

SIMULATION EINER ZWEIDIMENSIONALEN WASSERSTRÖMUNG

1. EINFÜHRUNG

In diesem Projekt wurde versucht eine zweidimensionale Wasserströmung anhand eines bestimmten Problems numerisch zu simulieren. Numerische Simulationen erlangen in der Wissenschaft immer größere Bedeutung. Während man vor wenigen Jahrzehnten für den Aufbau kostspieliger Versuchsaufbauten noch viel Zeit und Geld investieren musste, können heutzutage viele technische Experimente am Computer durchgeführt und ausgewertet werden. Durch die Verbindung des technischen Sachverstandes der Ingenieure, den numerischen Verfahren der Mathematiker und der modernen Methoden und Rechner der Informatiker entstehen heute Simulationsprogramme für Wasserströmungen, Luftströmungen - zur Luftwiderstandsmessung bei Flugzeugen oder Automobilen -, Schmelzprozesse und mechanische Prozesse. Auch in der Wettervorhersage, in der Geologie, der Bioinformatik, der Mechanik und in vielen anderen Gebieten der Physik und 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.

2. DAS PROBLEM

Dieses Projekt behandelt die Simulation einer zweidimensionalen Wasserströmung anhand eines Beispiels. Das Programm beschränkt sich dabei ausschließlich auf die Simulation eines inkompressiblen (=nicht komprimierbaren), viskosen (=zähen) und laminaren (=regelmäßig und inturbulenten) Mediums. Simuliert wird die Strömung in einem Hohlraum, über dem mit konstanter Geschwindigkeit Wasser fließt.

Die gesuchten Größen sind dabei die Geschwindigkeit der Strömung, gegeben durch ihre Komponenten in x und y -Richtung u und v sowie der Druck p .

3. DAS MATHEMATISCHE MODELL

Um ein Modell der Strömung erstellen zu können benötigen wir einige physikalische Formeln, die instationäre, inkompressible, viskose, laminare Flüssigkeiten beschreiben. Dazu verwenden wir die Navier-Stokes-Gleichungen. Diese Gleichungen bestehen aus einem System partieller Differentialgleichungen die wie folgt aussehen:

$$\begin{aligned}\frac{\partial u}{\partial t} + \frac{\partial p}{\partial x} &= \nu \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) - \frac{\partial(u^2)}{\partial x} - \frac{\partial(uv)}{\partial y} + g_x \\ \frac{\partial v}{\partial t} + \frac{\partial p}{\partial y} &= \nu \left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right) - \frac{\partial(uv)}{\partial x} - \frac{\partial(v^2)}{\partial y} + g_y \\ \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} &= 0\end{aligned}$$

Die beiden ersten Differentialgleichungen sind dabei die zwei Impulsgleichungen und die dritte Gleichung ist die Kontinuitätsgleichung (Massenerhaltung).

Dieses Gleichungssystem soll nun auf dem rechteckigen Gebiet $\Omega = [0, a] \times [0, b] \subset \mathbb{R}^2$ in dem

Zeitintervall $[0, T_{end}]$ gelöst werden.

Dabei sind die drei gesuchten Größen

- $u: \Omega \times [0, T_{end}] \rightarrow \mathbb{R}$ die Geschwindigkeit der Flüssigkeit in x -Richtung,
- $v: \Omega \times [0, T_{end}] \rightarrow \mathbb{R}$ die Geschwindigkeit der Flüssigkeit in y -Richtung,
- $p: \Omega \times [0, T_{end}] \rightarrow \mathbb{R}$ der Druck.

Die reelle Zahl ν ist der Viskositätskoeffizient. Je größer ν wird desto zäher wird das Medium. Die Terme $\frac{\partial u}{\partial t}$ und $\frac{\partial v}{\partial t}$ sind Beschleunigungen. Die Druckableitungen ($\frac{\partial p}{\partial x}$ und $\frac{\partial p}{\partial y}$) entsprechen durch Druckdifferenzen hervorgerufenen Kräften. Die zweiten Ableitungen ($\frac{\partial^2 u}{\partial x^2}$, $\frac{\partial^2 u}{\partial y^2}$, $\frac{\partial^2 v}{\partial x^2}$ und $\frac{\partial^2 v}{\partial y^2}$) entstehen durch innere Reibungskräfte, während $\frac{\partial(u^2)}{\partial x}$, $\frac{\partial(uv)}{\partial y}$, $\frac{\partial(v^2)}{\partial y}$, $\frac{\partial(uv)}{\partial x}$ durch die Überlagerung von Geschwindigkeiten entstehen.

$g_x: \Omega \times [0, T_{end}] \rightarrow \mathbb{R}$ und $g_y: \Omega \times [0, T_{end}] \rightarrow \mathbb{R}$ sind äußere Kräfte, etwa die Erdanziehung oder andere Beschleunigungen denen das System unterworfen ist. Zu Beginn seien Anfangsbedingungen $u = u_0(x, y)$ und $v = v_0(x, y)$ vorgegeben. 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, w_2 die Geschwindigkeitskomponente parallel zum Rand und $\frac{\partial w_1}{\partial n}$ bzw. $\frac{\partial w_2}{\partial n}$ die Ableitungen in Normalenrichtung.

Am senkrechten Rand gilt also

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

und am waagrechten Rand gilt

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

Für Punkte (x, y) des festen Randes sind folgende Randbedingungen möglich:

(1) Haftbedingung:

$$w_1(x, y) = 0, w_2(x, y) = 0 \text{ (Flüssigkeit haftet am Rand)}$$

(2) Rutschbedingung:

$$w_1(x, y) = 0, \frac{\partial w_2(x, y)}{\partial n} = 0 \text{ (keine Reibung am Rand)}$$

(3) Einströmbedingung:

$$w_1(x, y) = w_{1,0}, w_2(x, y) = w_{2,0}, w_{1,0}, w_{2,0} \text{ gegeben}$$

(4) Ausströmbedingung:

$$\frac{\partial w_1(x, y)}{\partial n} = 0, \frac{\partial w_2(x, y)}{\partial n} = 0.$$

Für unser Problem gilt:

4 Wände mit Haftbedingung. Die obere Wand bewegt sich mit konstanter Geschwindigkeit nach rechts (das Wasser strömt oben vorbei), d.h. $w_2 = w_{2,0}$ ist konstant am oberen Rand, im Gegensatz zu $w_2 = 0$ an den anderen Wänden.

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.

Dies bedeutet, dass wir die gesuchten Lösungen für u , v und p nur in endlich vielen Punkten bestimmen werden. Dazu müssen wir die Navier-Stokes-Gleichungen durch Funktionswerte annähern. In unserem 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 Gitterlinien betrachtet. Das Gitter habe in x -Richtung $imax$ und in y -Richtung $jmax$ Zellen der gleichen Größe, so dass die Gitterlinien die Abstände $\delta x = \frac{a}{imax}$ und $\delta y = \frac{b}{jmax}$ 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

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

wird so der kontinuierliche Differentialoperator $\frac{df}{dx}$ durch den Differenzenoperator $\frac{\delta f}{\delta x} = \frac{f(x+\delta x) - f(x)}{\delta x}$ approximiert, indem die Limesbildung weggelassen wird. Die Werte x und $x + \delta x$ sind dabei Gitterpunkte. Wir erhalten so die Methode der finiten Differenzen. Anschaulich ist klar, dass eine Verfeinerung des Gitters, d.h. eine Verkleinerung der Schrittweite δx , zu einer besseren Approximation des Differentialquotienten führt.

Die Navier-Stokes-Gleichungen werden jetzt wie folgt diskretisiert: Zunächst wollen wir die Ortsableitung behandeln. Die erste Impulsgleichung (erste Navier-Stokes-Gleichung) wird in den Mittelpunkten der senkrechten Kanten, die zweite Impulsgleichung (zweite Navier-Stokes-Gleichung) in den Mittelpunkten der waagrechten Kanten und die Kontinuitätsgleichung (dritte Navier-Stokes-Gleichung) in den Zellmittelpunkten ausgewertet.

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

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

Die Ausdrücke der zweiten Gleichung ersetzen wir am Mittelpunkt der oberen Kante der Zelle (i, j) , $i = 1, \dots, imax$, $j = 1, \dots, jmax - 1$, auf ähnliche Weise, durch

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

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

und die Terme in der dritten Gleichung ersetzen wir im Mittelpunkt der Zelle (i, j) , $i = 1, \dots, imax$, $j = 1, \dots, jmax$, durch

$$\begin{aligned} \left[\frac{\partial u}{\partial x} \right]_{i,j} &= \frac{u_{i,j} - u_{i-1,j}}{\delta x} \\ \left[\frac{\partial v}{\partial y} \right]_{i,j} &= \frac{v_{i,j} - v_{i,j-1}}{\delta y} \end{aligned}$$

Dabei ist α ein Parameter zwischen 0 und 1, der je nach Situation passend gewählt werden muss um eine Wirbelbildung zu vermeiden.

Bei der Diskretisierung der Impulsgleichungen bezüglich der Zeit werden die Terme auf der linken Seite zum Zeitpunkt $n + 1$ und die Terme auf der rechten Seite zum Zeitpunkt n ausgewertet. Die Zeitableitung $\frac{\partial u}{\partial t}$ zur Zeit $n + 1$ wird durch den Differenzenquotienten 1. Ordnung $\frac{u^{(n+1)} - u^{(n)}}{\delta t}$ ersetzt. Die Kontinuitätsgleichung wird zum Zeitpunkt $n + 1$ ausgewertet.

5. DER ALGORITHMUS

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

5.1. Die Zeitschleife. In einer äußeren Iterationsschleife wird beginnend beim Zeitpunkt $t = 0$ die Zeit solange um δt erhöht, bis eine Endzeit T_{end} erreicht ist. Im Zeitschritt n werden die Differentialgleichungen wie oben beschrieben diskretisiert. Dabei seien die Werte zum Zeitpunkt n bereits bekannt und die Werte zum Zeitpunkt $n + 1$ sollen berechnet werden.

5.2. 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

$$\begin{aligned} u_{i,j}^{(n+1)} &= F_{i,j}^{(n)} - \frac{\delta t}{\delta x} (p_{i+1,j}^{(n+1)} - p_{i,j}^{(n+1)}) \\ v_{i,j}^{(n+1)} &= G_{i,j}^{(n)} - \frac{\delta t}{\delta y} (p_{i,j+1}^{(n+1)} - p_{i,j}^{(n+1)}) \end{aligned}$$

Die Terme $F_{i,j}^{(n)}$ und $G_{i,j}^{(n)}$ beinhalten die in den diskreten Impulsgleichungen vorkommenden Geschwindigkeiten zum Zeitpunkt n .

5.3. Die Druckgleichung. Jetzt können wir die Geschwindigkeiten $u_{i,j}^{(n+1)}$ 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 $n + 1$ und Geschwindigkeiten vom Zeitpunkt n enthält:

$$\frac{p_{i+1,j}^{(n+1)} - 2p_{i,j}^{(n+1)} + p_{i-1,j}^{(n+1)}}{(\delta x)^2} + \frac{p_{i,j+1}^{(n+1)} - 2p_{i,j}^{(n+1)} + p_{i,j-1}^{(n+1)}}{(\delta y)^2} = \frac{1}{\delta t} \left(\frac{F_{i,j}^{(n)} - F_{i-1,j}^{(n)}}{\delta x} + \frac{G_{i,j}^{(n)} - G_{i,j-1}^{(n)}}{\delta y} \right)$$

Unter Berücksichtigung der Randwerte ist es nun möglich diese Druckgleichung mit einem beliebigen Verfahren zur Lösung linearer Gleichungssysteme zu lösen.

Mit den so berechneten Druckwerten zum Zeitpunkt $n + 1$ können dann auch die Geschwindigkeitswerte u und v zum Zeitpunkt $n + 1$ berechnet werden.

5.4. **Zusammenfassung.** Der Algorithmus sieht also zusammengefasst so aus:

```

Einlesen der Problemparameter
Belege  $u, v, p$  mit Anfangswerten
Visualisiere die Anfangsbedingungen
Setze  $T = 0, n = 0$ 
Solange  $T \leq T_{end}$ 
  Setze die Randwerte für  $u$  und  $v$ 
  Berechne  $F$  und  $G$  zum Zeitpunkt  $n$ 
  Berechne die rechte Seite der Druckgleichung
  Löse das Gleichungssystem für die Druckwerte
  Berechne  $u$  und  $v$  zum Zeitpunkt  $n + 1$ 
   $T := T + \delta t, n := n + 1$ 
Visualisiere die Ergebnisse für  $u$  und  $v$ 

```

6. DAS PROGRAMM

```

unit Stroemung_quelltext;
{Das Programm berechnet die Geschwindigkeitsmatrizen u und v sowie den Druck p
 in einem konstanten Gitternetz und stellt diese danach graphisch dar}

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls;

const
  SIZE_X = 10; // Anzahl der Gitternetzpunkte in x-Richtung
  SIZE_Y = 10; // Anzahl der Gitternetzpunkte in y-Richtung
  SIZE = (SIZE_X+2) * (SIZE_Y+2);

type
  RealVector = Array of Extended;
  Matrix = Array of RealVector;

type TRechne = class(TObject) // Klasse zur Berechnung
public
  dx, dy, gx, gy, alpha, visk, dt: Extended; // Konstanten die bei der Berechnung Auftreten
  wl, wr, wt, wb: Byte;
  f, g, pmat {Die Matrix zur Berechnung des Drucks}: Matrix;
  constructor Create; // Konstruktor
  function SolveLinearSystem(A: Matrix; m, n: Integer): RealVector;
  // Algorithmus zum Lösen linearer Gleichungssysteme; nicht selbst verfasst (Internet)

```

```

function du2dx(i,j: Integer): Extended;
function duvdy(i,j: Integer): Extended;
function d2udx2(i,j: Integer): Extended;
function d2udy2(i,j: Integer): Extended;
function dpdx(i,j: Integer): Extended;
function duvdx(i,j: Integer): Extended;
function dv2dy(i,j: Integer): Extended;
function d2vdx2(i,j: Integer): Extended;
function d2vdy2(i,j: Integer): Extended;
function dpdy(i,j: Integer): Extended;
function dudx(i,j: Integer): Extended;
function dvdy(i,j: Integer): Extended;
procedure setpmat; // initiatisierung von pmat
procedure pneu; // Berechnung des neuen Drucks
procedure schritt; // Durchführung eines Schrittes
procedure randwerte;
procedure rechnefg;
function alphaneu: Extended;
end;

type TGraphik = class(TObject) // Klasse zur Graphischen Darstellung
public
    function Max1: Extended; //Bestimmt den maximalen Betrag der Geschwindigkeitsvektoren
    procedure Pfeil(xs, ys, xe, ye: Integer);
        //Zeichnet Pfeil vom Punkt (xs/ys) zum Punkt (xe/ye)
    procedure Draw; // Zeichnet ein Raster sowie die Geschwindigkeitsvektoren
end;

type
    TForm1 = class(TForm)
        Button1: TButton;
        Button2: TButton;
        Edit1: TEdit;
        Label1: TLabel;
        procedure Button1Click(Sender: TObject); // Initialisiert die Variablen und zeichnet den Zeitschritt
        procedure Button2Click(Sender: TObject);
        procedure FormCreate(Sender: TObject);
        procedure FormResize(Sender: TObject); // Führt einen Zeitschritt durch und gibt ihn graphisch aus
    private
        { Private-Deklarationen }
    public
        procedure init; // Initalisiert die Variablen
    end;

var

```

```

Form1: TForm1; // Das Formular
u, v, p: Matrix; // Die Geschwindigkeitskomponenten u und v sowie der Druck p
r1: TRechne; // Die Instanz des Typs TRechne
z1: TGraphik; // Die Instanz des Typs TGraphik

implementation

{$R *.DFM}

constructor TRechne.Create;

var i: Integer;

begin
  wl := 1;
  wr := 1;
  wt := 2;
  wb := 1;
  dx := 0.1;
  dy := dx;
  dt := 0.01;
  gx := 0;
  gy := 0;
  alpha := 0.12;
  visk := 0.01;
  setpmat; // Initialisierung von pmat
  SetLength(f,SIZE_X+1); // Bestimmung der Größen der Matrizen f und g
  SetLength(g,SIZE_X+1);
  for i := 0 to SIZE_X do
    begin
      SetLength(f[i],SIZE_Y+1);
      SetLength(g[i],SIZE_Y+1);
    end;
  end;

function TRechne.SolveLinearSystem(A: Matrix; m, n: Integer): RealVector;
// Algorithmus zum Lösen linearer Gleichungssysteme; nicht selbst verfasst (Internet)
var
  i, j, k: Integer;
  Pivot: RealVector;
  PivotRow: Integer;
  Multiplicator, Sum: Extended;
begin
  SetLength(A, m, n);
  for i := 0 to m - 1 do

```

```

// Vorwärtselimination
for j := i to m - 2 do begin
  if (A[j, j] = 0) then begin
    // Pivotisierung
    SetLength(Pivot, n + 1);
    Pivot := A[j];
    PivotRow := 0;
    for k := j + 1 to m - 1 do begin
      if (Abs(A[k, j]) > Abs(Pivot[j])) then begin
        Pivot := A[k];
        PivotRow := k;
      end;
      if (PivotRow > 0) then begin
        A[PivotRow] := A[j];
        A[j] := Pivot;
      end else
        raise EMathError.Create('System insolvable');
      end;
    end;
    end;
    Multiplier := A[j + 1, i] / A[i, i];
    for k := i to n - 1 do
      A[j + 1, k] := A[j + 1, k] - (Multiplier * A[i, k]);
    end;
  // Rückwärtssubstitution
  SetLength(Result, m);
  for i := m - 1 downto 0 do begin
    Sum := 0;
    for k := i to m - 1 do
      Sum := Sum + Result[k] * A[i, k] / A[i, i];
    Result[i] := A[i, n - 1] / A[i, i] - Sum;
  end;
end;

function TRechne.du2dx(i, j: Integer): Extended;
begin
  du2dx := 1/(4*dx)*((Sqr(u[i, j]+u[i+1, j]) - Sqr(u[i-1, j]+u[i, j]))+alpha*
  (Abs(u[i, j]+u[i+1, j])*(u[i, j]-u[i+1, j])-Abs(u[i-1, j]+u[i, j])*(u[i-1, j]-u[i, j])
  ));
end;

function TRechne.duvdy(i, j: Integer): Extended;
begin
  duvdy := 1/(4*dy)*(((v[i, j]+v[i+1, j])*(u[i, j]+u[i, j+1])-(v[i, j-1]+v[i+1, j-1])*(
  (u[i, j-1]+u[i, j])) + alpha*(Abs(v[i, j]+v[i+1, j])*(u[i, j]-u[i, j+1])-
  Abs(v[i, j-1]+v[i+1, j-1])*(u[i, j-1]-u[i, j]))));
end;

```

```

end;

function TRechne.d2udx2(i, j: Integer): Extended;
begin
  d2udx2 := 1/Sqr(dx)*(u[i+1, j]-2*u[i, j]+u[i-1, j]);
end;

function TRechne.d2udy2(i, j: Integer): Extended;
begin
  d2udy2 := 1/Sqr(dy)*(u[i, j+1]-2*u[i, j]+u[i, j-1]);
end;

function TRechne.dpdx(i, j: Integer): Extended;
begin
  dpdx := 1/dx*(p[i+1, j]-p[i, j]);
end;

function TRechne.duvdx(i, j: Integer): Extended;
begin
  duvdx := 1/(4*dx)*(((u[i, j]+u[i, j+1])*(v[i, j]+v[i+1, j])-(u[i-1, j]+u[i-1, j+1])*(v[i-1, j]+v[i, j]))+alpha*(Abs(u[i, j]+u[i, j+1])*(v[i, j]-v[i+1, j])-Abs(u[i-1, j]+u[i-1, j+1])*(v[i-1, j]-v[i, j]))));
end;

function TRechne.dv2dy(i, j: Integer): Extended;
begin
  dv2dy := 1/(4*dy)*((Sqr(v[i, j]+v[i, j+1])-Sqr(v[i, j-1]+v[i, j]))+alpha*(Abs(v[i, j]+v[i, j+1])*(v[i, j]-v[i, j+1])-Abs(v[i, j-1]+v[i, j])*(v[i, j-1]-v[i, j]))));
end;

function TRechne.d2vdx2(i, j: Integer): Extended;
begin
  d2vdx2 := 1/Sqr(dx)*(v[i+1, j]-2*v[i, j]+v[i-1, j]);
end;

function TRechne.d2vdy2(i, j: Integer): Extended;
begin
  d2vdy2 := 1/Sqr(dy)*(v[i, j+1]-2*v[i, j]+v[i, j-1]);
end;

function TRechne.dpdy(i, j: Integer): Extended;
begin
  dpdy := 1/dy*(p[i, j+1]-p[i, j])
end;

```

```

function TRechne.dudx(i,j:Integer): Extended;
begin
  dudx := 1/dx*(u[i,j]-u[i-1,j])
end;

function TRechne.dvdy(i,j:Integer): Extended;
begin
  dvdy := 1/dy*(v[i,j]-v[i,j-1]);
end;

procedure TRechne.setpmat; //Initialisierung von pmat (Matrix zur Berechnung des Drucks)

var i, j, t: Integer;

begin
  SetLength(pmat,SIZE);
  for i := 0 to SIZE-1 do
    SetLength(pmat[i],SIZE);
  for i := 0 to SIZE-1 do
    for j := 0 to SIZE-1 do
      pmat[i,j] := 0;
  for i := 0 to SIZE_X+1 do
    for j := 0 to SIZE_Y+1 do
      begin
        t := j*(SIZE_X+2) + i;
        if (i=0) or (i=SIZE_X+1) then
          if (j=0) or (j=SIZE_Y+1) then
            pmat[t,t] := 1
          else
            begin
              pmat[t,t] := 1;
              if i=0 then
                pmat[t,t+1] := -1
              else
                pmat[t,t-1] := -1;
            end
          else
            if (j=0) or (j=SIZE_Y+1) then
              begin
                pmat[t,t] := 1;
                if j=0 then
                  pmat[t,t+SIZE_x+2] := -1
                else
                  pmat[t,t-SIZE_X-2] := -1;
              end
            end
          end
        end
      end
    end
  end
end

```

```

else
begin
  pmat[t,t] := 4;
  pmat[t,t-1] := -1;
  pmat[t,t-SIZE_X-2] := -1;
  pmat[t,t+1] := -1;
  pmat[t,t+SIZE_X+2] := -1;
end;
end;
end;

procedure TRechne.pneu; // Berechnung des neuen Drucks

var i, j: Integer;
    lerg: RealVector; // Ergebnisvektor
    lmat: Matrix; // Lineares Gleichungssystem

begin
  // Berechnung der rechten Seiten des Differenzialgleichungssystems
  SetLength(lmat,SIZE);
  for i := 0 to SIZE-1 do
    SetLength(lmat[i],SIZE+1);
  for i := 0 to SIZE-1 do
    for j := 0 to SIZE-1 do
      begin
        lmat[i,j] := pmat[i,j];
      end;
  for i := 0 to SIZE_X+1 do
    for j := 0 to SIZE_Y+1 do
      if (i=0) or (i=SIZE_X+1) then
        if (j=0) or (j=SIZE_Y+1) then
          lmat[j*(SIZE_X+2)+i,SIZE] := 0
        else
          if i=0 then
            case wl of
              1: lmat[j*(SIZE_X+2),SIZE] := -gx*dx;
              2: lmat[j*(SIZE_X+2),SIZE] := -gx*dx-2*visk*u[1,j]/dx;
              3: lmat[j*(SIZE_X+2),SIZE] := 0;
            end
          else
            case wr of
              1: lmat[(j+1)*(SIZE_X+2)-1,SIZE] := gx*dx;
              2: lmat[(j+1)*(SIZE_X+2)-1,SIZE] := gx*dx+2*visk*u[SIZE_x,j]/dx;
              3: lmat[(j+1)*(SIZE_X+2)-1,SIZE] := 0;
            end
          end
        end
      end
    end
  end
end

```

```

else
  if (j=0) or (j=SIZE_Y+1) then
    if j=0 then
      case wt of
        1: lmat[i,SIZE] := -gy*dy;
        2: lmat[i,SIZE] := -gy*dy-2*visk*v[i,1]/dy;
        3: lmat[i,SIZE] := 0;
      end
    else
      case wb of
        1: lmat[SIZE-SIZE_X+i-1,SIZE] := gy*dy;
        2: lmat[SIZE-SIZE_X+i-1,SIZE] := gy*dy+2*visk*v[i,SIZE_Y];
        3: lmat[SIZE-SIZE_X+i-1,SIZE] := 0;
      end
    else
      lmat[j*(SIZE_X+2)+i,SIZE] := -dx/dt*(f[i,j]-f[i-1,j]+g[i,j]-g[i,j-1]);
    SetLength(lerg,SIZE);
    lerg := SolveLinearSystem(lmat,SIZE,SIZE+1);
    for i := 0 to SIZE_X+1 do
      for j := 0 to SIZE_Y+1 do
        p[i,j] := lerg[j*(SIZE_X+2)+i];
      end;
    end;

  procedure TRechne.schritt;

  var i,j: Integer;

begin
  rechnefg;
  pneu; //Berechnung des Drucks
  // Berechnung der neuen Geschwindigkeitskomponenten
  for i := 1 to SIZE_X do
    for j := 1 to SIZE_Y do
      begin
        u[i,j] := f[i,j] - dt/dx*(p[i+1,j]-p[i,j]);
        v[i,j] := g[i,j] - dt/dy*(p[i,j+1]-p[i,j]);
      end;
    alpha := alphaneu;
    randwerte;
  end;

  procedure TRechne.randwerte;

  var i, j, gl, gr: Integer;

```

```
begin
  case wl of
    1: for j := 0 to SIZE_Y+1 do
      begin
        u[0,j] := 0;
        v[0,j] := v[1,j];
      end;
    2: for j := 0 to SIZE_Y+1 do
      begin
        u[0,j] := 0;
        v[0,j] := -v[1,j];
      end;
    3: for j := 0 to SIZE_Y+1 do
      begin
        u[0,j] := u[1,j];
        v[0,j] := v[1,j];
      end;
  end;
  case wr of
    1: for j := 0 to SIZE_Y+1 do
      begin
        u[SIZE_X,j] := 0;
        v[SIZE_X+1,j] := v[SIZE_X,j];
      end;
    2: for j := 0 to SIZE_Y+1 do
      begin
        u[SIZE_X,j] := 0;
        v[SIZE_X+1,j] := -v[SIZE_X,j];
      end;
    3: for j := 0 to SIZE_Y+1 do
      begin
        u[SIZE_X,j] := u[SIZE_X-1,j];
        v[SIZE_X+1,j] := v[SIZE_X,j];
      end;
  end;
  case wt of
    1: for i := 0 to SIZE_X+1 do
      begin
        u[i,0] := u[i,1];
        v[i,0] := 0;
      end;
    2: for i := 0 to SIZE_X+1 do
      begin
        u[i,0] := -u[i,1];
        v[i,0] := 0;
      end;
  end;
```

```

    end;
3: for i := 0 to SIZE_X+1 do
    begin
        u[i,0] := u[i,1];
        v[i,0] := v[i,1];
    end;
end;
case wb of
1: for i := 0 to SIZE_X+1 do
    begin
        u[i,SIZE_Y+1] := u[i,SIZE_Y];
        v[i,SIZE_Y] := 0;
    end;
2: for i := 0 to SIZE_X+1 do
    begin
        u[i,SIZE_Y+1] := -u[i,SIZE_Y];
        v[i,SIZE_Y] := 0;
    end;
3: for i := 0 to SIZE_X+1 do
    begin
        u[i,SIZE_Y+1] := u[i,SIZE_Y];
        v[i,SIZE_Y] := v[i,SIZE_Y-1];
    end;
end;
// problemabhängige randwerte
for i := 1 to SIZE_X do
    u[i,0] := 2-u[i,1];
end;

procedure TRechne.rechnefg;

var i,j: Integer;

begin
    for i := 1 to SIZE_X do
        for j := 1 to SIZE_Y do
            begin
                f[i,j] := u[i,j] + dt*(visk*(d2udx2(i,j)+d2udy2(i,j))-du2dx(i,j)-duvdy(i,j)+gx);
                g[i,j] := v[i,j] + dt*(visk*(d2vdx2(i,j)+d2vdy2(i,j))-dudvx(i,j)-dv2dy(i,j)+gy);
            end;
        // Randwerte
        for j := 1 to SIZE_Y do
            begin
                case wl of
                    1: f[0,j] := u[0,j] + dt*gx;

```

```

2: f[0,j] := u[0,j] + dt*(gx+2*visk*u[1,j]/Sqr(dx));
3: f[0,j] := u[0,j];
end;
case wr of
1: f[SIZE_X,j] := u[SIZE_X,j] + dt*gx;
2: f[SIZE_X,j] := u[SIZE_X,j] + dt*(gx+2*visk*u[SIZE_X-1,j]/Sqr(dx));
3: f[SIZE_X,j] := u[SIZE_X,j];
end;
end;
for i := 1 to SIZE_X do
begin
case wt of
1: g[i,0] := v[i,0] + dt*gy;
2: g[i,0] := v[i,0] + dt*(gy+2*visk*v[i,1]/Sqr(dy));
3: g[i,0] := v[i,0];
end;
case wb of
1: g[i,SIZE_Y] := v[i,SIZE_Y] + dt*gy;
2: g[i,SIZE_Y] := v[i,SIZE_Y] + dt*(gy+2*visk*v[i,SIZE_Y-1]/Sqr(dx));
3: g[i,SIZE_Y] := v[i,SIZE_Y];
end;
end;
end;

function TRechne.alphaneu: Extended;

var i,j: Integer;
    max: Extended;

begin
max := 0;
for i := 0 to SIZE_X+1 do
for j := 0 to SIZE_Y+1 do
begin
if Abs(u[i,j]) > max then
max := Abs(u[i,j]);
if Abs(v[i,j]) > max then
max := Abs(v[i,j]);
end;
alphaneu := dx/dt*max + 0.1;
end;

function TGraphik.Max1: Extended;

var i,j: Integer;

```

```

    lmax, lerg: Extended;

begin
    lmax := 0;
    for i := 0 to SIZE_X+1 do
        for j := 0 to SIZE_Y+1 do
            begin
                lerg := Sqrt(Sqr(u[i,j]) + Sqr(v[i,j]));
                if lerg > lmax then
                    lmax := lerg;
            end;
        Max1 := lmax;
    end;

procedure TGraphik.Pfeil(xs, ys, xe, ye: Integer);

var lx, ly: Integer;

begin
    Form1.Canvas.MoveTo(xs,ys);
    Form1.Canvas.LineTo(xe,ye);
    lx := xs - xe;
    ly := ys - ye;
    Form1.Canvas.LineTo(xe + Round((lx*Sqrt(3)/2 - ly/2)/3),
        ye + Round((lx/2 + ly*Sqrt(3)/2)/3));
    Form1.Canvas.MoveTo(xe,ye);
    Form1.Canvas.LineTo(xe + Round((lx*Sqrt(3)/2 + ly/2)/3),
        ye + Round((-lx/2 + ly*Sqrt(3)/2)/3));
end;

procedure TGraphik.Draw;

var lmax, lfx, lfy, lx, ly: Extended;
    i,j, xs, ys, xe, ye: Integer;

begin
    lmax := Max1;
    lfx := (Form1.ClientWidth - 100) / (SIZE_X+2);
    lfy := (Form1.ClientHeight - 100) / (SIZE_Y+2);
    if lfx < lfy then
        lfy := lfx
    else
        lfx := lfy;
    Form1.Canvas.Pen.Color := clBlack;
    Form1.Canvas.Brush.Color := clBtnFace;

```

```

Form1.Canvas.FillRect(Rect(-1,-1,Form1.ClientWidth+1,Form1.ClientHeight+1));
for i := 0 to SIZE_X+1 do
  for j := 0 to SIZE_Y+1 do
    Form1.Canvas.Rectangle(Round(50+i*lfx),Round(50+j*lfy),
      Round(51+(i+1)*lfx),Round(51+(j+1)*lfy));
  end;
end;

for i := 0 to SIZE_X+1 do
  for j := 0 to SIZE_Y+1 do
    if (u[i,j] <> 0) or (v[i,j] <> 0) then
      begin
        lx := u[i,j]*lfx/lmax;
        ly := v[i,j]*lfy/lmax;
        xs := Round((50 + (i+0.5)*lfx) - lx*0.5);
        ys := Round((50 + (j+0.5)*lfy) - ly*0.5);
        xe := Round((50 + (i+0.5)*lfx) + lx*0.5);
        ye := Round((50 + (j+0.5)*lfy) + ly*0.5);
        Pfeil(xs,ys,xe,ye);
      end;
    end;
  end;
end;

Form1.Label1.Refresh;
end;

procedure TForm1.init;

var i, j: Integer;

begin
  r1 := TRechne.Create;
  z1 := TGraphik.Create;
  SetLength(u,SIZE_X+2);
  SetLength(v,SIZE_X+2);
  SetLength(p,SIZE_X+2);
  for i := 0 to SIZE_X+1 do
    begin
      SetLength(u[i],SIZE_Y+2);
      SetLength(v[i],SIZE_Y+2);
      SetLength(p[i],SIZE_Y+2);
    end;
  end;
  for i := 0 to SIZE_X+1 do
    for j := 0 to SIZE_Y+1 do
      begin
        u[i,j] := 0;
        v[i,j] := 0;
        p[i,j] := 0;
      end;
    end;
  end;
  r1.randwerte;
end;

```

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  init;
  z1.Draw;
end;
```

```
procedure TForm1.Button2Click(Sender: TObject);
```

```
var i: Integer;
```

```
begin
  for i := 1 to StrToInt(Edit1.Text) do
    r1.schritt;
  z1.Draw;
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  init;
  z1.Draw;
end;
```

```
procedure TForm1.FormResize(Sender: TObject);
begin
  z1.Draw;
end;
```

```
end.
```