Institut für Angewandte Mathematik der Universität Bonn

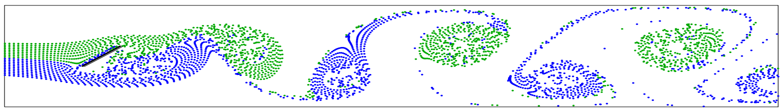
Abteilung Wissenschaftliches Rechnen und Numerische Simulation

Professor Dr. Michael Griebel

Praktikum Sommersemester 2001

Numerische Simulation und Visualisierung

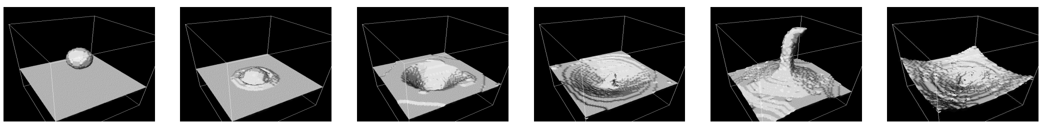
- Leitung: *Prof. Dr. M. Griebel*, Zi. 604, Tel. 73 3437
- Mitarbeiter: *F. Koster*, Wegelerstr. 4, Tel. 73 3173



Blatt 1

1. Einführung

In dem noch jungen Forschungsgebiet des wissenschaftlichen Rechnens werden der technische Sachverstand der Ingenieure, die numerischen Verfahren der Mathematiker und die modernen Methoden und Rechner der Informatiker gleichgewichtig interdisziplinär eingesetzt. Die mathematische Vorausberechnung technischer Prozesse, kurz numerische Simulation genannt, ist dabei ein wichtiges Ziel.
Durch die numerische Simulation kann beispielweise der Aufbau kostspieliger Versuchsaufbauten vermieden werden. Darüberhinaus lassen sich in vielen Fällen Aussagen treffen, bei denen dies auf Grund technischer Unzulänglichkeiten sonst nicht möglich wäre, oder ein praktisches Experiment sich von vornherein verbietet.
Ein wichtiges Beispiel für den Einsatz der numerischen Simulation ist die Berechnung von Strömungsvorgängen wie die Umströmung eines Flugzeugflügels, der Luftwiderstand eines Automobils, Verbrennungsvorgänge in Kraftwerken, sowie Schmelzprozesse und Kristallwachstum bei der Halbleiterfertigung. Aber auch bei der Wettervorhersage, in der Geologie, der Bioinformatik, der Mechanik (Elastizitätstheorie) und in vielen anderen Gebieten der Physik und der Chemie kommen Simulationsmethoden zum Einsatz.
Das folgende Vorgehen ist dabei charakteristisch: Aus der Beobachtung der Realität wird ein mathematisches Modell entwickelt, das nach geeigneter Diskretisierung näherungsweise gelöst wird. Mittels geeigneter Visualisierungsmethoden können die Ergebnisse interpretiert werden. Gegebenenfalls müssen dann die numerischen Lösungsverfahren oder das Modell nachgebessert werden (vergleiche hierzu auch Abbildung 1).



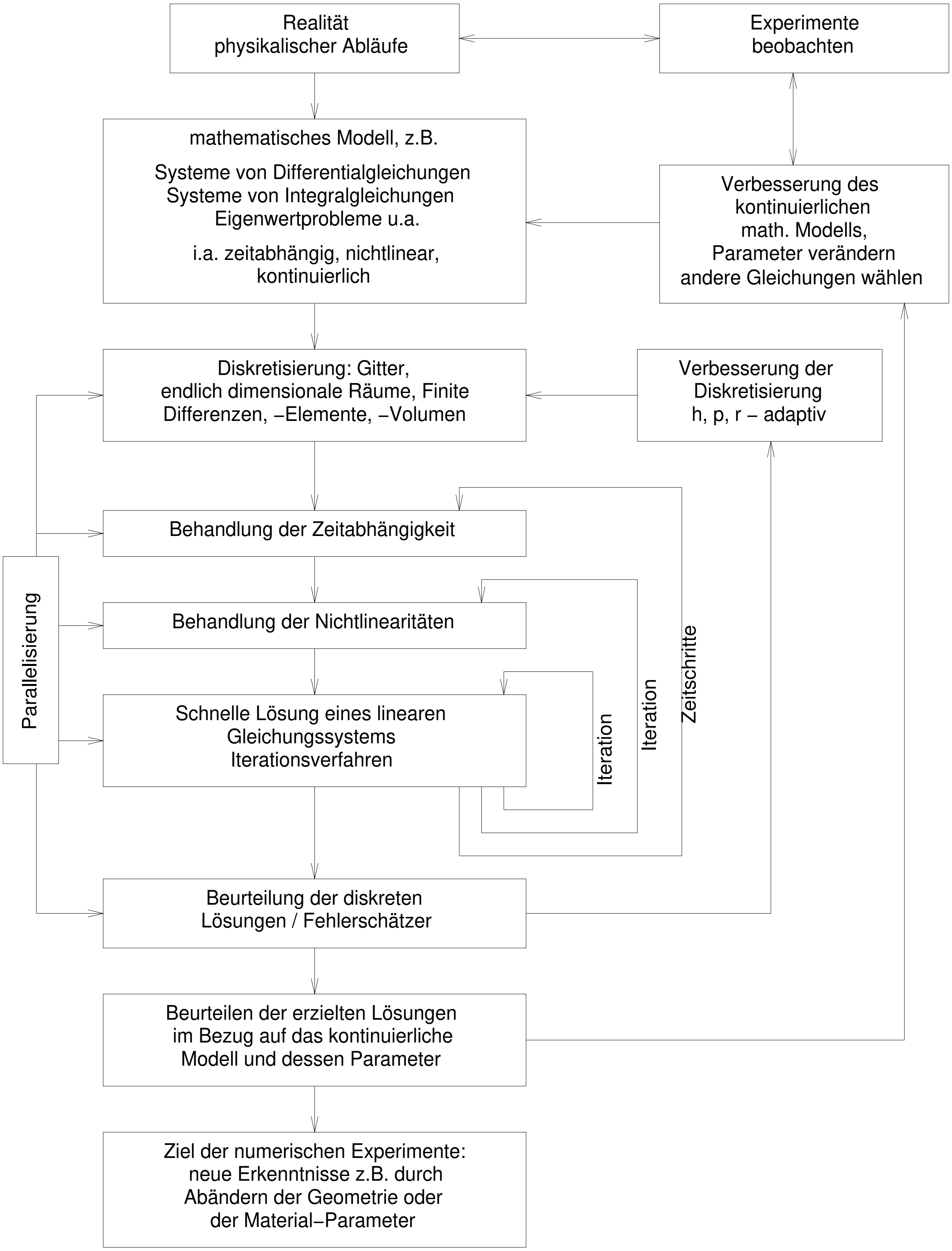


Abb. 1: Typische Vorgehensweise in einem Teilgebiet des Wissenschaftlichen Rechnens

2. Die Problemstellung: Inkompressible viskose Strömungen

Im Praktikum wollen wir diese Schritte an einem konkreten Beispiel aus der Strömungsmechanik erlernen. Wir betrachten dazu die Behandlung viskoser, instationärer, laminarer Strömungen.
Dabei beschränken wir uns auf *inkompressible* Medien, wie beispielsweise Wasser, die, wie der Name sagt, nicht komprimierbar sind. Luft wäre hingegen ein Beispiel für ein kompressibles Medium, bei dem ein und dieselbe Masse unterschiedliche Volumina ausfüllen kann. Die *Viskosität* beschreibt die Zähigkeit des Mediums. So besitzt Luft eine sehr geringe, Wasser eine etwas größere und Honig eine hohe Viskosität. Bei nicht zu hohen Geschwindigkeiten und nicht zu geringen Viskositäten sind Strömungen relativ regelmäßig und wenig mischungsintensiv, d.h. *laminar* und nicht turbulent. *Instationär* bedeutet schließlich, daß sich die Strömung im Gegensatz zu stationären Strömungen mit der Zeit verändern kann.
Beispiele aus der Praxis sind in den Abbildungen 2 und 3 zu sehen.

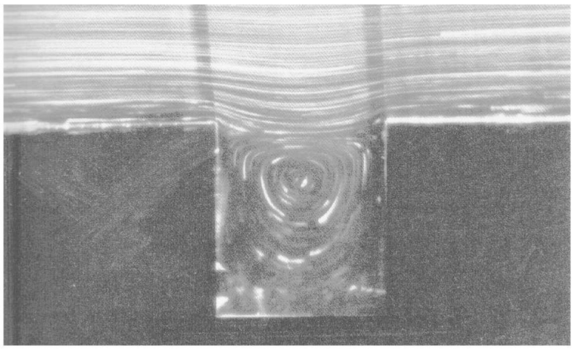


Abb. 2: Strömung über einen Hohlraum

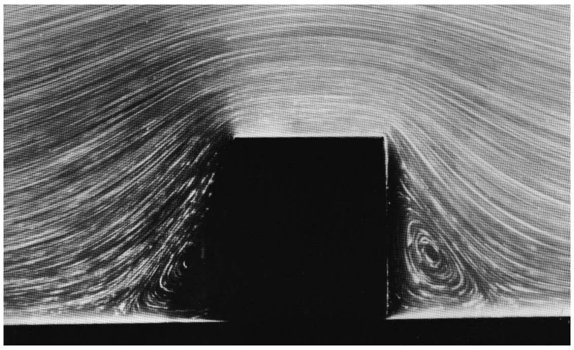


Abb. 3: Strömung über eine Stufe

Bilder und MPEG-Movies von weiteren numerisch simulierten Beispielen findet man in

http://wwwwissrech.iam.uni-bonn.de/research/films/praktikum_d.html
http://gaia.iwr.uni-heidelberg.de/~featflow

3. Das mathematische Modell: Die Navier–Stokes–Gleichungen

Instationäre, inkompressible, viskose, laminare Strömungen lassen sich durch die *Navier–Stokes–Gleichungen* beschreiben. Der Einfachheit halber beschränken wir uns dabei auf den zweidimensionalen Fall und legen ein kartesisches Koordinatensystem zu Grunde.
Dann erhalten wir ein System partieller Differentialgleichungen, bestehend aus den zwei *Impulsgleichungen*

$$\frac{\partial u}{\partial t} + \frac{\partial p}{\partial x} = \frac{1}{Re}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) - \frac{\partial (u^2)}{\partial x} - \frac{\partial (uv)}{\partial y} + g_x, \tag{1}$$

$$\frac{\partial v}{\partial t} + \frac{\partial p}{\partial y} = \frac{1}{Re}\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) - \frac{\partial (uv)}{\partial x} - \frac{\partial (v^2)}{\partial y} + g_y \tag{2}$$

und der *Kontinuitätsgleichung*

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0, \tag{3}$$

das zunächst auf dem rechteckigen Gebiet $\Omega := [0,a] \times [0,b] \subset \mathbb{R}^2$ in dem Zeitintervall $[0, t_{end}]$ gelöst werden soll. [1]

Dabei sind die drei gesuchten Größen

- $u : \Omega \times [0, t_{end}] \to \mathbb{R}$ die Geschwindigkeit des Fluids in x–Richtung,
- $v : \Omega \times [0, t_{end}] \to \mathbb{R}$ die Geschwindigkeit des Fluids in y–Richtung,
- $p : \Omega \times [0, t_{end}] \to \mathbb{R}$ der Druck.

Die reelle Zahl Re ist die *Reynoldszahl*, die die Zähigkeit des Fluids angibt. Je kleiner Re ist, umso zäher ist das Fluid.

$g_x : \Omega \times [0, t_{end}] \to \mathbb{R}$ und $g_y : \Omega \times [0, t_{end}] \to \mathbb{R}$ sind äußere Kräfte, etwa die Erdanziehung oder andere Beschleunigungen, denen das System unterworfen ist. In der Literatur findet man häufig nicht die obige, komponentenweise Notation sondern die kompaktere Schreibweise

$$\begin{aligned} \frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot (\mathbf{u} \otimes \mathbf{u}) &= \mathbf{g} - \nabla p + \nu \Delta \mathbf{u} \ , \\ \nabla \cdot \mathbf{u} &= 0 \ . \end{aligned}$$

Dabei ist $\mathbf{u} = (u, v)^T$ bzw. $(u, v, w)^T$ in 3D, $\mathbf{g} = (g_x, g_y\{, g_z\})^T$ und $\mathbf{u} \otimes \mathbf{u}$ die 2×2 (3×3) Matrix $\mathbf{u} \cdot \mathbf{u}^T$. Die Divergenz $\nabla \cdot$ einer Matrix ist dabei wie üblich der Vektor der Divergenzen der Spalten.

Zu Beginn ($t = 0$) seien Anfangsbedingungen $u = u_0(x, y)$ und $v = v_0(x, y)$ vorgegeben, die (3) erfüllen. Zusätzlich sind zu allen Zeitpunkten Bedingungen an den 4 Gebietsrändern erforderlich, so daß wir insgesamt ein *Anfangs–Randwertproblem* erhalten.

Für die Formulierung der Randbedingungen bezeichne w_1 die Geschwindigkeitskomponente senkrecht zum Rand (in Normalenrichtung), w_2 die Geschwindigkeitskomponente parallel zum Rand (in Tangentialrichtung) und $\partial w_1 / \partial n$ bzw. $\partial w_2 / \partial n$ die Ableitung in Normalenrichtung.

Am senkrechten Rand gilt also

$$w_1 = u, \qquad w_2 = v, \qquad \frac{\partial w_1}{\partial n} = \frac{\partial u}{\partial x}, \qquad \frac{\partial w_2}{\partial n} = \frac{\partial v}{\partial x},$$

und am waagrechten Rand gilt

$$w_1 = v, \qquad w_2 = u, \qquad \frac{\partial w_1}{\partial n} = \frac{\partial v}{\partial y}, \qquad \frac{\partial w_2}{\partial n} = \frac{\partial u}{\partial y}.$$

Für Punkte (x, y) des festen Randes $\Gamma := \partial\Omega$ sind folgende Randbedingungen möglich:

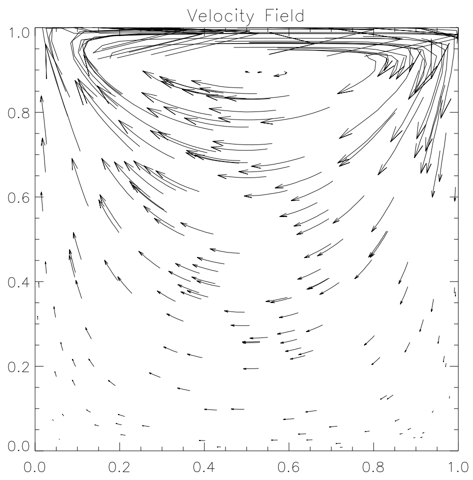
[1] Neben der hier verwendeten Formulierung mit den Unbekannten (u, v, p) existiert auch eine Formulierung mit der Stromfunktion ψ und der Wirbelfunktion ω als Unbekannte (siehe auch Blatt 2).

1. Haftbedingung (no–slip): $w_1(x,y) = 0$, $w_2(x,y) = 0$.
   (Fluid haftet am Rand)
2. Rutschbedingung (free–slip): $w_1(x,y) = 0$, $\partial w_2(x,y)/\partial n = 0$.
   (keine Reibung am Rand)
3. Einströmbedingung (inflow): $w_1(x,y) = w_1^0$, $w_2(x,y) = w_2^0$.
   w_1^0, w_2^0 gegeben
4. Ausströmbedingung (outflow): $\partial w_1(x,y)/\partial n = 0$, $\partial w_2(x,y)/\partial n = 0$.

Sind an allen Rändern die Geschwindigkeiten selbst und nicht deren Normalenableitungen vorgegeben, so muß zusätzlich das Randintegral über die Geschwindigkeiten senkrecht zum Rand Null sein, d.h.

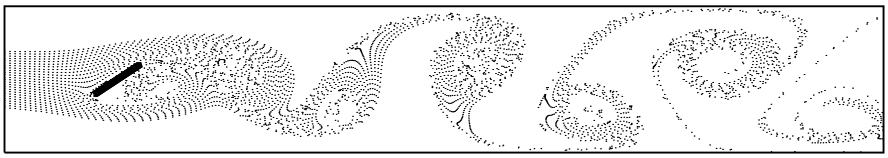
$$\int_\Gamma \begin{pmatrix} u \\ v \end{pmatrix} \cdot n \, ds = 0$$

Beispiele für Strömungen mit unterschiedlichen Randbedingungen sind etwa:



4 Wände mit Haftbedingungen
Die obere Wand bewegt sich mit konstanter Geschwindigkeit nach rechts,
d.h. $w_2 = w_2^0 = const.$ am oberen Rand, im Gegensatz zu $w_2 = 0$ bei gewöhnlichen Haftbedingungen (siehe auch Abschnitt 9.)

Abb. 4: Driven Cavity (Geschwindigkeitsfeld)



links: Einströmung (inflow), rechts: Ausströmung (outflow)
oben und unten starre Wände (no-slip)

Abb. 5: Strömung über einen schrägen Balken
(Streichlinien)

Mit physikalisch richtigen Randbedingungen erhalten wir insgesamt ein „gut–gestelltes“Problem, das eine eindeutige Lösung besitzt.
Diese Lösung ist jedoch im allgemeinen Fall nicht analytisch berechenbar, d.h. sie kann nicht in einer geschlossenen Formel angegeben werden. Sie ist meist nur *numerisch* approximierbar.

4. Die Diskretisierung

Mit dem Begriff *Diskretisierung* beschreibt man in der Numerik den Übergang von einem kontinuierlichen Problem zu einem Problem, das nur in endlich vielen Punkten betrachtet wird. Im einfachsten Fall wird das Gebiet Ω durch ein Gitter überdeckt, und das zu lösende Problem wird statt auf dem ganzen Gebiet jetzt nur noch in den Kreuzungspunkten der Gitterlinen betrachtet. Das Gitter habe in x–Richtung $imax$ und in y–Richtung $jmax$ Zellen der gleichen Größe, so daß die Gitterlinien die Abstände $\delta x := a/imax$ und $\delta y := b/jmax$ haben. Die Zelle mit dem Index (i, j) entspricht dem Gebiet $[(i-1)\,\delta x, i\,\delta x] \times [(j-1)\,\delta y, j\,\delta y]$. Die Differentialoperatoren lassen sich dann durch Differenzensterne ersetzen. Gemäß der Definition der Ableitung

$$\frac{df}{dx} := \lim_{\delta x \to 0} \frac{f(x+\delta x) - f(x)}{\delta x}$$

wird so im Eindimensionalen der kontinuierliche Differentialoperator df/dx durch den Differenzenoperator $\delta f/\delta x := (f(x+\delta x) - f(x))/\delta x$ zur Schrittweite δx approximiert, indem die Limesbildung weggelassen wird. x und $x + \delta x$ sind dabei Gitterpunkte. Wir erhalten so die Methode der *finiten Differenzen*[2]. Anschaulich ist klar, daß eine Verfeinerung des Gitters, d.h eine Verkleinerung der Schrittweite δx, zu einer besseren Approximation des Differentialquotienten führt.

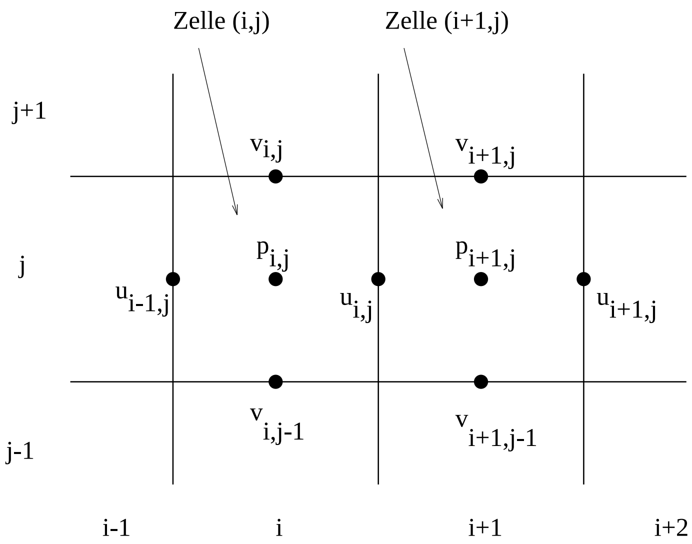


Abb. 6: Staggered Grid

[2]Daneben existieren eine Reihe anderer Diskretisierungsverfahren, wie *finite Elemente* oder *finite Volumen.*

Im Fall der Navier–Stokes–Gleichungen wird zur Diskretisierung des Gebietes Ω oftmals ein versetztes Gitter ("staggered grid") verwendet, bei dem die verschiedenen gesuchten Variablen nicht in den selben Gitterpunkten liegen. Wir verwenden ein Gitter, in dem der Druck p nur in den Mittelpunkten der Zellen, die Geschwindigkeit u nur in den Mittelpunkten der senkrechten Zellkanten und die Geschwindigkeit v nur in den Mittelpunkten der waagrechten Zellkanten betrachtet wird. Den Index (i, j) tragen der Druck im Mittelpunkt der Zelle (i, j), die u–Geschwindigkeit an der rechten Kante und die v–Geschwindigkeit an der oberen Kante der Zelle (i, j). Der Druck $p_{i,j}$ liegt also bei den Koordinaten $((i - 0.5)\,\delta x, (j - 0.5)\,\delta y)$, $u_{i,j}$ bei den Koordinaten $(i\,\delta x, (j - 0.5)\,\delta y)$ und $v_{i,j}$ bei den Koordinaten $((i - 0.5)\,\delta x, j\,\delta y)$ (vgl. Abbildung 6).
Die Punkte für u, v und p gehören also zu drei, hier nicht dargestellten Gittern, die jeweils um eine halbe Maschenweite versetzt sind.
In Folge dessen kommen nicht alle Werte auf dem Rand des Gebietes zu liegen. So gibt es etwa auf den senkrechten Rändern keine v–Werte und auf den waagrechten Rändern keine u–Werte. Daher wird noch eine Randschicht aus Gitterzellen mitgeführt (vgl. Abbildung 7), und die Randbedingungen werden durch eine Mittelung der nächstliegenden Werte erfüllt.

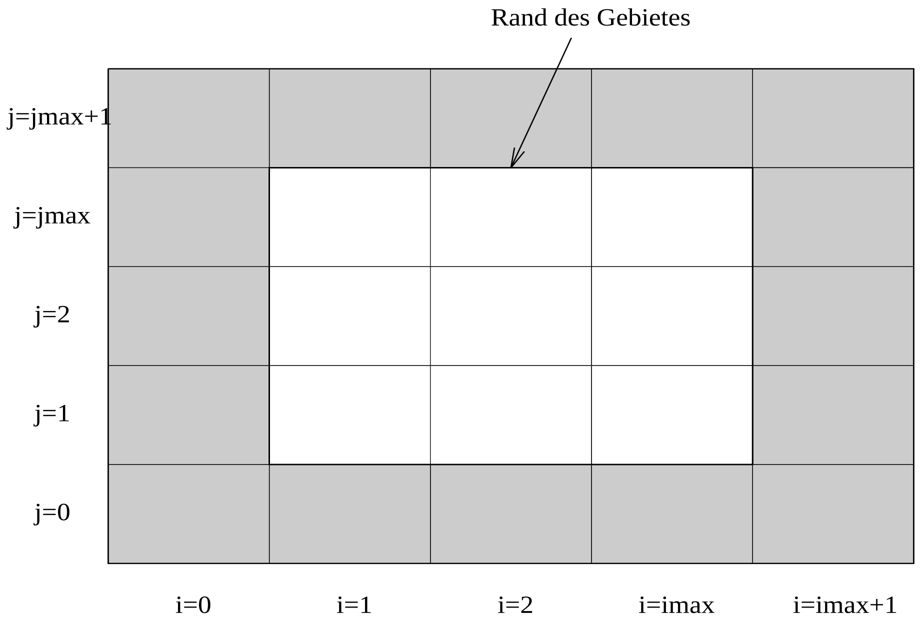


Abb. 7: Gebiet mit Randzellen

Weiterhin wird auch das Zeitintervall $[0, t_{end}]$ in gleiche Teilintervalle $[n\,\delta t, (n + 1)\,\delta t]$, $n = 0, \ldots, t_{end}/\delta t - 1$ zerlegt. Werte für u, v und p werden also nur zu den Zeitpunkten $n\,\delta t$ betrachtet.

Die Navier–Stokes–Gleichungen werden jetzt wie folgt diskretisiert:
Zunächst wollen wir die Ortsableitungen behandeln. Die Impulsgleichung (1) wird in den Mittelpunkten der senkrechten Kanten, die Impulsgleichung (2) wird in den Mittelpunkten der waagrechten Kanten, und die Kontinuitätsgleichung (3) wird in den Zellmittelpunkten ausgewertet.

Die Ausdrücke in Gleichung (1) ersetzen wir am Mittelpunkt der rechten Kante der Zelle (i,j), $i = 1,\ldots,imax-1$, $j = 1\ldots,jmax$, durch

$$
\begin{aligned}
\left[\frac{\partial(u^2)}{\partial x}\right]_{i,j} &:= \frac{1}{\delta x}\left(\left(\frac{u_{i,j}+u_{i+1,j}}{2}\right)^2 - \left(\frac{u_{i-1,j}+u_{i,j}}{2}\right)^2\right) + \\
&\quad\ \alpha\frac{1}{\delta x}\left(\frac{|u_{i,j}+u_{i+1,j}|}{2}\frac{(u_{i,j}-u_{i+1,j})}{2} - \frac{|u_{i-1,j}+u_{i,j}|}{2}\frac{(u_{i-1,j}-u_{i,j})}{2}\right), \\
\left[\frac{\partial(uv)}{\partial y}\right]_{i,j} &:= \frac{1}{\delta y}\left(\frac{(v_{i,j}+v_{i+1,j})}{2}\frac{(u_{i,j}+u_{i,j+1})}{2} - \frac{(v_{i,j-1}+v_{i+1,j-1})}{2}\frac{(u_{i,j-1}+u_{i,j})}{2}\right) + \\
&\quad\ \alpha\frac{1}{\delta y}\left(\frac{|v_{i,j}+v_{i+1,j}|}{2}\frac{(u_{i,j}-u_{i,j+1})}{2} - \frac{|v_{i,j-1}+v_{i+1,j-1}|}{2}\frac{(u_{i,j-1}-u_{i,j})}{2}\right), \qquad (4) \\
\left[\frac{\partial^2 u}{\partial x^2}\right]_{i,j} &:= \frac{u_{i+1,j}-2u_{i,j}+u_{i-1,j}}{(\delta x)^2}, \\
\left[\frac{\partial^2 u}{\partial y^2}\right]_{i,j} &:= \frac{u_{i,j+1}-2u_{i,j}+u_{i,j-1}}{(\delta y)^2}, \qquad \left[\frac{\partial p}{\partial x}\right]_{i,j} := \frac{p_{i+1,j}-p_{i,j}}{\delta x}.
\end{aligned}
$$

Die Ausdrücke in Gleichung (2) ersetzen wir am Mittelpunkt der oberen Kante der Zelle (i,j), $i = 1,\ldots,imax$, $j = 1\ldots,jmax-1$, durch

$$
\begin{aligned}
\left[\frac{\partial(uv)}{\partial x}\right]_{i,j} &:= \frac{1}{\delta x}\left(\frac{(u_{i,j}+u_{i,j+1})}{2}\frac{(v_{i,j}+v_{i+1,j})}{2} - \frac{(u_{i-1,j}+u_{i-1,j+1})}{2}\frac{(v_{i-1,j}+v_{i,j})}{2}\right) + \\
&\quad\ \alpha\frac{1}{\delta x}\left(\frac{|u_{i,j}+u_{i,j+1}|}{2}\frac{(v_{i,j}-v_{i+1,j})}{2} - \frac{|u_{i-1,j}+u_{i-1,j+1}|}{2}\frac{(v_{i-1,j}-v_{i,j})}{2}\right), \\
\left[\frac{\partial(v^2)}{\partial y}\right]_{i,j} &:= \frac{1}{\delta y}\left(\left(\frac{v_{i,j}+v_{i,j+1}}{2}\right)^2 - \left(\frac{v_{i,j-1}+v_{i,j}}{2}\right)^2\right) + \\
&\quad\ \alpha\frac{1}{\delta y}\left(\frac{|v_{i,j}+v_{i,j+1}|}{2}\frac{(v_{i,j}-v_{i,j+1})}{2} - \frac{|v_{i,j-1}+v_{i,j}|}{2}\frac{(v_{i,j-1}-v_{i,j})}{2}\right), \qquad (5) \\
\left[\frac{\partial^2 v}{\partial x^2}\right]_{i,j} &:= \frac{v_{i+1,j}-2v_{i,j}+v_{i-1,j}}{(\delta x)^2}, \\
\left[\frac{\partial^2 v}{\partial y^2}\right]_{i,j} &:= \frac{v_{i,j+1}-2v_{i,j}+v_{i,j-1}}{(\delta y)^2}, \qquad \left[\frac{\partial p}{\partial y}\right]_{i,j} := \frac{p_{i,j+1}-p_{i,j}}{\delta y}
\end{aligned}
$$

und die Terme in der Kontinuitätsgleichung (3) ersetzen wir im Mittelpunkt der Zelle (i,j), $i = 1,\ldots,imax$, $j = 1\ldots,jmax$, durch

$$
\left[\frac{\partial u}{\partial x}\right]_{i,j} := \frac{u_{i,j}-u_{i-1,j}}{\delta x}, \qquad \left[\frac{\partial v}{\partial y}\right]_{i,j} := \frac{v_{i,j}-v_{i,j-1}}{\delta y}. \qquad (6)
$$

Dabei ist α ein Parameter zwischen 0 und 1. Mit $\alpha = 0$ erhält man für die Differentialoperatoren eine Approximation 2. Ordnung, d.h. der Approximationsfehler hat die Genauigkeit

$O(\delta x^2)$ bzw. $O(\delta y^2)$. Diese Approximation kann jedoch für geringe Viskositätswerte, oder wenn die Ableitungsterme $\partial/\partial x$ bzw. $\partial/\partial y$ mit wesentlich größeren Faktoren in das Problem eingehen als die Ableitungsterme $\partial^2/\partial x^2$ und $\partial^2/\partial y^2$, zu Oszillationen in der Lösung führen. In solchen Fällen muß das sogenannte Donor–Cell–Schema ($\alpha = 1$) verwendet werden, das nur eine Approximation 1. Ordnung liefert. In der Praxis benutzt man eine Mischung aus beiden Verfahren mit $\alpha \in [0, 1]$. Der Parameter α sollte dabei etwas größer als der maximale Wert von $|u\,\delta t/\delta x|$ und $|v\,\delta t/\delta y|$ im Gitter gewählt werden.
Bei der Diskretisierung der Impulsgleichungen (1) und (2) bezüglich der Zeit werden die Terme auf der linken Seite zum Zeitpunkt t_{n+1} und die Terme auf der rechten Seite zum Zeitpunkt t_n ausgewertet. Die Zeitableitung $\partial u/\partial t$ zur Zeit t_{n+1} wird durch den Differenzenquotienten 1. Ordnung $(u^{(n+1)} - u^{(n)})/\delta t$ ersetzt. Dies entspricht einem expliziten Zeitschritt–Verfahren mit Hilfe der Euler–Formel[3].
Die Kontinuitätsgleichung (3) wird zum Zeitpunkt t_{n+1} ausgewertet.

5. Der Algorithmus

Das gesamte Lösungsverfahren läßt sich nun mit den folgenden Einzelschritten beschreiben.

Die Zeitschleife
In einer äußeren Iterationsschleife wird beginnend beim Zeitpunkt $t = 0$ die Zeit solange um δt erhöht, bis die Endzeit t_{end} erreicht ist. Im Zeitschritt n ($n = 0, 1, \ldots$) werden die Differentialgleichungen wie oben beschrieben diskretisiert. Dabei seien die Werte zum Zeitpunkt t_n bereits bekannt, und die Werte zum Zeitpunkt t_{n+1} sollen berechnet werden. In der kompakten Notation hat der gesamte Algorithmus die folgende Form

(Transport Schritt)

$$\frac{\mathbf{F}^{(n)} - \mathbf{u}^{(n)}}{\partial t} = \mathbf{g} - \nabla \cdot (\mathbf{u}^{(n)} \otimes \mathbf{u}^{(n)}) + \nu \Delta \mathbf{u}^{(n)} \tag{7}$$

(Projektions-Schritt) Löse das Sattelpunkt Problem

$$\begin{aligned} \frac{\mathbf{u}^{(n+1)} - \mathbf{F}^{(n)}}{\partial t} &= -\nabla p^{(n+1)} && (8) \\ \nabla \cdot \mathbf{u}^{(n+1)} &= 0 \quad . && (9) \end{aligned}$$

Dies geschieht, indem man auf Gleichung (8) den Divergenzoperator anwendet und Gleichung (9) ausnutzt. Dann erhält man die sogenannte Druck-Poisson Gleichung

$$\Delta p^{(n+1)} = \frac{1}{\partial t} \nabla \cdot \mathbf{F}^{(n)} \ .$$

Verfahren der obigen Bauart werden in der Literatur als Splitting-Methoden bezeichnet. Der Grund liegt in der getrennten (gesplittet) Behandlung der Impulsgleichung bzw. der

[3] Es gibt auch implizite Verfahren, bei denen auf der rechten Seite auch Werte zum Zeitpunkt t_{n+1} vorkommen. Diese erlauben wesentlich größere Zeitschritte. Die Lösung der entstehenden Gleichungssysteme ist aber meist wesentlich aufwendiger.

Kontinuitätsgleichung. Die *entkoppelte* Behandlung beider Gleichungen hat enorme algorithmische Vorteile. Splitting Methoden sind deshalb die wohl effizientesten Methoden für die instationären, inkompressiblen Navier-Stokes Gleichungen. Ihr Prototyp ist (1968) von A.J. Chorin (unabhänging davon auch durch R. Temam) angegeben worden, so daß man oft auch die Bezeichnung Chorinsche Projektionsmethode liest.

Die diskreten Impulsgleichungen
Die diskreten Impulsgleichungen werden so umgestellt, daß auf der linken Seite nur noch die Geschwindigkeit $u_{i,j}^{(n+1)}$ bzw. $v_{i,j}^{(n+1)}$ stehen. Dann erhalten die Gleichungen die Form

$$\begin{aligned} u_{i,j}^{(n+1)} &= F_{i,j}^{(n)} - \frac{\delta t}{\delta x}(p_{i+1,j}^{(n+1)} - p_{i,j}^{(n+1)}) && (10) \\ & \qquad i = 1,\ldots,imax-1, \quad j = 1,\ldots,jmax; \\ v_{i,j}^{(n+1)} &= G_{i,j}^{(n)} - \frac{\delta t}{\delta y}(p_{i,j+1}^{(n+1)} - p_{i,j}^{(n+1)}) && (11) \\ & \qquad i = 1,\ldots,imax, \quad j = 1,\ldots,jmax-1. \end{aligned}$$

Die Terme $F_{i,j}^{(n)}$ und $G_{i,j}^{(n)}$ beinhalten die in den diskreten Impulsgleichungen vorkommenden Geschwindigkeiten zum Zeitpunkt n. Mit den diskretisierten Ausdrücken aus (4) und (5) sind dies:

$$\begin{aligned} F_{i,j} &:= u_{i,j} + \delta t \left(\frac{1}{Re} \left(\left[\frac{\partial^2 u}{\partial x^2}\right]_{i,j} + \left[\frac{\partial^2 u}{\partial y^2}\right]_{i,j} \right) - \left[\frac{\partial (u^2)}{\partial x}\right]_{i,j} - \left[\frac{\partial (uv)}{\partial y}\right]_{i,j} + g_x \right) && (12) \\ & \qquad i = 1,\ldots,imax-1, \quad j = 1,\ldots,jmax; \\ G_{i,j} &:= v_{i,j} + \delta t \left(\frac{1}{Re} \left(\left[\frac{\partial^2 v}{\partial x^2}\right]_{i,j} + \left[\frac{\partial^2 v}{\partial y^2}\right]_{i,j} \right) - \left[\frac{\partial (uv)}{\partial x}\right]_{i,j} - \left[\frac{\partial (v^2)}{\partial y}\right]_{i,j} + g_y \right) && (13) \\ & \qquad i = 1,\ldots,imax, \quad j = 1,\ldots,jmax-1. \end{aligned}$$

Die Druckgleichung
Jetzt können wir die durch die rechten Seiten von (10) und (11) gegebenen Ausdrücke für die Geschwindigkeiten $u_{i-1,j}^{(n+1)}, u_{i,j}^{(n+1)}, v_{i,j-1}^{(n+1)}$ und $v_{i,j}^{(n+1)}$ in die diskrete Kontinuitätsgleichung der Zelle (i,j) einsetzen und erhalten eine Gleichung, die nur noch Druckwerte zum Zeitpunkt t_{n+1} und (bereits bekannte) Geschwindigkeiten zum Zeitpunkt t_n enthält:

$$\begin{aligned} & \frac{p_{i+1,j}^{(n+1)} - 2p_{i,j}^{(n+1)} + p_{i-1,j}^{(n+1)}}{(\delta x)^2} + \frac{p_{i,j+1}^{(n+1)} - 2p_{i,j}^{(n+1)} + p_{i,j-1}^{(n+1)}}{(\delta y)^2} = \\ & \qquad \frac{1}{\delta t} \left(\frac{F_{i,j}^{(n)} - F_{i-1,j}^{(n)}}{\delta x} + \frac{G_{i,j}^{(n)} - G_{i,j-1}^{(n)}}{\delta y} \right), \qquad (14) \\ & \qquad i = 1,\ldots,imax, \quad j = 1,\ldots,jmax. \end{aligned}$$

Dies ist die bekannte Form einer diskretisierten Poisson–Gleichung für die Größe $p^{(n+1)}$

$$\frac{\partial^2 p^{(n+1)}}{\partial x^2} + \frac{\partial^2 p^{(n+1)}}{\partial y^2} = rhs$$

auf dem Gebiet Ω mit einer beliebigen rechten Seite rhs. Zur eindeutigen Lösbarkeit benötigen wir aber noch die Randwerte $p_{i,j}$ $(i = 0,\ imax+1,\ j = 0,\ jmax+1)$, $F_{i,j}$ $(i = 0, imax)$ und $G_{i,j}$ $(j = 0, jmax)$, die wir aus den Impulsgleichungen am Rand gewinnen können (siehe unten).

Jetzt ist es möglich, die Druckgleichung (14) mit einem beliebigen Verfahren zur Lösung linearer Gleichungssysteme zu lösen. Da direkte Verfahren wie die Gaußelimination bei sehr großen Problemen (hier $imax \cdot jmax$ Unbekannte) einen zu großen Rechenaufwand besitzen, kommen bei der numerischen Lösung partieller Differentialgleichungen gewöhnlich iterative Gleichungslöser zum Einsatz. Bei unserem Problem haben wir jeweils für die Lösung der Gleichungssysteme eine recht gute Näherung: den Druck $p^{(n)}$ aus dem alten Zeitschritt. Nur iterative Verfahren können das effizient ausnutzen.

Ein klassisches iteratives Verfahren ist das Gauß–Seidel–Verfahren, bei dem, ausgehend von einem Startwert, in jedem Zyklus sukzessive alle Zellen abgelaufen werden und in der Zelle (i, j) der Druck in dieser Zelle so geändert wird, daß die zugehörige Gleichung exakt erfüllt wird.

Eine verbesserte Variante ist das SOR–Verfahren:

$it = 1, \ldots, itmax$
 $i = 1, \ldots, imax$
  $j = 1, \ldots, jmax$

$$\begin{aligned} p_{i,j}^{it+1} &:= (1-\omega)\, p_{i,j}^{it} + \\ &\frac{\omega}{2\left(\frac{1}{(\delta x)^2} + \frac{1}{(\delta y)^2}\right)} \left(\frac{p_{i+1,j}^{it} + p_{i-1,j}^{it+1}}{(\delta x)^2} + \frac{p_{i,j+1}^{it} + p_{i,j-1}^{it+1}}{(\delta y)^2} - rhs_{i,j} \right) \end{aligned} \tag{15}$$

$rhs_{i,j}$ ist die rechte Seite der Druckgleichung (14) für die Zelle (i, j) und ω ist ein Parameter (Relaxationsfaktor), der aus dem Intervall $]0, 2]$ gewählt werden muß (häufig $\omega = 1.7$). Für $\omega = 1$ ergibt sich das Gauß–Seidel–Verfahren. Die Iteration wird abgebrochen, wenn die maximale Anzahl von Iterationsschritten $itmax$ erreicht ist oder wenn das Residuum

$$res := \left(\sum_{i=1}^{imax} \sum_{j=1}^{jmax} \left(\frac{p_{i+1,j} - 2p_{i,j} + p_{i-1,j}}{(\delta x)^2} + \frac{p_{i,j+1} - 2p_{i,j} + p_{i,j-1}}{(\delta y)^2} - rhs_{i,j} \right)^2 / (imax \cdot jmax) \right)^{1/2} \tag{16}$$

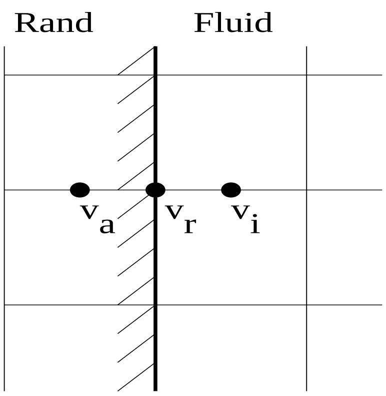
eine vom Benutzer zu wählende Toleranzgrenze ε unterschritten hat.

Als Startwert der Iteration zur Berechnung der Druckwerte $p^{(n+1)}$ dienen jeweils die Druckwerte zum Zeitpunkt n.

Mit den so berechneten Druckwerten zum Zeitpunkt t_{n+1} können dann auch die Geschwindigkeitswerte u und v zum Zeitpunkt t_{n+1} gemäß (10) und (11) berechnet werden.

Die Randwerte für die diskreten Gleichungen bei Haftbedingungen
Bei der Berechnung von F und G nach (12) und (13) wird für $i = 1, imax$ und $j = 1, jmax$ auf u– und v–Werte zugegriffen, die auf dem Gebietsrand oder sogar außerhalb liegen können. Wir wollen uns hier zunächst auf Haftbedingungen beschränken, bei denen die kontinuierlichen Geschwindigkeiten am Rand jeweils 0 sein sollen. Also setzen wir für die Werte, die direkt auf dem Rand liegen

$$u_{0,j} = 0,\ u_{imax,j} = 0,\ j = 1,\ldots,jmax \qquad v_{i,0} = 0,\ v_{i,jmax} = 0,\ i = 1,\ldots,imax. \tag{17}$$



Da weiterhin auf den senkrechten Rändern keine v–Werte und auf den waagrechten Rändern keine u–Werte liegen, wird hier der Randwert 0 durch eine Mittelung der Werte zu beiden Seiten des Randes erzielt:

$$v_r := \frac{v_a + v_i}{2} = 0 \quad \Rightarrow \quad v_a = -v_i.$$

An den 4 Rändern erhalten wir somit die weiteren Bedingungen:

$$\begin{aligned} v_{0,j} &= -v_{1,j}, & v_{imax+1,j} &= -v_{imax,j}, & j &= 1,\ldots,jmax, \\ u_{i,0} &= -u_{i,1}, & u_{i,jmax+1} &= -u_{i,jmax}, & i &= 1,\ldots,imax. \end{aligned} \tag{18}$$

Wie schon erwähnt, wird der Druck am Rand aus den diskreten Impulsgleichungen hergeleitet, und zwar durch Multiplikation mit dem jeweiligen Normalenvektor. Es ergibt sich insgesamt:

$$\begin{aligned} p_{0,j} &= p_{1,j}, & p_{imax+1,j} &= p_{imax,j}, & j &= 1,\ldots,jmax; \\ p_{i,0} &= p_{i,1}, & p_{i,jmax+1} &= p_{i,jmax}, & i &= 1,\ldots,imax \end{aligned} \tag{19}$$

und

$$\begin{aligned} F_{0,j} &= u_{0,j}, & F_{imax,j} &= u_{imax,j}, & j &= 1,\ldots,jmax, \\ G_{i,0} &= v_{i,0}, & G_{i,jmax} &= v_{i,jmax}, & i &= 1,\ldots,imax. \end{aligned} \tag{20}$$

Die Stabilitätsbedingung
Um die Stabilität des numerischen Algorithmus zu gewährleisten und keine Oszillationen zu erzeugen, müssen schließlich die folgenden drei Stabilitätsbedingungen an die Schrittweiten δx, δy und δt gestellt werden:

$$\frac{2}{Re}\,\delta t < \frac{(\delta x)^2(\delta y)^2}{(\delta x)^2 + (\delta y)^2}, \qquad |u_{max}|\,\delta t < \delta x, \qquad |v_{max}|\,\delta t < \delta y.^{4} \tag{21}$$

[4] Die letzten beiden Bedingungen sind die sogenannten Courant–Friedrichs–Levi–(CFL)–Bedingungen.

$|u_{max}|$ und $|v_{max}|$ sind die maximalen Absolutwerte der vorkommenden Geschwindigkeiten. Basierend auf diesen Stabilitätsbedingungen kann eine adaptive Zeitschrittweitensteuerung verwendet werden. Dabei wird δt für den nächsten Zeitschritt so gewählt, daß jede der drei Bedingungen erfüllt ist:

$$\delta t := \tau \min\left(\frac{Re}{2}\left(\frac{1}{\delta x^2}+\frac{1}{\delta y^2}\right)^{-1}, \frac{\delta x}{|u_{max}|}, \frac{\delta y}{|v_{max}|}\right). \tag{22}$$

Der Faktor $\tau \in]0,1]$ ist ein Sicherheitsfaktor.

Zusammenfassung
Abschließend sei der Algorithmus nochmals übersichtlich dargestellt:

```
Einlesen der Problemparameter
Setze t := 0, n := 0
Belege u,v,p mit Anfangswerten
Solange t < t_end
   Wähle δt nach (22)
   Setze die Randwerte für u und v gemäß (17),(18)
   Berechne F^(n) und G^(n) gemäß (12),(13),(20)
   Berechne die rechte Seite rhs der Druckgleichung (14)
   Setze it := 0
   Solange it < itmax und res > ε
     Setze die Randwerte für den Druck gemäß (19)
     Führe einen SOR-Zyklus gemäß (15) durch
     Berechne das Residuum res der Druckgleichung gemäß (16)
     it := it + 1
   Berechne u^(n+1) und v^(n+1) gemäß (10),(11)
   Gegebenenfalls Ausgabe der Werte u,v,p zur Visualisierung
   t := t + δt
   n := n + 1
Ausgabe der Werte u,v,p zur Visualisierung
```

6. Die Problemgrößen und Datenstrukturen

Der oben beschriebene Algorithmus soll im Praktikum in der Programmiersprache C realisiert werden. Er benötigt die folgenden Größen, die, bis auf die Variablen `t`, `it`, `delx` und `dely`, am besten in einer Eingabedatei zur Verfügung gestellt werden. `REAL` steht dabei jeweils für `float` oder `double` und kann mittels `#define REAL float` bzw. `double` gesetzt werden.

- Geometriegrößen:

`REAL xlength`	Gebietslänge in x–Richtung
`REAL ylength`	Gebietslänge in y–Richtung
	Das Berechnungsgebiet ist somit $\Omega = [0, \texttt{xlength}] \times [0, \texttt{ylength}]$.
`int imax`	Anzahl der inneren Zellen in x–Richtung
`int jmax`	Anzahl der inneren Zellen in y–Richtung
`REAL delx`	Länge einer Zelle in x–Richtung δx
`REAL dely`	Länge einer Zelle in y–Richtung δy

- Größen für die Zeititeration:

`REAL t`	aktuelle Zeit
`REAL t_end`	Endzeit t_{end}
`REAL delt`	Zeitschrittweite δt
`REAL tau`	Sicherheitsfaktor für die Zeitschrittweitensteuerung τ
`REAL del_vec`	Zeitabstand, in dem Daten zur Visualisierung in eine Datei geschrieben werden sollen

- Parameter für die Druckiteration:

`int itermax`	Maximale Anzahl von Druck-Iterationen pro Zeitschritt
`int it`	Zähler für die SOR–Iterationen
`REAL res`	Norm des Residuums der Druckgleichung
`REAL eps`	Genauigkeitskriterium ε für die Druckiteration (res $<$ eps)
`REAL omg`	Relaxationsfaktor ω für die SOR-Iteration
`REAL alpha`	Upwind-Differencing-Faktor α

- Problemabhängige Größen:

`REAL Re`	Reynoldszahl Re
`REAL GX,GY`	äußere Kräfte g_x, g_y, z.B. Gravitation
`REAL UI,VI,PI`	Anfangsbelegung der Geschwindigkeiten und des Drucks

Weiterhin sollen die folgenden Felder der Dimension `[0,imax+1]x[0,jmax+1]` als Datenstrukturen verwendet werden:

- Datenfelder

`REAL **U`	Geschwindigkeit in x–Richtung
`REAL **V`	Geschwindigkeit in y–Richtung
`REAL **P`	Druck
`REAL **RHS`	Rechte Seite für die Druckiteration
`REAL **F,**G`	F, G

Für diese Größen sollte der Speicherplatz *dynamisch allokiert* werden, d.h. es wird beim Compilieren kein fester und maximal großer Speicherplatz reserviert, sondern bei jedem Programmstart nur soviel wie nötig. Dazu sei hier eine Möglichkeit angegeben [5].

[5] vgl. Press, W., Flannery, B., Teukolsky, S. und W.T. Vetterling (1990). *Numerical Recipes in C.* Cambridge: Cambridge University Press.

```
REAL **RMATRIX(int nrl,int nrh,int ncl,int nch){
/** Reserviert Speicherplatz fuer Matrix der Groesse [nrl,nrh]x[ncl,nch] **/
  int i;
  REAL **m;
  /**** Pointer auf Zeilen allokieren ****/
  if((m=(REAL**) malloc((unsigned)(nrh-nrl+1)*sizeof(REAL*)))==NULL){
    printf("kein speicher\n");
    exit(0);}
  m -= nrl;

  /** Zeilen allokieren und vorher allokierte Pointer darauf setzen **/
  for(i=nrl;i<=nrh;i++){
    if((m[i]=(REAL*) malloc((unsigned)(nch-ncl+1)*sizeof(REAL)))==NULL){
      printf("kein speicher\n");
      exit(0);}
    m[i] -= ncl;}

  /** Den Pointer auf den Array von Pointern auf Zeilen zurueckgeben **/
  return m;
}
```

Zur sauberen Speicherverwaltung gehört auch die Freigabe nicht mehr benötigten Speichers. Die zu `RMATRIX` gehörende Prozedur zur Speicherfreigabe lautet:

```
void FREE_RMATRIX(REAL **m, int nrl,int nrh,int ncl,int nch){
/** Gibt Speicher einer mit RMATRIX allokierten Matrix frei **/
  int i;
  for(i=nrh; i>=nrl; i--) free((char*) (m[i]+ncl));
  free((char*) (m+nrl));
}
```

Typische Aufrufe dieser Prozeduren sind
`U = RMATRIX(0,imax,0,jmax);` und `FREE_RMATRIX(U,0,imax,0,jmax);`.

7. Das Programm

Schreiben Sie unter Verwendung obiger Größen und Datenstrukturen die folgenden Prozeduren: [6]

1. `int READ_PARAMETER(char *Inputfile,xlength,ylength,imax,jmax,delx,dely, delt,t_end,tau,itermax,eps,omg,alpha,Re,GX,GY,UI,VI,PI)`
   Die aufgelisteten Größen werden aus der durch den Übergabeparameter `Inputfile` bestimmten Eingabedatei eingelesen und `delx`, `dely` werden aus `imax` und `xlength` bzw. `jmax` und `ylength` bestimmt. Der Rückgabewert dient einer möglichen Fehlerbehandlung.

[6] Zur Abkürzung haben wir hier bei bereits eingeführten Größen die Typangabe im jeweiligen Prozedurkopf weggelassen. Um derart lange Übergabelisten zu vermeiden, können sinngebundene Größen auch mit Hilfe des Sprachelements `structure` gruppiert werden.

2. `void INIT_UVP(U,V,P,imax,jmax,UI,VI,PI)`
   Die Felder `U,V,P` werden auf dem ganzen Gebiet mit den konstanten Werten `UI,VI` und `PI` als Anfangswerten vorbelegt.

3. `void COMP_delt(delt,imax,jmax,delx,dely,U,V,Re,tau)`
   Nach (22) wird die Schrittweite `delt` für den nächsten Zeitschritt berechnet. Ist `tau` negativ, so soll die in `READ_PARAMETER` eingelesene Zeitschrittweite verwendet werden.

4. `void SETBCOND(U,V,imax,jmax)`
   Die Randwerte für die Felder `U` und `V` werden gemäß den Formeln (17), (18) gesetzt.

5. `void COMP_FG(U,V,F,G,imax,jmax,delt,delx,dely,GX,GY,alpha,Re)`
   Berechnung von `F` und `G` gemäß (12) und (13). Dabei müssen am Rand die Formeln (20) angewendet werden.

6. `void COMP_RHS(F,G,RHS,imax,jmax,delt,delx,dely)`
   Berechnung der rechten Seite der Druckgleichung (14).

7. `int POISSON(P,RHS,imax,jmax,delx,dely,eps,itermax,omg,res)`
   SOR–Iteration für die Druck–(Poisson–)Gleichung gemäß (15). In jeder Iteration müssen zunächst die Randwerte für `P` gemäß (19) gesetzt werden. Die Iteration wird abgebrochen, wenn die Norm des Residuums `res` die Toleranzgrenze `eps` unterschreitet oder die maximale Iterationsanzahl `itermax` erreicht ist. Nach dem Abbruch soll die Anzahl der benötigten Iterationen zurückgegeben werden und in `res` die aktuelle Größe der Norm des Residuums stehen.

8. `void ADAP_UV(U,V,F,G,P,imax,jmax,delt,delx,dely)`
   Die neuen Geschwindigkeiten werden gemäß (10) und (11) berechnet.

9.
```
void OUTPUT (char *filename, REAL **X,
             REAL delx, REAL dely,
             int avx, int avy,
             int ia, int ja, int ie, int je)
```
   Ausgabe der Teilmatrix `X[ia..ie][ja..je]` aus der Matrix `X[..][..]` in ein File. Falls `avx!=0` dann soll an Stelle von `X[i][j]` der gemittelte Wert `(X[i][j]+X[i-1][j])/2` ausgegeben werden und falls `avy!=0` dann `(X[i][j]+X[i][j-1])/2`. Damit können wir sehr einfach Geschwindigkeiten ausgeben, die im Zellmittelpunkt liegen. Das Format des Files ist:

4 ×	`int`	`ia,ja,ie,je`
2 ×	`double`	`delx,dely`
.. ×	`double`	`X[ia][ja],X[ia][ja+1],...,X[ie][je]`

   Alle Werte werden *binär* mit der Funktion `fwrite(...)` geschrieben. Zur Visualisierung der Daten wird das sehr mächtige Programm MatLab benutzt, das eigentlich kein Visualisierungsprogramm ist, sondern eine Programmierumgebung für numerische Probleme. Zum Starten von MatLab:

```
rlogin vergil
setenv MATLABPATH /mount/people/cfdprakt/Nast2d/matlab/
matlab -nojvm
```

In MatLab selbst können die Daten mittels z.B.

```
>>[ij,dxy,p]=Input('/mount/people/cfdprakt/Nast2d/jobs/DrivenC/p') ;
```

eingelesen werden und mittels

```
>>pcolor(p); shading flat;
```

dargestellt. Für weitere Informationen zu MatLab sei auf

http://wissrech.iam.uni-bonn.de/miscellany/doc/matlab/helpdesk.html

verwiesen. Bemerkung: In MatLab könnt ihr z.B. mit `>> p[10][10]` einzelne Werte ausgeben lassen. Dadurch ist es möglich, untereinader zu überprüfen, ob ihr *genau* die gleiche Ergebnisse erhalten habt.

Im Hauptprogramm soll schließlich der oben angegebene Algorithmus umgesetzt werden.

8. Hinweise zum modularen Programmieren

Um das entstehende Programm übersichtlich und flexibel zu gestalten und bei der Entwicklung unnötig lange Compilierzeiten zu vermeiden, ist es zweckmäßig, die Prozeduren in verschiedenen Modulen (Dateien) zu programmieren, die nach dem Compiliervorgang mit dem Hauptprogramm zusammengebunden werden. So können z.B. die Prozeduren `READ_PARAMETER`, `RMATRIX` und `INIT_UVP` in einer Datei `init.c` gruppiert werden. Ist dieses Modul einmal fertig entwickelt und compiliert, so muß es nicht bei jeder Änderung anderer Details erneut übersetzt werden.

Für das hier zu entwickelnde Programm bietet sich (auch in Hinsicht auf spätere Erweiterungen) folgende modulare Struktur an:

`init.c`:	`READ_PARAMETER`, `RMATRIX`, `FREE_RMATRIX` und `INIT_UVP`
`boundary.c`:	`SETBCOND`
`uvp.c`:	`COMP_FG`, `COMP_RHS`, `POISSON`, `COMP_delt` und `ADAP_UV`
`visual.c`:	`OUTPUT`
`main.c`:	Hauptprogramm

Benötigt man dann in einem Modul `A.c` eine Prozedur, die in einem anderem Modul `B.c` definiert ist, so muß der zugehörige Prozedurkopf im Modul `A.c` bekannt gemacht werden, was meist mit Hilfe von sogenannten *Header–Files* geschieht: Alle Prozedurköpfe eines Moduls `A.c`, die in anderen Modulen benötigt werden, werden in das Header–File `A.h` geschrieben, das dann in den entsprechenden aufrufenden Modulen mittels `#include` "`A.h`" eingebunden wird. Dies erspart einem die explizite Auflistung der jeweils benötigten Prozedurköpfe in jeder Datei.

In einer UNIX–Umgebung kann dann der Compilier– und Link–Vorgang mit Hilfe eines *makefiles*, das in der Datei gleichen Namens spezifiziert wird, automatisiert werden. In unserem Fall könnte dies wie folgt aussehen:

```
CC = gcc
CFLAGS = -Wall -pedantic
.SUFFIXES: .o .c
.c.o:  ; $(CC) -c $(CFLAGS) $*.c
OBJ = init.o boundary.o uvp.o main.o

run: $(OBJ)
<TAB>$(CC) $(CFLAGS) -o run $(OBJ) -lm

init.o    : init.h
boundary.o: boundary.h
uvp.o     : uvp.h
main.o    : init.h boundary.h uvp.h
```

Erläuterungen zu den einzelnen Befehlen:

- `CC = gcc` und `CFLAGS = -Wall -pedantic` sind *Makrodefinitionen*, die einen bestimmten C–Compiler und mögliche Compilier–Flags festlegen. Hierbei verursachen die Flags `-Wall` und `-pedantic` viele Warnungen bei kleinen „Unsauberkeiten" im Programm. Verwendet man das Flag `-O`, so wird ein laufzeitoptimiertes Programm erzeugt.

- `.SUFFIXES: .o .c` legt eine allgemeine Abhängigkeit fest. Wird ein .c–File geändert, so muß das zugehörige .o–File neu erzeugt werden.

- `.c.o: ; $(CC) -c $(CFLAGS) $*.c` bewirkt die *Transformation* der .c–Files in die .o–Files (Objectfiles) mittels des Befehls `$(CC) -c $(CFLAGS) $*.c` . Hierbei bedeutet `$(CC)` z.B., daß an dieser Stelle der durch das Makro `CC` festgelegte C–Compiler eingesetzt wird.

- `OBJ = init.o boundary.o uvp.o main.o` ist wieder eine Makrodefinition.

- `run: $(OBJ)` legt die Abhängigkeiten des Programms `run` fest. Ist ein neues .o–File in `OBJ` entstanden, so muß das Programm neu zusammengelinkt werden.

- `$(CC) $(CFLAGS) -o run $(OBJ) -lm` ist die Anweisung, die die im Makro `OBJ` festgelegten Module zusammenbindet. Das Flag `-o run` bewirkt hierbei, daß das erzeugte Programm `run` heißt (default ist `a.out`) und `-lm` bindet die Mathematik–Bibliothek mit ein. WICHTIG: Dieser Befehl muß mit einem Tabulatorzeichen beginnen!

- Die restlichen Befehle legen spezielle Abhängigkeiten fest. Wird hier zum Beispiel `uvp.h` geändert, so muß auch `uvp.o` und `main.o` neu erzeugt werden.

Weitere Informationen zu diesem Themenkreis finden sich in gängigen UNIX– und C–Büchern.

9. Das erste Beispiel: "Driven Cavity"

Als erstes Beispiel soll ein typisches Modellproblem der numerischen Strömungsmechanik, das sogenannte "Driven–Cavity–Problem" (zu deutsch "Nischenströmung") simuliert werden. Es handelt sich dabei um einen mit Flüssigkeit gefüllten Topf, über den ein Band mit konstanter Geschwindigkeit gezogen wird. Dabei werden an allen 4 Rändern Haftbedingungen verwendet. Lediglich am oberen Rand wird die Geschwindigkeit in x–Richtung u nicht auf 0 sondern auf 1 gesetzt, um das gezogene Band zu simulieren. Im Programm wird dies realisiert, indem die Randwerte am oberen Rand auf

$$u_{i,jmax+1} = 2.0 - u_{i,jmax}, \quad i = 1, \ldots, imax$$

gesetzt werden.

Als weitere Parameter sollen verwendet werden:

`imax = 50`	`jmax = 50`	`xlength = 10`	`ylength = 10`
`delt = 0.02`	`t_end = 2.0`	`tau = 0.5`	`del_vec = 2.0`
`eps = 0.001`	`omg = 1.7`	`alpha = 0.5`	`itermax = 100`
`GX = 0.0`	`GY = 0.0`	`Re = 10`	
`UI = 0.0`	`VI = 0.0`	`PI = 0.0`	