-1,j}. & \end{array} \tag{5}$$

Die Randwerte für `B_S`– und `B_O`–Zellen werden analog gesetzt.

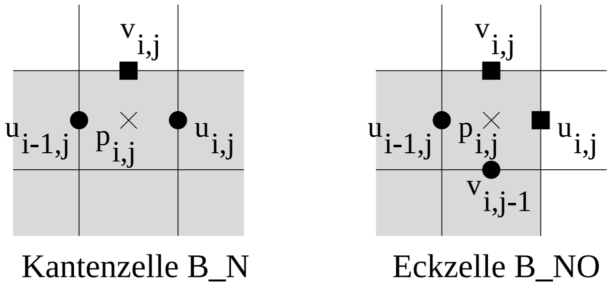


Bild 3 Randbedingungen an Hinderniszellen

Bei Eckzellen verwenden wir für die zwei Kanten, die an Fluidzellen angrenzen, die Bedingung für die normale Geschwindigkeitskomponente, während wir für die anderen beiden Kanten die Bedingung für die tangentiale Geschwindigkeitskomponente anwenden. So erhalten wir zum Beispiel für eine Zelle (i,j) mit Flag `B_NO`

$$\begin{array}{lll} u_{i,j} = 0, & u_{i-1,j} = -u_{i-1,j+1}, & F_{i,j} = u_{i,j}, \\ v_{i,j} = 0, & v_{i,j-1} = -v_{i+1,j-1}, & G_{i,j} = v_{i,j}, \\ & p_{i,j} = \frac{1}{\delta x^2 + \delta y^2}(\delta x^2 \; p_{i,j+1} + \delta y^2 \; p_{i+1,j}). & \end{array} \tag{6}$$

[1] Als *benachbart* gelten hier die Zellen, die eine ganze Kante gemeinsam haben und nicht Zellen, die nur eine Ecke gemeinsam haben.

Die Lösung der Druckgleichung (Blatt 1, Formel (11)) beschränkt sich dann jeweils auf die Fluidzellen, während die Berechnung der F– und G–Werte nach (9) und (10), Blatt 1, sowie die Korrektur der Geschwindigkeitswerte nach (7), (8), Blatt 1, nur noch an Kanten zwischen zwei Fluidzellen erfolgt.
Abschließend seien hier nochmal alle Zell- und Kantendefinitionen aufgeführt:

- Fluidzelle: Zelle, die ganz oder größtenteils in Ω liegt
- Hinderniszelle: Zelle, die ganz oder größtenteils in $\mathcal{R} \setminus \Omega$ liegt
- Randzelle: Hinderniszelle, die an eine Fluidzelle grenzt
- Kantenzelle: Randzelle, die an genau eine Fluidzelle grenzt
- Eckzelle: Randzelle, die an zwei benachbarte, über Eck liegende Fluidzellen grenzt
- verbotene Randzelle: Randzelle, die weder Eck– noch Kantenzelle ist
- Randkante: Kante zwischen Fluid– und Randzelle

Aufgabenstellung:
Das Integerfeld `int **FLAG` der Dimension `[0,imax+1]x[0,jmax+1]` wird jetzt als neues Datenfeld benötigt. Dabei bietet es sich an, die Belegungen in Binärschreibweise darzustellen. Jeder Zelle selbst und den vier benachbarten Zellen kann dann ein Bit zugeordnet werden, z.B.

Mitte	Ost	West	Süd	Nord

Jedes Bit kann dann entweder mit einer Eins belegt werden, falls die entsprechende Zelle eine Fluidzelle ist oder mit einer Null für eine Hinderniszelle. Die Zellen im Inneren der Hindernisse hätten dann das Flag 00000, Randzellen (einschließlich der verbotenen) ein Flag zwischen 00001 und 01111, Fluidzellen, die an Hinderniszellen angrenzen, ein Flag zwischen 10000 und 11110 und Zellen im Inneren des Fluidgebietes bekommen das Flag 11111. Beispielsweise entspricht dem Flag `B_SO` die Bitcodierung 01010 bzw. die Dezimalzahl 10. Mit den in der Programmiersprache C möglichen Bitoperatoren lassen sich die Fallunterscheidungen relativ elegant formulieren.
Zur Kennzeichnung des Fluidgebietes brauchen wir zwei neue Prozeduren, die in die Datei `init.c` aufgenommen werden sollten:

1.) `int **IMATRIX(nrl,nrh,ncl,nch)`
Für ein Integerfeld der Dimension `[nrl,nrh]x[ncl,nch]` wird analog zu `RMATRIX` der Speicherplatz allokiert.
Der Aufruf erfolgt im Initialisierungsteil des Hauptprogramms.

2.) `void INIT_FLAG(problem,FLAG,imax,jmax)`
Das Feld `FLAG` wird abhängig von dem Problemparameter `problem` mit den Flags `C_F` für Fluidzellen und `C_B` für Hinderniszellen belegt. Im Anschluß daran müssen in einer Schleife über alle Zellen noch die Randzellen mit den entsprechenden Flags `B_xy` versehen werden, je nachdem in welcher Richtung benachbarte Fluidzellen liegen.
Der Aufruf erfolgt im Initialisierungsteil des Hauptprogramms.

Daneben müssen die folgenden Prozeduren angepaßt werden:

1.) In SETBCOND und COMP_FG muß an den im Flagfeld definierten Randzellen die Belegung der Randwerte, wie oben beschrieben, eingeführt werden.

2.) In POISSON müssen für den Druck vor jedem Iterationsschritt neben den Randwerten in der Randschicht auch die Randwerte in den inneren Randzellen gesetzt werden.

3.) In POISSON beschränkt sich die SOR–Iteration und die Residuumsberechnung auf die Fluidzellen, während in COMP_FG die F– und G–Werte und in ADAP_UV die Geschwindigkeiten nur noch an Kanten zwischen zwei Fluidzellen berechnet werden. Außerdem darf bei der Normierung des Residuums in POISSON nur durch die Anzahl der Fluidzellen und nicht durch $i_{\max} j_{\max}$ geteilt werden.

3. Weitere Beispiele

Mit diesen Modifikationen des Strömungsprogramms können wir jetzt eine Fülle neuer Beispiele berechnen. Dazu muß jeweils eine neue Eingabedatei zur Verfügung gestellt werden. Daneben müssen die problemspezifischen Hinderniszellen in der Prozedur INIT_FLAG und die Einströmbedingungen und weitere spezielle Randbedingungen in der Prozedur SETSPECBCOND implementiert werden.

Aufgabenstellung:
Simulieren Sie die zwei folgenden Strömungen:

a) von Karman'sche Wirbelstraße

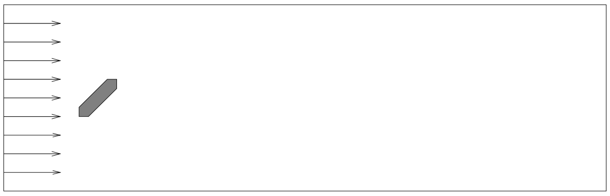


Abb. 2.1: von Karman'sche Wirbelstraße

Eine Strömung soll in einem Kanal auf einen schrägen Balken treffen. Am linken Rand strömt das Fluid mit einem konstanten Geschwindigkeitsprofil ($u = 1.0$, $v = 0.0$) ein und rechts wieder aus, während am oberen und unteren Rand Haftbedingungen verwendet werden.

Der Balken soll ein Fünftel der Kanalbreite ausfüllen und drei Zellen dick sein. Der Abstand vom linken Rand soll dabei genauso groß sein, wie der Abstand vom unteren und oberen Rand. Achten Sie darauf, daß keine *verbotenen Hinderniszellen* entstehen.
Verwenden Sie als Werte für die Parameter:

imax = 100	jmax = 20	xlength = 10	ylength = 2
delt = 0.03	t_end = 20	tau = 0.5	del_vec = 2.0
eps = 0.001	omg = 1.7	alpha = 0.9	itermax = 500
GX = 0.0	GY = 0.0	Re = 10000	
UI = 1.0	VI = 0.0	PI = 0.0	
wl = 3	wr = 3	wt = 1	wb = 1

b) Strömung über eine Stufe

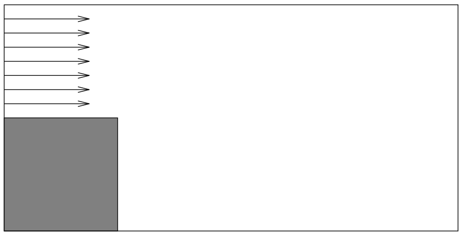


Abb. 2.2: Strömung über Stufe

In einen Kanal, der sich nach einer Seite ausweitet, strömt ein Fluid mit einer konstanten Geschwindigkeit ($u = 1.0$, $v = 0.0$) ein. Oben und unten soll das Fluid jeweils haften.

Das Hindernisgebiet soll quadratisch sein und die Höhe des Kanals zur Hälfte ausfüllen. Verwenden Sie als Werte für die Parameter:

imax = 100	jmax = 20	xlength = 10	ylength = 2
delt = 0.05	t_end = 100	tau = 0.5	del_vec = 10.0
eps = 0.001	omg = 1.7	alpha = 0.9	itermax = 500
GX = 0.0	GY = 0.0	Re = 100	
UI = 1.0	VI = 0.0	PI = 0.0	
wl = 3	wr = 3	wt = 1	wb = 1

Bei der Initialisierung der Geschwindigkeitswerte in `INIT_UVP` sollte dabei in der unteren Hälfte u auf 0 statt auf 1 gesetzt werden.

Rechnen Sie zusätzlich ein selbst gewähltes Beispiel.

4. Strömungsvisualisierung mittels Partikelverfolgung und Streichlinien

Bei Strömungen, wie etwa der oben beschriebenen Hindernisströmung, ist es von Interesse, den Strömungsverlauf anhand der Partikelbewegungen zu analysieren. Wir betrachten dazu zwei Methoden:

(A)
1.) **Bahnlinie (Partikelverfolgung):** Es werden die Positionen *eines* Partikels zu *verschiedenen*, aufeinanderfolgenden Zeitpunkten dargestellt.
2.) **Streichlinien (Streaklines):** Es werden in regelmäßigen kurzen Zeitabständen an bestimmten Stellen der Strömung Partikel “injiziert” und die zu *einem* Zeitpunkt gehörenden Positionen der *verschiedenen* Partikel dargestellt. [1]

Grundlegend für die Berechnung solcher Partikelbewegungen ist es, von der Position eines Partikels im Zeitschritt t_n auf seine Position im Zeitschritt $t_{n+1} = t_n + \delta t$ schließen zu können. Dies geschieht in zwei Schritten:

[1] Die Bezeichnung “Streichlinie” rührt daher, daß man Partikel beobachten möchte, die mit einem *Pinselstrich* auf der Flüssigkeit angebracht wurden. Eine Möglichkeit, dies im physikalischen Experiment zu bewerkstelligen, besteht darin, kleine Luftbläschen längs eines dünnen Drahtes (der die Strömung nicht beeinflußt) einzuführen.

(B)
1.) Finde die zur Position $(x^{(n)}, y^{(n)})$ im Strömungsfeld gehörenden Geschwindigkeiten $u^{(n)}, v^{(n)}$.
2.) Bestimme die neue Position des Partikels im Zeitschritt t_{n+1} mit einem Eulerschritt:
$$x^{(n+1)} = x^{(n)} + \delta t \cdot u^{(n)}, \quad y^{(n+1)} = y^{(n)} + \delta t \cdot v^{(n)} \,.$$

Hierbei ist der zweite Schritt recht einfach, der erste jedoch bedarf einiger Erläuterung:

Interpolation der Geschwindigkeiten im "staggered grid":
Da die Geschwindigkeiten u und v im "staggered grid" nur an bestimmten Stellen des Gitters vorhanden sind, muß man die gesuchten Geschwindigkeiten aus den vorhandenen annähern. Es sind also
gegeben: Position (x, y),
Geschwindigkeiten u_{ij}, v_{ij} im "staggered grid":

$$u_{i,j} = u(x_i, y_j) \text{ mit } x_i = i \cdot \delta x, \;\; y_j = (j - \frac{1}{2}) \cdot \delta y, \;\; i = 0, ..., imax, \; j = 0, ..., jmax + 1,$$
$$v_{i,j} = v(x_i, y_j) \text{ mit } x_i = (i - \frac{1}{2}) \cdot \delta x, \;\; y_j = j \cdot \delta y, \;\; i = 0, ..., imax + 1, \; j = 0, ..., jmax,$$

gesucht: Geschwindigkeiten $u(x, y)$, $v(x, y)$.

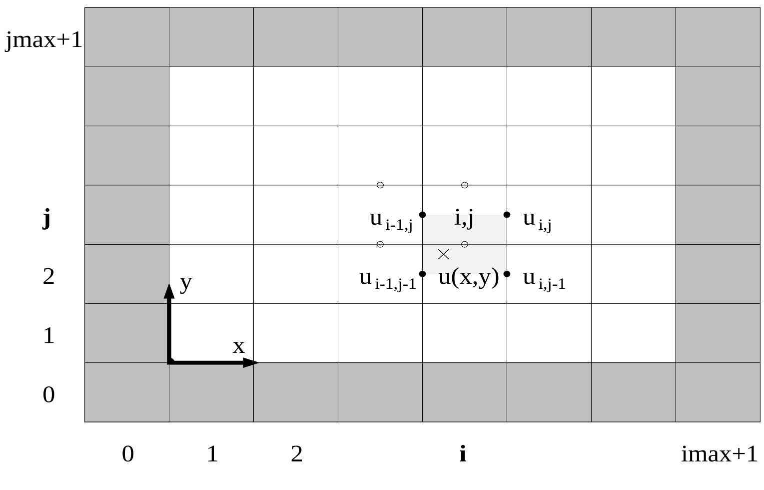


Abb. 4.1: u-*Zelle* (i, j) im "staggered grid"

i) **Berechnung von $u(x, y)$:**
Zu der gegebenen Position (x, y) bestimmt man die diese Position umgebende u–Zelle (vgl. Abbildungen 4.1 und 4.2). Dies geschieht mit Hilfe der ganzzahligen Operationen

$$i := (int)\left(\frac{x}{\delta x}\right) + 1, \quad j := (int)\left(\frac{y + \frac{\delta y}{2}}{\delta y}\right) + 1,$$

die die Numerierung der u–Zelle bestimmen und der Operationen

$$x_1 := (i-1)\cdot\delta x, \qquad y_1 := ((j-1) - \tfrac{1}{2})\cdot\delta y,$$
$$x_2 := i\cdot\delta x, \qquad y_2 := (j - \tfrac{1}{2})\cdot\delta y,$$

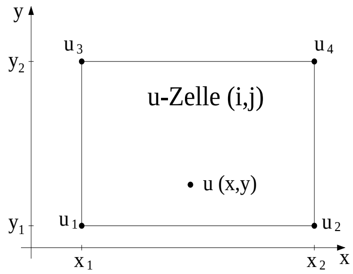


Abb. 4.2: u–Zelle (i,j)

die die Positionen der Eckpunkte dieser u–Zelle bestimmen, auf denen die vier "staggered–grid"–Geschwindigkeiten

$$u_1 := u_{i-1,j-1}, \quad u_2 := u_{i,j-1}, \quad u_3 := u_{i-1,j}, \quad u_4 := u_{i,j}$$

liegen.

Jetzt kann man die gesuchte Geschwindigkeit $u(x,y)$ in der Position (x,y) mittels bilinearer Interpolation ermitteln:

$$u(x,y) := \frac{1}{\delta x\,\delta y}\Big[(x_2-x)(y_2-y)u_1 + (x-x_1)(y_2-y)u_2 + (x_2-x)(y-y_1)u_3 + (x-x_1)(y-y_1)u_4\Big]$$

ii) **Berechnung von $v(x,y)$:**
Das Vorgehen ist ganz analog zu (i):
Die Numerierung der die Position (x,y) umgebenden v–Zelle ist

$$i := (int)\left(\frac{x + \frac{\delta x}{2}}{\delta x}\right) + 1, \quad j := (int)\left(\frac{y}{\delta y}\right) + 1,$$

die Positionen der Eckpunkte mit den v–Geschwindigkeiten des "staggered grid" sind

$$x_1 := ((i-1) - \tfrac{1}{2})\cdot\delta x, \qquad y_1 := (j-1)\cdot\delta y$$
$$x_2 := (i - \tfrac{1}{2})\cdot\delta x, \qquad y_2 := j\cdot\delta y,$$

und die Interpolation lautet

$$v(x,y) := \frac{1}{\delta x\,\delta y}\Big[(x_2-x)(y_2-y)v_1 + (x-x_1)(y_2-y)v_2 + (x_2-x)(y-y_1)v_3 + (x-x_1)(y-y_1)v_4\Big]$$

Aufgabenstellung:
Schreiben Sie zwei Prozeduren `PARTICLE_TRACING` und `STREAKLINES`, die innerhalb der Zeitschleife des Hauptprogramms regelmäßig aufgerufen werden und gemäß (A) und (B) die Bahnlinien bzw. Streaklines zu vorgegebenen Positionen berechnen und in ein File ablegen. Hierbei sollen die Partikel in der folgenden Datenstruktur gehalten werden:

```
struct particle{
       REAL x, y;
       struct particle *next;
}
struct particleline{
       int length;
       struct particle *Particles;
}
struct particleline *Particlelines;
```

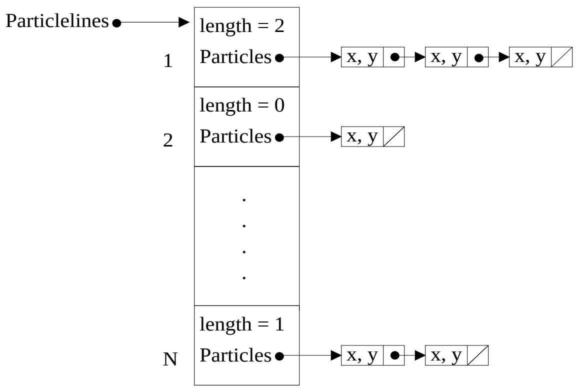


Abb. 4.3: Datenstruktur `Particlelines` mit Beispielen für deren Längen

Insbesondere soll gelten:

- `PARTICLE_TRACING`: Das erste Element einer `particleline` von `Particlelines` ist bloßes Hilfsfeld, damit das Aushängen von Partikeln, die den betrachteten Strömungsbereich verlassen, *ohne* Vorgänger–Zeiger funktioniert.

- `STREAKLINES`: Erstes Element einer `particleline` von `Particlelines` enthält Partikelposition, an denen zu gegebener Zeit Partikel injiziert werden sollen.

Die benötigten Prozeduren können in ein neues Modul `visual.c` aufgenommen werden, in das auch die schon bekannte Prozedur `OUTPUTVEC` gehört. Im einzelnen sind zu programmieren:

1. `struct particleline *SET_PARTICLES(int N,`
   `REAL x1,REAL y1,REAL x2,REAL y2,)`
   Der benötigte Speicherplatz für die Partikel wird dynamisch belegt und die Partikelpositionen in die jeweilige `particleline` eingetragen. Jede `particleline` hat dabei zunächst die Länge 1 (siehe Abbildung 4.3). Im Hilfsfeld und im ersten Partikel stehen die gleichen Koordinaten. Zurückgeliefert wird ein Pointer auf den allokierten Bereich. Die Partikel sollen entlang der Geraden mit den Endpunkten (`x1`,`y1`) und (`x2`,`y2`) injiziert werden, wobei der Parameter `N` die Anzahl der äquidistant auf der Geraden

verteilten Partikel bestimmt.
Beachte: Setzt man `Particlelines=SET_PARTICLES(N,x1,y1,x2,y2)` für

- `PARTICLE_TRACING`, so ist das erste Element jeder `particleline` bloßes Hilfsfeld,
- `STREAKLINES`, so enthält das erste Element jeder `particleline` den Ort des jeweils zu injizierenden Partikels, alle folgenden Elemente die Positionen der tatsächlich zu verfolgenden Partikel. Im Gegensatz zu `PARTICLE_TRACING` werden im Verlauf der Rechnung weitere Partikel in die Liste eingefügt.

Die Parameter `N,x1,y1,x2,y2` sollen in der Prozedur `READ_PARAMETER` aus der Eingabedatei eingelesen werden. Der Aufruf erfolgt in main.

2. `void ADVANCE_PARTICLES(imax,jmax,delx,dely,delt,U,V,`
   `int N,struct particleline *Partlines)`
   Die in `Partlines` (Array von `particleline`) gehaltenen Partikelpositionen des Zeitschrittes t_n werden gemäß (B) in die Partikelpositionen des Zeitschrittes t_{n+1} „fortbewegt". Partikel, die dabei das Grundgebiet $\mathcal{R} = [0, i_{\max}\,\delta x] \times [0, j_{\max}\,\delta y]$ verlassen, oder in Hinderniszellen laufen, werden aus der Liste ausgehängt und der von diesen Partikeln belegte Speicherplatz wird mittels `free` wieder freigegeben. [2]
   Der Aufruf erfolgt in `PARTICLE_TRACING` und `STREAKLINES`.

3. `INJECT_PARTICLES(int N,struct particleline *Partlines)`
   Die in `SET_PARTICLES` spezifizierten Partikel werden entlang der spezifizierten Geraden injiziert, indem in jede `particleline` von `Partlines` das zu injizierende Partikel mit der im Hilfsfeld spezifizierten Position zusätzlich am Listenanfang eingehängt wird. Außerdem muß der benötigte Speicherplatz für das neue Partikel allokiert werden.

   Der Aufruf erfolgt in `STREAKLINES`.

4. `void WRITE_PARTICLES(char* filename,int N, REAL xlength,REAL ylength,`
   `struct particleline *Partlines)`
   Die Partikelpositionen von `Partlines` werden in die durch den Übergabeparameter `filename` spezifizierte Datei geschrieben.

   Die erzeugte Datei soll das folgende Format haben:

[2] Wenn die CFL–Bedingung erfüllt ist, so können Partikel an Rändern mit Rutsch– oder Haftbedingungen lediglich unkorrekterweise in Eckzellen verschwinden, nicht aber in Kantenzellen. Will man dies verhindern, so muß der Interpolationsalgorithmus in der Nähe von Kantenzellen geeignet modifiziert werden, indem man die Zusatzinformation verwendet, daß die normale Geschwindigkeitskomponente nicht nur an den Mittelpunkten der Randkanten null ist, sondern auf der gesamten Kante.

$1 \times$ `int`	$\sum_{i=1}^{N}$ `Partlines[i].length`
$2 \times$ `double`	`xlength, ylength`
$1 \times$ `double`	`Partlines[1].Particles-->next-->x`
$1 \times$ `double`	`Partlines[1].Particles-->next-->y`
	.
	.
$1 \times$ `double`	`Partlines[N].Particles-->next-->x`
$1 \times$ `double`	`Partlines[N].Particles-->next-->y`
	.
	.

Alle Werte werden *binär* mit der Funktion `fwrite(...)` geschrieben.

In MatLab (schneller zu starten mit matlab -nojvm) selbst können die Daten mittels z.B.

```
>>part=ReadParticles('/mount/people/cfdprakt/Nast2d/bin/test-part') ;
```

eingelesen werden und mittels

```
>>spy(part');
```

dargestellt. Dabei werden die Partikel bzw. deren Positionen als Nicht-Null Eintrage in der dünnbesetzten Matrix `part` representiert. Die Größe der Punkte mit der `spy` die Nicht-Null Eintrage darstellt, kann mittels `spy(part',<groesse>)` variiert werden. Die Achsen-Markierungen können mit

```
hold on
set(gca,'XTick',[] , 'YTick',[]) ;
set(get(gca,'Xlabel'),'Visible','off') ;
hold off
```

abgeschaltet werden.

5. `void PARTICLE_TRACING(char* outputfile,REAL delt_trace,`
   `t,imax,jmax,delx,dely,delt,U,V,`
   `int N,struct particleline *Partlines)`

   Berechnet mit Hilfe von `ADVANCE_PARTICLES` zu jedem Zeitschritt die aktuellen Positionen der zu verfolgenden Partikel und überschreibt die alten Positionen in `Partlines`. In den Zeitabständen `delt_trace` [3] werden die aktuellen Positionen der Partikel mittels `WRITE_PARTICLES` in eine Datei geschrieben, deren Name sich aus dem Übergabeparameter `outputfile` und dem Zeitschritt ergibt. Der Aufruf erfolgt in der Zeitschleife

[3] Da der jeweilige Zeitschritt `delt` aus einer Zeitschrittweitensteuerung ermittelt wird, können die Werte für `delt_trace, delt_inject, delt_streak` nicht a priori so bestimmt werden, daß die jeweils zugehörige „Aktionszeit" in der Zeitschleife genau erreicht wird. In der Regel ist jedoch `delt` klein gegen `delt_trace, delt_inject, delt_streak`, so daß die jeweilige Aktion beim ersten Überschreiten der zugehörigen Aktionszeit ausgeführt werden kann.

von main.

6. `void STREAKLINES(char* outputfile,REAL delt_inject,`
   `REAL delt_streak,t,imax,jmax,delx,dely,delt,`
   `U, V, int N, struct particleline *Partlines,)`
   Jeden Zeitschritt werden mit Hilfe von `ADVANCE_PARTICLES` die aktuellen Positionen der Partikel berechnet und die alten Positionen in `Partlines` überschrieben. In den Zeitabständen `delt_inject` werden mittels der Prozedur `INJECT_PARTICLES` Partikel injiziert und in den Zeitabständen `delt_streak` die aktuellen Partikelpositionen mittels `WRITE_PARTICLES` ausgegeben. Dabei sind die Daten, die zu verschiedenen Zeitpunkten geschrieben wurden, auch zu verschiedenen Zeitpunkten zu visualisieren.

7. Die verschiedenen Zeitabstände `delt_trace, delt_inject, delt_streak` sollen mittels `READ_PARAMETER` eingelesen werden.

8. Schreiben Sie ein geeignetes MatLab-Programm zur Visualisierung der Bahn– und Streichlinien zu aufeinander folgenden Zeitpunkten.

Wenden Sie die Visualisierung mittels Bahnlinien auf die *backward facing step* an und die Visualisierung mittels Streichlinien auf die *Hindernisströmung.*

5. Stromfunktion ψ und Wirbelstärke ζ

Wie schon auf dem ersten Aufgabenblatt verschämt in einer Fußnote erwähnt wurde, kann man die 2-dimensionalen Navier–Stokes–Gleichungen nicht nur in den sogenannten "primitiven" Variablen u, v, p formulieren (vgl. Blatt 1, (1)-(3)), sondern dazu auch die Größen ψ und ζ, die Stromfunktion und die Wirbelstärke verwenden. Darauf wird hier aber nicht näher eingegangen. Vielmehr sollen diese Größen definiert und anschaulich interpretiert werden, so daß weitere Hilfsmittel gewonnen werden, um die berechnete Strömung analysieren zu können.

Definition:

Sind u und v die Geschwindigkeiten des Stömungsfeldes, so definiert man die *Stromfunktion* $\psi(x,y)$ mittels

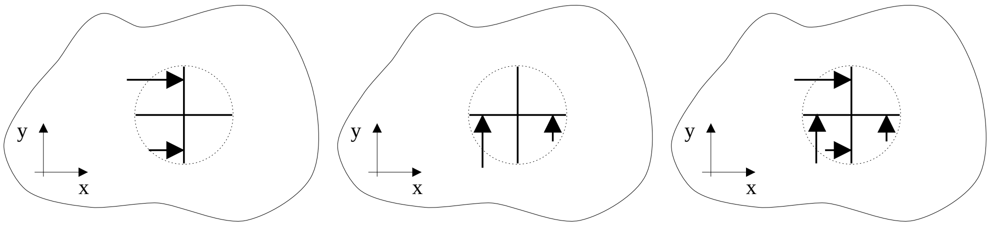
$$\frac{\partial\psi(x,y)}{\partial x} := -v, \qquad \frac{\partial\psi(x,y)}{\partial y} := u, \tag{7}$$

und die *Wirbelstärke* $\zeta(x,y)$ als

$$\zeta(x,y) := \frac{\partial u}{\partial y} - \frac{\partial v}{\partial x}. \tag{8}$$

Interpretation:

- **Wirbelstärke ζ:**
  ζ mißt – wie der Name verrät – die Stärke der Wirbel im Geschwindigkeitsfeld. Dazu sei die Abb. 5.1 betrachtet:



(a) (b) (c)

Abb. 5.1: Gedankenexperiment zur Wirbelstärke ζ

  Hier wird in einem Gedankenexperiment ein Rad in ein Geschwindigkeitsfeld hineingelegt. Im Fall (a) *steigt* die u–Geschwindigkeit in y–Richtung an: $\frac{\partial u}{\partial y} > 0$, und es wird eine Drehung des Rades im Uhrzeigersinn verursacht. Im Fall (b) wird ebenfalls eine Drehung im Uhrzeigersinn verursacht, jedoch *fällt* hier die v–Geschwindigkeit in x–Richtung: $\frac{\partial v}{\partial x} < 0$.
  Um nun das Gesamtmaß der "Drehwirkung" zu erfassen, werden die Fälle (a) und (b) im Fall (c) zusammengenommen: Wirbelstärke $\zeta = \frac{\partial u}{\partial y} - \frac{\partial v}{\partial x}$.

- **Stromfunktion ψ:**
  Zunächst zur Wohldefiniertheit der Stromfunktion:
  Damit eine Funktion in 2 Unabhängigen x und y mittels ihrer partiellen Ableitungen wie in (7) sinnvoll definiert werden kann, muß eine hinreichende Bedingung erfüllt sein, die sogenannte *Integrabilitätsbedingung* (Vertauschbarkeit der Ableitungen):

$$\frac{\partial^2\psi}{\partial x\partial y} = \frac{\partial^2\psi}{\partial y\partial x}.$$

  Im Fall der Stromfunktion sorgt die Kontinuitätsgleichung (Blatt 1, (3)) für das Einhalten dieser Bedingung, denn:

$$\frac{\partial}{\partial x}u + \frac{\partial}{\partial y}v = 0 \iff \frac{\partial}{\partial x}\left(\frac{\partial\psi}{\partial y}\right) + \frac{\partial}{\partial y}\left(-\frac{\partial\psi}{\partial x}\right) = 0$$

$$\iff \frac{\partial^2\psi}{\partial x\partial y} - \frac{\partial^2\psi}{\partial y\partial x} = 0.$$

Für eine anschauliche Interpretation der Stromfunktion, muß zunächst der Begriff der *Stromlinie* eingeführt werden.

Definition:

Eine *Stromlinie* ist eine Kurve, die zu einer festen Zeit t in jedem ihrer Punkte (x, y) eine zum zugehörigen Geschwindigkeitsvektor $\binom{u}{v}$ parallele Tangente besitzt.

Für die Stromfunktion ψ gilt nun, daß alle ihre Höhenlinien $\psi(x, y) = const$ Stromlinien sind.

Desweiteren gilt, daß die Durchflußmasse m pro Zeiteinheit zwischen zwei Stromlinien $\psi(x, y) = \psi_a$, $\psi(x, y) = \psi_b$ mit

$$m = \psi_b - \psi_a$$

konstant ist.

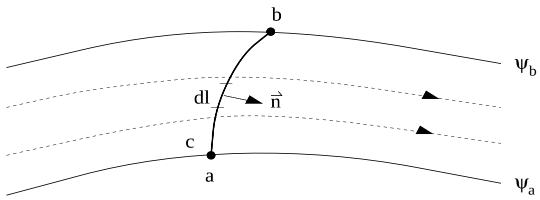


Abb. 5.2: Massenfluß zwischen Stromlinien

Bemerkung für Interessierte:

- Die Höhenlinien einer Funktion $f(x, y)$ in zwei Unabhängigen sind definiert als die Mengen

$$N_c := \{(x, y) : \ f(x, y) = c\}, \ c \in \mathbb{R}.$$

Ist f hinreichend "zahm", läßt sich N_c als C^1–Kurve parametrisieren: $\varphi_c(s) = \binom{x_c(s)}{y_c(s)} \in N_c$, und es gilt:

$$f(\varphi_c(s)) = f(x_c(s), y_c(s)) = c\,.$$

Hieraus folgt nun leicht, daß der Gradient grad$f(\varphi_c(s))$ senkrecht auf den Tangenten $\dot{\varphi}_c(s)$ der Höhenlinie $\varphi_c(s)$ von f steht:

$$0 = \frac{df(\varphi_c(s))}{ds} = \frac{\partial f}{\partial x}\frac{dx_c(s)}{ds} + \frac{\partial f}{\partial y}\frac{dy_c(s)}{ds} = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right) \cdot \begin{pmatrix}\dot{x}_c(s)\\ \dot{y}_c(s)\end{pmatrix} = \mathrm{grad}f \cdot \dot{\varphi}_c(s)\,.$$

Dies bezogen auf die Stromfunktion $\psi(x, y)$ und die Stromlinien mit den Tangentenrichtungen $\binom{u}{v}$ ergibt die Behauptung, daß die Höhenlinien der Stromfunktion die Stromlinien sind, denn

$$\mathrm{grad}\psi \cdot \begin{pmatrix}u\\ v\end{pmatrix} = \left(\frac{\partial\psi}{\partial x}, \frac{\partial\psi}{\partial y}\right) \cdot \begin{pmatrix}u\\ v\end{pmatrix} = (-v, u) \cdot \begin{pmatrix}u\\ v\end{pmatrix} = -uv + uv = 0\,,$$

und die Tangentenrichtungen $\dot{\varphi}_c(s)$ längs der Höhenlinien parallel zum Geschwindigkeitsvektor $\binom{u}{v}$ verlaufen.

- Die Durchflußmasse m pro Zeiteinheit zwischen zwei Stromlinien $\psi = \psi_a$, $\psi = \psi_b$ ist die Masse, die die Kurve c pro Zeiteinheit passiert. Für die Masse m gilt: $m = \int_\Omega \varrho \, dxdy$, wobei ϱ die Dichte des Fluides ist und für die Masse, die die Kurve c passiert, gilt: $m = \int_c \varrho \binom{u}{v} \cdot \vec{n} \, dl$ (vgl. Abbildung 5.2).
  Bei inkompressiblen Strömungen ist $\varrho = \varrho_0 = const$, und nach Normierung $\varrho_0 = 1$ ergibt sich:

  $$m = \int_c \varrho_0 \binom{u}{v} \cdot \vec{n} \, dl = \int_c \binom{u}{v} \cdot \vec{n} \, dl = \int_a^b \binom{u}{v} \cdot \binom{dy}{-dx} = \int_a^b u dy - v dx = \int_a^b \frac{\partial \psi}{\partial y} dy + \frac{\partial \psi}{\partial x} dx = \int_a^b d\psi = \psi_b - \psi_a \, .$$

Aufgabenstellung:
Zu berechnen sind die diskretisierten Größen $\psi_{i,j}$ und $\zeta_{i,j}$. Um jedoch unnötig umständliche Mittelungen im "staggered grid" zu vermeiden, sollen diese Größen nicht in den jeweiligen Zellmittelpunkten, sondern in den jeweiligen *rechten, oberen Ecken der Zelle* (i, j) berechnet werden. Hierbei sind die Indizes der Stromfunktion $\psi_{i,j}$ $i = 0, .., i_{\max}$, $j = 0, .., j_{\max}$ und der Wirbelfunktion $\zeta_{i,j}$ $i = 1, .., i_{\max} - 1$, $j = 1, .., j_{\max} - 1$.
Es ist folgendes zu programmieren:

1. `void COMP_PSI_ZETA(U,V,REAL **PSI,REAL **ZETA,`
   `int **FLAG,imax,jmax,delx,dely)`
   Berechnet zum einen die diskrete Stromfunktion $\psi_{i,j}$ über die Definitionsgleichung

   $$\frac{\partial \psi}{\partial y} = u \quad \text{mit Hilfe der Diskretisierung} \quad \left[\frac{\partial \psi}{\partial y}\right]_{i,j} = \frac{\psi_{i,j} - \psi_{i,j-1}}{\delta y}$$

   und der Festlegung $\psi_{i,0} := 0$ gemäß der Formel

   $$\psi_{i,j} = \psi_{i,j-1} + u_{i,j} \, \delta y \, , \quad i = 0, ..., i_{\max} \, , \quad j = 1, ..., j_{\max} \, ,$$

   und zum anderen die diskrete Wirbelstärke $\zeta_{i,j}$ über die Definitionsgleichung (8) gemäß der Formel

   $$\zeta_{i,j} := \frac{u_{i,j+1} - u_{i,j}}{\delta y} - \frac{v_{i+1,j} - v_{i,j}}{\delta x} \, .$$

   Es ist jeweils die durch das Flagfeld gegebene Hindernisstruktur zu berücksichtigen. So kann man $\zeta_{i,j}$ in und am Rand von Hinderniszellen auf Null setzen und für $\psi_{i,j}$ darf in den Hinderniszellen nichts aufaddiert werden: $\psi_{i,j} = \psi_{i,j-1}$.

Die neue Prozedur `void COMP_PSI_ZETA` kann im Modul `visual.c` aufgenommen werden. Visualisieren Sie mittels MatLab die Höhenlinien der Stromfunktion (Stromlinien) und der Wirbelstärke für die Beispiele der *driven cavity, Hindernisströmung* und *backward facing step.* Dazu kann die Funktion `contour(.)` benutzt werden. Die Parameter dieser Funktion können Sie der Online-Dokumentation entnehmen.