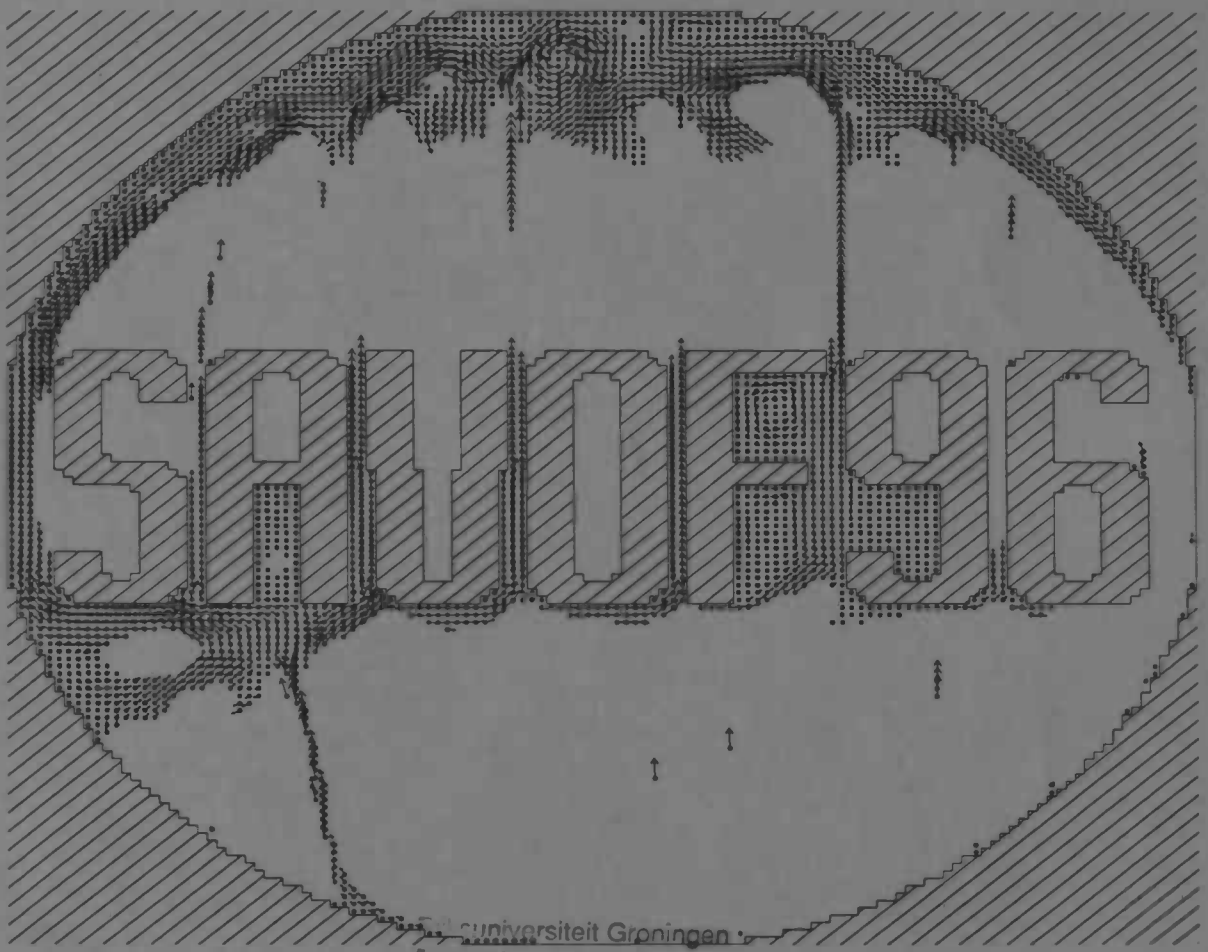
SAVOF96 - Simulation of free-surface liquid dynamics in moving complex geometries

B. de Groot



Department of Mathematics

RuG

Master's thesis

SAVOF96 - Simulation of free-surface liquid dynamics in moving complex geometries

B. de Groot

University of Groningen
Department of Mathematics
P.O. Box 800
9700 AV Groningen

August 1996

Preface

This thesis is a report of my graduation work at the mathematics department of the University of Groningen. I started working on the assignment last april. During this period, I have been supported very well by Prof. Dr A.E.P. Veldman and Ir. J.J. Nies, to whom I wish to express my sincere gratitude. They did not only support me with very good advice, but inspired and kept me on track by repeatedly displaying their enthousiasm and interest. Further, I wish to thank fellow graduate student Jan-Willem Phylipsen for his practical help concerning computer-related problems.
This writing has become a detailed description of the computer program SAVOF96 and can actually be considered a manual. It has been written in a short, to-the-point way which hopefully clarifies the essence without omitting important details.
I wish the reader a pleasant time studying it.

Bart de Groot, Groningen
August 1996

Contents

1 Introduction 3

2 Mathematical Model 4
- 2.1 Navier-Stokes equations . . . 4
- 2.2 The axisymmetric case . . . 5

3 Numerical Model 7
- 3.1 Discretization and solving . . . 7
  - 3.1.1 The Poisson Equation . . . 7
  - 3.1.2 Spatial Discretization . . . 8
  - 3.1.3 Iteration - SOR . . . 9
  - 3.1.4 Description of a free surface . . . 9
  - 3.1.5 Boundary conditions . . . 10
- 3.2 Functional extensions of SAVOF . . . 11
  - 3.2.1 In- and outflow . . . 11
  - 3.2.2 Obstacles . . . 12
  - 3.2.3 Extra motion . . . 14
  - 3.2.4 Increased output capabilities . . . 15

4 Results 20
- 4.1 Testcases . . . 20
  - 4.1.1 In- and outflow tests . . . 20
  - 4.1.2 Toricelli's Theorem . . . 23
  - 4.1.3 Flow over a weir . . . 24
- 4.2 Demonstrations . . . 26
  - 4.2.1 Dambreak problem . . . 26
  - 4.2.2 Vortex formation . . . 27
  - 4.2.3 Oscillating container . . . 27

5 Conclusions and recommendations 29

A Program description 30
- A.1 Calling sequence . . . 30
- A.2 Common block variables . . . 31
- A.3 Subroutines . . . 33
- A.4 Files . . . 34
  - A.4.1 Input files . . . 34
  - A.4.2 Output files . . . 37

Chapter 1

Introduction

In all day life we are surrounded by flows in all sorts of media: draft through the house, water flow in the sink, blood flow through our veins. The world would not look the same without the current understanding of these flows. The designs of aeroplanes and cars for example, are mainly based on our knowledge of fluid mechanics. Often, experiments in windtunnels are performed to test such designs. Experiments are not always possible however. Before a spacecraft is sent into space, some understanding of its behaviour with respect to the contained fluids is required. Since experiments under low g conditions are difficult to perform on earth, another way of predicting the behaviour would be valuable. Also, experiments don't always allow the desired flow pattern to be observed. For example, the sloshing of fluid in an oscillating part of a mechanical system that cannot be opened to study during operation is hard to monitor, sometimes even by experimental means. This is where mathematics and computer science enter the picture.

A means of predicting fluid flow is available in the form of the *Navier-Stokes* equations (see chapter 2). These are second order partial differential equations that usually can't be solved analytically. They can be simplified in order to handle certain problems 'by hand', but in most cases the flow is too complex, and numerical approximation is required. For this purpose SAVOF was conceived. It can simulate fluid flow in two dimensions or in axisymmetric geometries.

SAVOF has been developed in the period 1980-1986 at the National Aerospace Laboratory (NLR) in Amsterdam with support from the Netherland's Agency of Aerospace Programs (NIVR). A detailed description of its capabilities as of 1986 can be found in [6]. Recently, the need arose to revitalise the program for new applications in the area of micro-g liquid management involving the experimental satellite *SloshSat*. New features were added and for post-processing purposes software was developed.

This report describes the underlying mathematics (chapter 2), the numerical model used to solve the governing equations (chapter 3), and the extensions that were implemented (section 3.2). Further, the testing of (the functional extensions to) SAVOF is discussed in chapter 4. In the Appendix, a description of the program can be found.

Chapter 2

Mathematical Model

2.1 Navier-Stokes equations

Fluid motion is dictated by the incompressible Navier-Stokes equations, which consist of two mechanical conservation laws:

- conservation of mass,
- conservation of momentum.

In this report we shall use the following notation (two dimensions) :

$$\begin{aligned} \boldsymbol{x} = \begin{pmatrix} x \\ y \end{pmatrix} &: \text{ place} \\ \boldsymbol{u} = \begin{pmatrix} u \\ v \end{pmatrix} &: \text{ velocity} \\ t &: \text{ time} \\ \rho(x,y) &: \text{ density} \\ p(x,y) &: \text{ pressure} \end{aligned}$$

The above conservation laws can be described by mathematical equations as follows.

Conservation of mass:

$$\frac{\partial \rho}{\partial t} + \frac{\partial(\rho u)}{\partial x} + \frac{\partial(\rho v)}{\partial y} = 0.$$

Conservation of momentum:

$$\begin{aligned} \frac{\partial(\rho u)}{\partial t} + \frac{\partial(\rho u^2)}{\partial x} + \frac{\partial(\rho uv)}{\partial y} &= \rho F_x + \frac{\partial \sigma_{xx}}{\partial x} + \frac{\partial \sigma_{xy}}{\partial y} \\ \frac{\partial(\rho v)}{\partial t} + \frac{\partial(\rho uv)}{\partial x} + \frac{\partial(\rho v^2)}{\partial y} &= \rho F_y + \frac{\partial \sigma_{yx}}{\partial x} + \frac{\partial \sigma_{yy}}{\partial y}. \end{aligned}$$

F_x and F_y are the components of an external force in x and y direction respectively and $\sigma = (\sigma_{ij})$ is the stress tensor. For incompressible Newtonian fluids, it is known that the stress is linearly

related to the rate of deformation. This is the case we study here, so we take

$$\begin{pmatrix} \sigma_{xx} & \sigma_{xy} \\ \sigma_{yx} & \sigma_{yy} \end{pmatrix} = \begin{pmatrix} -p & 0 \\ 0 & -p \end{pmatrix} + \mu \begin{pmatrix} 2\frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \\ \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} & 2\frac{\partial v}{\partial y} \end{pmatrix}.$$

Using this definition of σ, the 2D incompressible Navier-Stokes equations to be solved, become:

$$\begin{aligned} \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} &= 0, \\ \frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} &= F_x - \frac{1}{\rho}\frac{\partial p}{\partial x} + \nu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right), \\ \frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} &= F_y - \frac{1}{\rho}\frac{\partial p}{\partial y} + \nu\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right). \end{aligned}$$

Where the kinematic viscosity $\nu = \frac{\mu}{\rho}$ was introduced.
In later chapters it is convenient to use the Navier-Stokes equations in their vector notation:

$$\begin{aligned} \text{div } \boldsymbol{u} &= 0, \\ \frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u} \cdot \text{grad})\ \boldsymbol{u} &= \boldsymbol{F} - \frac{1}{\rho}\text{grad } p + \nu \text{ div grad } \boldsymbol{u}. \end{aligned}$$

Evidently, the above equations have to be completed by boundary conditions. Usually, for viscous fluids, these will be $\boldsymbol{u} = \boldsymbol{0}$ on the boundary. This describes two phenomena: impermeability of the solid wall and the fact that the fluid 'sticks' to the wall due to its viscosity. The latter is often referred to as the *no-slip* condition. In section 3.1.5 of this report special attention is paid to the implementation of the boundary conditions.

2.2 The axisymmetric case

As mentioned in the introduction, SAVOF is capable of simulating fluid flow in axisymmetric containers as well. Compared to the two dimensional case of the previous section, an axisymmetric calculation requires an extra dimension: the *azimuthal direction*. For the axisymmetric formulation of the model, cylindrical co-ordinates (r, φ, z) are used. Further, w is introduced for the azimuthal velocity. Using this notation, the equation for the conservation of mass becomes:

$$\frac{1}{r}\frac{\partial (ru)}{\partial r} + \frac{\partial v}{\partial z} = 0.$$

The momentum equations for the radial, axial and azimuthal direction respectively, become:

$$\begin{aligned} \frac{\partial u}{\partial t} + u\frac{\partial u}{\partial r} + v\frac{\partial u}{\partial z} - \frac{w^2}{r} &= F_r - \frac{1}{\rho}\frac{\partial p}{\partial r} + \nu\left(\frac{1}{r}\frac{\partial}{\partial r}(r\frac{\partial u}{\partial r}) + \frac{\partial^2 u}{\partial z^2} - \frac{u}{r^2}\right), \\ \frac{\partial v}{\partial t} + u\frac{\partial v}{\partial r} + v\frac{\partial v}{\partial z} &= F_z - \frac{1}{\rho}\frac{\partial p}{\partial z} + \nu\left(\frac{1}{r}\frac{\partial}{\partial r}(r\frac{\partial v}{\partial r}) + \frac{\partial^2 v}{\partial z^2}\right), \\ \frac{\partial w}{\partial t} + u\frac{\partial w}{\partial r} + v\frac{\partial w}{\partial z} + \frac{uw}{r} &= F_\varphi + \nu\left(\frac{1}{r}\frac{\partial}{\partial r}(r\frac{\partial w}{\partial r}) + \frac{\partial^2 w}{\partial z^2} - \frac{w}{r^2}\right). \end{aligned}$$

Where again, the F's contain the external forces. In this case however, they also include the centrifugal and rotational terms due to possible rotation of the tank (see also section 3.2.3).

Rotation of the tank about its axis with angular velocity ω results in a centrifugal term $\omega^2 r + 2\omega w$ and a rotational term $-2\omega u - \frac{d\omega}{dt} r$ to be added to the right-hand side of the momentum equations in radial and azimuthal direction respectively.

Chapter 3

Numerical Model

In this chapter the numerical model used by SAVOF is discussed. A section is devoted to the functional extensions that were implemented.

3.1 Discretization and solving

3.1.1 The Poisson Equation

SAVOF computes the pressure by means of a Poisson equation 3.3 that follows from the incompressible version of the Navier-Stokes equations. If we use the abbreviation

$$\boldsymbol{R} = -(\boldsymbol{u} \cdot \text{grad})\boldsymbol{u} + \nu \text{ div grad } \boldsymbol{u} + \boldsymbol{F},$$

and set $\rho = 1$, we can write the Navier-Stokes equations as:

$$\begin{aligned} \text{div } \boldsymbol{u} &= 0, \\ \frac{\partial \boldsymbol{u}}{\partial t} + \text{grad } p &= \boldsymbol{R}. \end{aligned}$$

The $\frac{\partial \boldsymbol{u}}{\partial t}$ term is discretized in time with a forward Euler method, yielding

$$\text{div } \boldsymbol{u}^{n+1} = 0, \tag{3.1}$$

$$\frac{\boldsymbol{u}^{n+1} - \boldsymbol{u}^n}{\delta t} + \text{grad } p^{n+1} = \boldsymbol{R}^n. \tag{3.2}$$

δt denotes the time step, n denotes the 'old' time level and $n+1$ the 'new' one. Rearranging terms gives:

$$\boldsymbol{u}^{n+1} = \boldsymbol{u}^n + \delta t \ \boldsymbol{R}^n - \delta t \text{ grad } p^{n+1}.$$

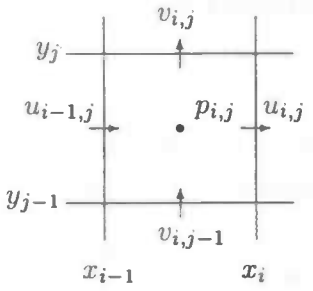
Substituting this into (3.1), we obtain

$$\text{div grad } p^{n+1} = \text{div } (\frac{\boldsymbol{u}^n}{\delta t} + \boldsymbol{R}^n). \tag{3.3}$$

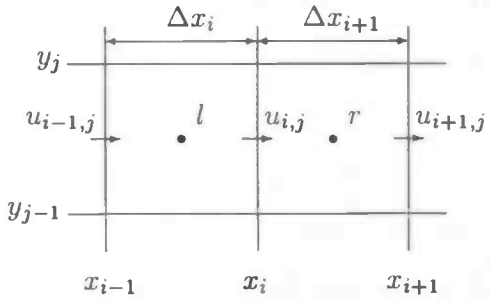
This is called the *Poisson equation* for the pressure.

3.1.2 Spatial Discretization

SAVOF uses a Cartesian grid for its discretization, with optional refinements near the outer walls. The unknown variables u, v and p are placed as follows: the horizontal velocity u on the vertical faces of a cell, the vertical velocity v on the horizontal faces, and the pressure p in the center. This is the well known MAC (Marker-And-Cell) method. It has an important benifit compared to other methods of placement that suffer problems as to the uniqueness of the pressure.



The Poisson equation is solved in two steps. First, in subroutine TILDE [1], the momentum equation is integrated, i.e. the part $\frac{\boldsymbol{u}^n}{\delta t} + \boldsymbol{R}^n$ is calculated. We will show how this is done for the x-direction. First the viscous terms in $\boldsymbol{R}^n$: $\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$.



The second order derivative is discretized centrally. First:

$$\left.\frac{\partial u^n_{i,j}}{\partial x}\right|_l = \frac{u^n_{i,j} - u^n_{i-1,j}}{\Delta x_i},$$
$$\left.\frac{\partial u^n_{i,j}}{\partial x}\right|_r = \frac{u^n_{i+1,j} - u^n_{i,j}}{\Delta x_{i+1}}.$$

Then these are used to form

$$\frac{\partial^2 u^n_{i,j}}{\partial x^2} = \frac{\left.\frac{\partial u^n_{i,j}}{\partial x}\right|_r - \left.\frac{\partial u^n_{i,j}}{\partial x}\right|_l}{\frac{1}{2}(\Delta x_i + \Delta x_{i+1})}.$$

[1] For a global description of the subroutine structure of SAVOF, see the Appendix

That takes care of the diffusive terms, now the convective terms $u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y}$. These are treated with upwind discretisation which is controlled by the upwind parameter α ($\alpha = 1$ is full upwind).

$$u_{i,j}^n \frac{\partial u_{i,j}^n}{\partial x} = \begin{cases} \frac{u}{(1+\alpha)\Delta x_i + (1-\alpha)\Delta x_{i+1}} \left((1-\alpha)\Delta x_{i+1} \left.\frac{\partial u_{i,j}^n}{\partial x}\right|_r + (1+\alpha)\Delta x_i \left.\frac{\partial u_{i,j}^n}{\partial x}\right|_l \right), & u \geq 0 \\ \frac{u}{(1-\alpha)\Delta x_i + (1+\alpha)\Delta x_{i+1}} \left((1+\alpha)\Delta x_{i+1} \left.\frac{\partial u_{i,j}^n}{\partial x}\right|_r + (1-\alpha)\Delta x_i \left.\frac{\partial u_{i,j}^n}{\partial x}\right|_l \right), & u < 0 \end{cases}$$

If the same is done for the y-direction and for v, the Poisson equation can be solved by an iterative proces.

3.1.3 Iteration - SOR

For presentational reasons, let's write the Poisson equation like

$$Ax = b,$$

where

$$A = \text{ div grad }, \; x = p^{n+1}, \; b = \text{ div } (\frac{\boldsymbol{u}^n}{\delta t} + \boldsymbol{R}^n).$$

SAVOF uses a *Red-Black* or *Checkerboard ordening*, that is the pressure matrix is divided into 'red' and 'black' grid points. These points are stored into one-dimensional vectors which enhances vectorization. Let D be a vector containing the diagonal elements of A, L the lower triangular matrix of $-A$ and U the upper triangular matrix of $-A$ (that is $A = D - L - U$). The SOR (Successive OverRelaxation) method with relaxation parameter ω is defined as

$$x^{(k+1)} = C_\omega x^{(k)} + Q^{-1} b,$$

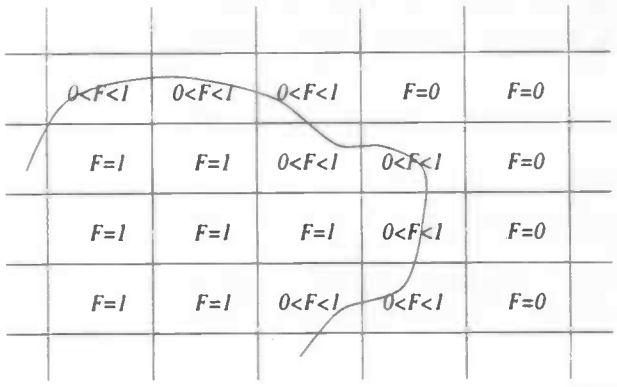
where $C_\omega = (D - \omega L)^{-1}((1-\omega)D + \omega U)$, and $Q = \frac{1}{\omega}D - L$. This equation is solved in subroutine SOLVEP, automatically adjusting ω for optimal convergence following a method originally designed by Botta and Ellenbroek (see [1]). This gives the pressure p^{n+1}. Subsequently, the new velocity can be calculated using $u_{i,j}^{n+1} = (u_{i,j}^n + \delta t R_{i,j}^n) - \delta t \text{ grad } p_{i,j}^{n+1}$. The gradient is discretized as

$$\frac{p_{i+1,j}^{n+1} - p_{i,j}^{n+1}}{\frac{1}{2}\Delta x_i + \frac{1}{2}\Delta x_{i+1}}.$$

This is also done in SOLVEP. SOR is a fairly simple process and there are better algorithms that are, however, more difficult to implement. Besides: the performance of SOR for simple cases (that require only a few iterations) is very good, while other -more complex methods- require a considerable amount of iterations before the complexity starts to pay off. Another advantage of SOR is that it vectorizes and parallelizes excellently (see [7]).

3.1.4 Description of a free surface

A means of keeping track of the free surface of a fluid during calculation is necessary to know where to apply the momentum equation. Evidently, this only has to be done at cells that contain fluid. In SAVOF, use is made of an *indicator function*. This is a function $F(i,j)$ that is 1 in the case that cell (i,j) is completely filled with fluid, and 0 if it is empty. Also values in between can be attained, depending on the percentage of the cell that is filled. Besides the indicator function, there is the cell labeling, which gives more qualitative information about a cell.
`NF(i,j)` is a two-dimensional array in which this information for cell `(i,j)` is stored. It can have the following values:



- 0: full cell,
- 1: surface cell with full cell to the left,
- 2: surface cell with full cell to the right,
- 3: surface cell with full cell at the bottom,
- 4: surface cell with full cell at the top,
- 5: degenerated cell,
- 6: empty cell,
- 7: 'outflow' cell,
- 8: 'inflow' cell,
- 9: obstacle or boundary cell.

Using the above labeling, the position where the momentum equations must be applied can be identified: at cell faces between full and/or surface and/or outflow cells. At faces between surface and empty cells boundary conditions are applied, to be described in the next section. For more details concerning the labeling and the displacement of the free liquid surface, see e.g. [5] or [6].

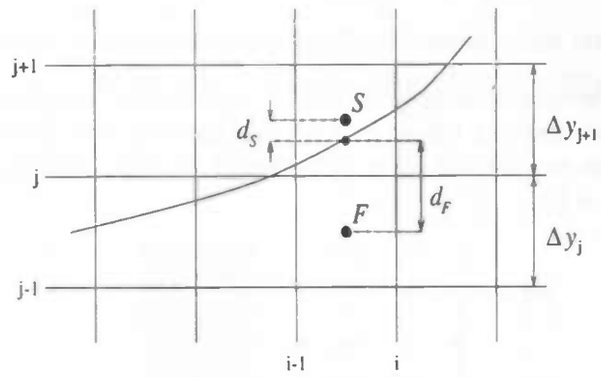
3.1.5 Boundary conditions

As noted earlier, the boundary conditions for the Navier-Stokes equations usually are $u = v = 0$ on solid walls (no-slip condition). Since the velocities are not always defined on the walls (or objects) because of the placement of the variables, we have to interpolate and make use of virtual mirror points. For example, consider the lower boundary ($i = 1$). The horizontal velocity u is not defined *on* the wall, but half a grid point higher. So what we do is, we set $u(0, j) = -u(1, j)$ so that after interpolation of these two velocities the resulting velocity on the wall is zero.
At the free surface of a fluid, the governing boundary conditions are:

$$-p + 2\mu \frac{\partial u_n}{\partial n} = -p_0 + 2\gamma H, \tag{3.4}$$

$$\mu \left(\frac{\partial u_n}{\partial t} + \frac{\partial u_t}{\partial n} \right) = 0. \tag{3.5}$$

Here u_n denotes the velocity in the direction normal to the free surface, u_t the velocity in tangential direction; p_0 represents the pressure of the surrounding gas, γ is the surface tension and H is the curvature of the surface. The curvature is calculated using information from the indicator function.



The pressure at the free surface needed in (3.4) is interpolated as follows. Suppose we have a full cell ($F_{i,j} = 1$, pressure p_F) with above it a cell that is only partially filled ($0 < F_{i,j+1} < 1$, pressure p_S). Then we can estimate the average height of the surface in the upper cell to be $F_{i,j+1}\Delta y_{j+1}$, so the distance from the center of the upper cell to the surface is $d_S := \frac{1}{2}\Delta y_{j+1} - F_{i,j+1}\Delta y_{j+1}$ and the distance from the center of the lower cell to the surface is $d_F := \frac{1}{2}\Delta y_j + F_{i,j+1}\Delta y_{j+1}$. Since the distance between the centers of the cells is $d := \frac{1}{2}\Delta y_j + \frac{1}{2}\Delta y_{j+1}$, we can now linearly interpolate the pressures in the two cells to obtain the pressure at the free surface:

$$p_f = \frac{d_S p_F + d_F p_S}{d}.$$

Because the momentum equations have to be calculated in between surface cells as well, we need to establish extra velocities on those cell faces. To overcome this problem, we apply the continuity equation div $u = 0$ in surface cells as well. If we have three velocities available, then this procedure gives the fourth. If less velocities are available, we take e.g. $\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y} = 0$. The remaining velocities required, lying just outside the surface cell layer, are calculated using (3.5).

3.2 Functional extensions of SAVOF

In this section some additions to the original version of SAVOF (see [6]) are discussed. These extensions have been made to make the program more suitable as a tool for calculating 'real' problems.

3.2.1 In- and outflow

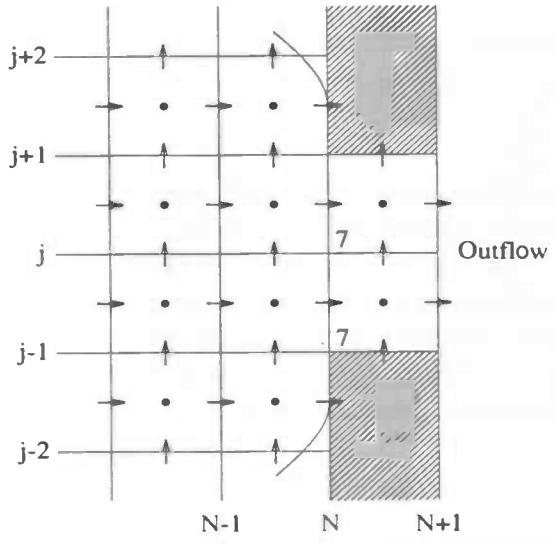
Applications with varying amount of fluid contained in the space considered are abundant: flow through a pipe, over a weir, the filling of a tank etc. So the need arose to add or discharge fluid from the geometry by means of in- and outflow openings. This has implications for the boundary conditions at those points.

The cells with the newly introduced label `NF=8` form the inflow openings. These cells are filled with fluid ($F_{i,j} = 1$) and the velocity is set at the prescribed value (depending on the orientation of the outer wall in which the opening lies). The inflow cells are further treated the same as boundary cells.

The cells with label NF=7 make up the outflow opening. Such openings are currently restricted to the outer walls. The boundary conditions at such a region of the wall can be chosen to be:

1. set the normal derivatives of the velocity at 0 (suitable for flow through long pipes),
2. prescribe the normal stress (equation (3.4), natural boundary condition).

The last option was implemented, simply because it yielded the best results. In an outflow cell, we set the pressure at its ambient value: $p = 0$ (thus ignoring any viscous contribution to the normal stress; the surface tension influence is neglected as well). Further, the outflow cells are always kept empty ($F_{i,j} = 0$).



Intuitively, it is clear that for example in the figure, the change in vertical velocity is likely to be small as the fluid passes through the opening: $\frac{\partial v}{\partial x} = 0$. As a consequence, because of the continuity equation, $\frac{\partial^2 u}{\partial x^2} = \frac{\partial}{\partial x}\left(\frac{\partial u}{\partial x}\right) = \frac{\partial}{\partial x}\left(-\frac{\partial v}{\partial y}\right) = -\frac{\partial}{\partial y}\left(\frac{\partial v}{\partial x}\right) = 0$. This is discretized centrally, resulting in for example $u_{N+1,j} = 2u_{N,j} - u_{N-1,j}$. This velocity is necessary to compute the momentum equation at the outflow cell faces which are treated the same as all other internal cell faces. This new feature has been tested, as described in section 4.1.1.

3.2.2 Obstacles

In many cases we are interested in the behaviour of a fluid that flows or sloshes in a space of complex geometry, containing all sorts of obstacles. In order to input such obstacles into SAVOF some way of describing them had to be developed. An extra input file was designed in which this could be done in a rather convenient fashion. Each geometry boundary is considered to consist of a combination of elementary objects: circular arcs and lines. The geometry file (**geo#.in**, with # a number 1..9 to be specified in the main input file) has the following layout:

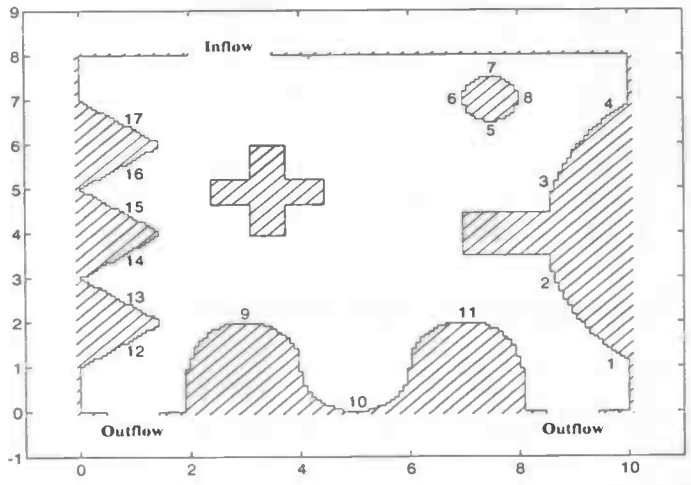
```
WEIRD GEOMETRY EXAMPLE
iCyl    xMin    xMax    yMin    yMax
   0    0.00   10.00    0.00    8.00
number of: arcs    lines    inflows    outflows
                5       11          1           2
arcs:
   xm      ym      r    teta1    teta2  side  left  right  top  bottom  m-points
```

```
12.00    4.00   3.50   124.85   235.15      0      0      1     0        0          4
 7.50    7.00   0.50     0.00   360.00      0      0      0     0        0          4
 3.00    1.00   1.00     0.00   180.00      0      0      0     0        0          1
 5.00    1.00   1.00   180.00   360.00      1      0      0     0        1          1
 7.00    1.00   1.00     0.00   180.00      0      0      0     0        0          1
lines:
   x1      y1     x2      y2   side   left   right   top   bottom   m-points
 0.00    1.00   1.50    2.00      1      1       0     0        0          1
 1.50    2.00   0.00    3.00      0      1       0     0        0          1
 0.00    3.00   1.50    4.00      1      1       0     0        0          1
 1.50    4.00   0.00    5.00      0      1       0     0        0          1
 0.00    5.00   1.50    6.00      1      1       0     0        0          1
 1.50    6.00   0.00    7.00      0      1       0     0        0          1
 2.50    4.75   4.50    5.25      2      0       0     0        0          0
 3.25    4.00   3.75    6.00      2      0       0     0        0          0
 7.00    3.50   7.00    4.50      1      0       1     0        0          0
 2.00    1.00   4.00    1.00      0      0       0     0        1          0
 6.10    1.00   8.10    1.00      0      0       0     0        1          0
inflows:
   p1      p2   side (left=1,right=2,top=3,bottom=4)
 2.00    3.50      3
outflows:
   p1      p2   side (left=1,right=2,top=3,bottom=4)
 0.50    1.50      4
 8.50    9.50      4
```

This is an example of a geometry in a rectangular space of 10x8 units. On the first line, a name for the geometry can be stated. Then, the parameter for two-dimensional (`iCyl=0`) or axisymmetric (`iCyl=1`) geometries is set and the dimensions of the geometry are given. In the next line, the number of objects is defined. Then, per class of objects the objects are defined, beginning with circular arcs, then lines, inflow and outflow openings. The circular arcs are defined by their center (`xm,ym`) (does not necessarily have to lie inside the space of the geometry !), radius `r`, start angle `teta1` and end angle `teta2`. Then the information is given as to where the obstacle cells have to be put: `side=0` results in an arc filled at the side of the center of the circle; `side=0` is used if the opposite side is to be filled. The next four arguments to be given tell SAVOF towards which outer wall the object is to be extended (either `0` or `1`). At the end of the line, the number of measure points must be defined. If greater than 0 SAVOF 'creates' measure points along the object, equally divided over the surface and 1.5 gridpoints away from it. These points are numbered and corresponding files (`speed##.out` with `##` the number) are created on which every time step interpolated velocities are written. This is a feature, added to increase the amount of information available after calculation (a plot of selected velocities as a function of time can be very helpfull). Every line of the '`arcs`' section represents an arc.
Then the lines are defined. A line is described by its begin (`x1,y1`) and end (`x2,y2`). The '`side`' column can have three different values here:

- 0 in the case that the object is situated underneath the line, or -in the case of a vertical line- to the left of the line,
- 1 when the object is above (or -for vertical lines- to the right of) the line,
- 2 if both sides of the line have to be filled (this results in a rectangle).

The other arguments do the same as they do for circular arcs.
Finally, the in- and outflow openings have to be stated. Since these openings can only be situated in the outer walls, three arguments suffice: `p1` and `p2` determine the co-ordinates of the opening, and `side` is used to interpret the meaning of them: if `side=1` or `side=2`, (`p1`,`p2`) are the y-co-ordinates of the opening and for the x-co-ordinates the minimum or the maximum value in horizontal direction is taken. Analogous for `side=3` or `side=4`.
The specific geometry-file used above results in the following geometry (a 128x128 grid was used):



3.2.3 Extra motion

SAVOF used to be equiped with a rotation and an oscillation as external motions. That is, the axisymmetric tank could be spinned around its axis at a certain speed and over a certain period of time. Also, the tank could be moved up and down with arbritrary amplitude and frequency. The accelerations in both x- and y-direction could be prescribed, but were fixed during the entire time-interval.
To make it suited for more general calculations (e.g. analysis of the liquid dynamics inside *Slosh-Sat*), SAVOF was extended with capabilities to simulate more motions that could be controlled more extensively. In the two dimensional case, the accelerations in both directions can now be turned on and off at arbitrary points in time. An initial velocity in both directions can be given (useful if a certain speed is required but the acceleration leading to it isn't interesting - calculating time can be reduced). Also oscillation under an angle was added. Finally, 2D rotation about an arbitrary point was fitted. The latter was done by adding an extra term to $\boldsymbol{R}$, the vector containing the external forces. Let's consider a point P somewhere in the container. We have to identify two co-ordinate systems: one attached to the container (body-fixed co-ordinate sytem) and an inertial reference frame. We use the following notation:

$$r = \begin{pmatrix} r_1 \\ r_2 \\ r_3 \end{pmatrix}: \text{ position of } P \text{ with respect to the moving origin 0,}$$

$v = \begin{pmatrix} u \\ v \\ w \end{pmatrix}$: velocity of P with respect to the body-fixed reference frame,

q_0: velocity of the container with respect to the inertial reference frame,

$\omega = \begin{pmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{pmatrix}$: rotation of the container about its moving origin.

Using this notation, we have for the absolute velocity of P:

$$q = v + q_0 + \omega \times r,$$

and for the absolute acceleration:

$$\frac{\mathrm{D}q}{\mathrm{D}t} = \frac{\mathrm{D}v}{\mathrm{D}t} + \frac{dq_0}{dt} + \omega \times (\omega \times r) + \frac{d\omega}{dt} \times r + 2\,\omega \times v.$$

For 2D rotation of a point (x, y) about a point (x_0, y_0) with angular velocity ω, that is:

$$\omega = \begin{pmatrix} 0 \\ 0 \\ \omega \end{pmatrix}, \text{ and } r = \begin{pmatrix} x - x_0 \\ y - y_0 \\ 0 \end{pmatrix}$$

the last three terms add up to:

$$\begin{pmatrix} -\omega^2(x - x_0) - \frac{\mathrm{d}\omega}{dt}(y - y_0) - 2\omega v \\ -\omega^2(y - y_0) + \frac{\mathrm{d}\omega}{dt}(x - x_0) + 2\omega u \end{pmatrix}$$

These terms were implemented in SAVOF's subroutine TILDE.
In the axisymmetric situation a time interval over which the vertical acceleration can be turned on was added, as well as the possibility of an initial velocity.
With the mentioned additions, most of the occuring motions in practice can be performed by SAVOF.

3.2.4 Increased output capabilities

The visualisation of results of a calculation is something to pay special attention to. For mathematicians this is usually not very interesting, they can deduce enough from a bunch of numbers spitted out by an algorithm they have written. But for students or physicists this is not always true. The visualisation is crucial, especially for engineers aiming on interpreting the outcome of the calculations.
In order to achieve a reasonable visualisation, a lot of postprocessing routines in SAVOF and peripheral software were written.

MATLAB output

To begin with, a routine was implemented that writes all the velocities at equal time intervals to file. For every prescribed number of time steps a new file is created, resulting in the data files `M0001.DAT`...`M0187.DAT` for example. These files can be read into MATLAB and graphically represented by an 'm-file'. Also, to make things 'come alive', a TURBO PASCAL program was conceived, with which movies can be made. The motion of the container is taken into account,

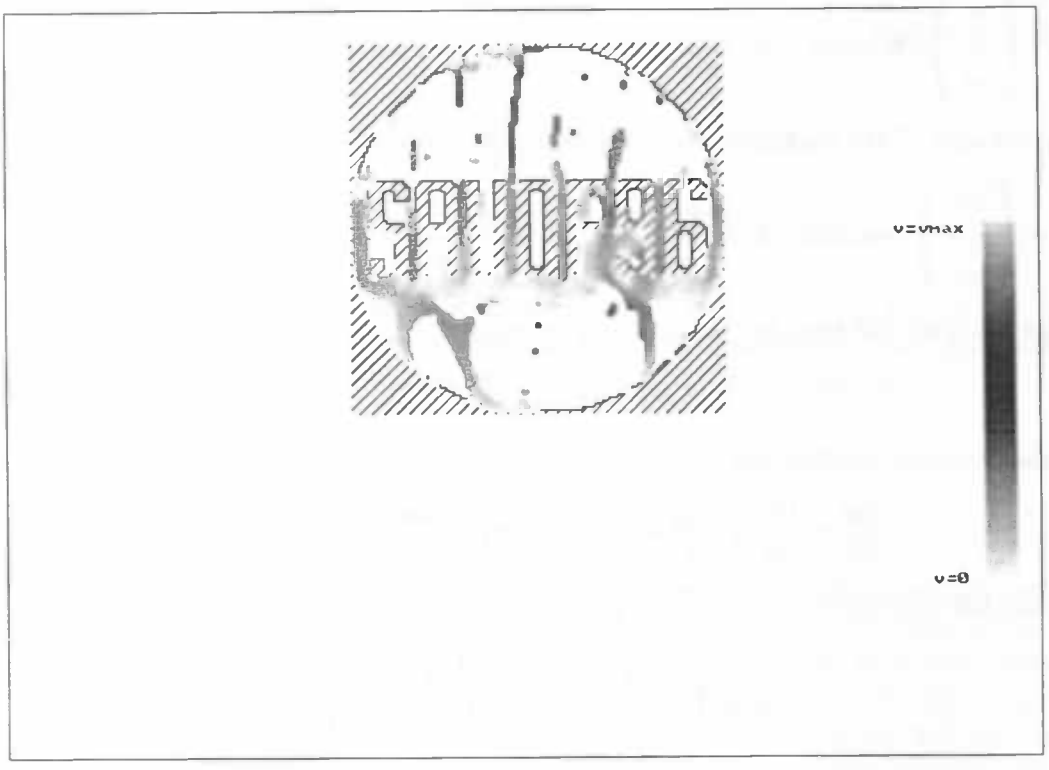


Figure 3.1: Screen dump of a .FLC movie

so that for example, one sees the geometry moving up and down over the screen, making the fluid slosh or flow. This gives a good impression of how far the result looks natural. The movies created are in the AutoDesk .FLC format, so they can be played at several platforms. For an example, see figure 3.1. The bar at the right provides a crude impression of the velocities. The 'particles' have a colour from this bar, depending on their velocity.

Often, a more quantitative analysis is required. To that end SAVOF was extended with the following output options:

- fluxes at several arbitrary places,
- fill ratios of arbitrary regions of the grid,
- forces exerted by the moving fluid in horizontal as well as vertical direction,
- stream lines, i.e. particles released at given time from given points are followed.

Fluxes

The first point, the fluxes, was quite easy to implement, because SAVOF already makes use of the fluxes to and from cells internally. Some scaling factors had to be taken into account in the case of an axial symmetric geometry.

Fill ratios

The fill ratios can be calculated by making use of the indicator function F. Say, we want to monitor the fill ratio of a region bounded by (x_1, y_1) and (x_2, y_2) (which represent the lower left

and the upper right corner respectively). Then the corresponding indices characteristic to the grid are determined: (i_1, j_1) and (i_2, j_2). Then the fill ratio for the 2D geometry simply is:

$$\sum_{i=i_1}^{i_2-1} \sum_{j=j_1}^{j_2-1} s_i F_{i,j},$$

with s_i a scaling factor for the axisymmetric case.

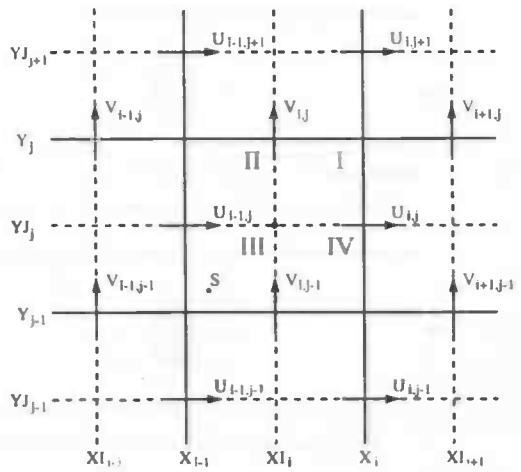
Forces

Sometimes interesting, are the forces that result from the liquid splashing against the walls of the container. The forces in x- as well as y-direction can be calculated by integrating the pressure over the surface of obstacles and outer walls:

$$F_x = \int_{\substack{\text{vertical} \\ \text{walls}}} p_{i,j}, \qquad F_y = \int_{\substack{\text{horizontal} \\ \text{walls}}} p_{i,j}.$$

Streamlines

Of special interest are the paths that particles in the fluid follow. Often, calculations like those executed with SAVOF are the only way of determining such paths. Some means of specifying which particles and over what time intervals the trajectory of particles are to be registered had to be thought of. To the input file a section was added in which this could be realized. As input arguments, the upper-right and lower-left corner of a rectangular region in which the particles of interest lie are given, as well as the number of particles to be traced, and the time to start tracing them.
The initial co-ordinates are determined by equally distributing the given number of particles over the prescribed region. Then, every time step, the velocity of each particle is determined. Since the particles don't necessarily have to lie on grid points, this velocity has to be obtained by interpolation.



Because of the placing of the variables, we have to distinguish four different cases. According to the quadrant the particle is positioned in, a different set of velocities have to be used for

interpolation. For example, let's say a particle S has co-ordinates (x_S, y_S) that puts it into the third quadrant. Then the interpolated velocity (u_S, v_S) becomes:

$$\begin{aligned} u_S &= \frac{(y_S - YJ_{j-1})u_{i-1,j} + (YJ_j - y_S)u_{i-1,j-1}}{YJ_j - YJ_{j-1}}, \\ v_S &= \frac{(x_S - XI_{i-1})v_{i,j-1} + (XI_i - x_S)v_{i-1,j-1}}{XI_i - XI_{i-1}}. \end{aligned}$$

XI and YJ represent virtual gridpoints halfway the usual grid: $XI_i = \frac{1}{2}(X_{i-1} + X_i)$, $YJ_j = \frac{1}{2}(Y_{j-1} + Y_j)$. Evidently, similar formulas hold for the other quadrants. Then these velocities are integrated to determine the position of the particles at the next time level:

$$\begin{aligned} x_S^{n+1} &= x_S^n + \delta t\ u_S, \\ y_S^{n+1} &= y_S^n + \delta t\ v_S. \end{aligned}$$

These co-ordinates are written to a file that can be either interpreted with a MATLAB 'm-file', or a TURBO PASCAL program generating a movie in which the particles move with a trace behind them. In the current version, the tracing of a particle is stopped when it hits an obstacle or a solid wall. An impression of the possibilities is given by the following example. Consider a rectangular area with a sphere in the middle. A uniform flow enters at the left and can exit at the right. Particles are released in front of the sphere: three rows of five particles. At three different times such particles are released (so 45 particles are traced in total). Figure 3.2 displays the resulting streamlines.

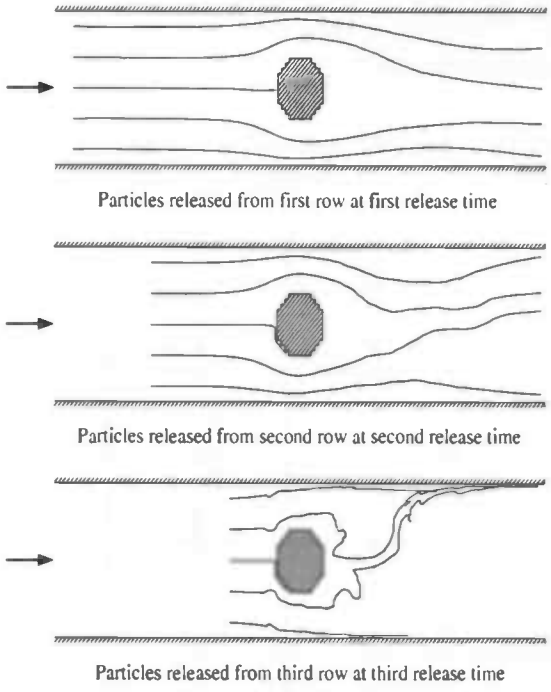


Figure 3.2: Streamlines around a sphere

Saving and restarting

An option was added to save the entire state of SAVOF to file at prescribed points of the calculation. At a later time, the calculation can be performed that has the previously saved state as an initial configuration. The reasons for implementing this option were purely practical. In the case of time-consuming calculations it can be convenient to be able to split the calculation in pieces, e.g. to calculate only at night. Sometimes it is desirable to continue a calculation with changed parameters, so that two different calculations can be obtained that have the first piece in common. Something that also might occur, is that the prescribed initial time step turns out to be too large after a considerable part of the calculation is complete. Since the number of reductions of the time step that is performed by SAVOF is restricted to a maximum, it is conceivable that a computation will cease because the velocities of the fluid reached during computation demand a much smaller time step than the maximum number of reductions allows. In such a case the calculation can be resumed with an adjusted time step, without calculating the part before the error again.

With the implemented additions, it is remarkebly easy to perform calculations and process the results. Some demonstrations of this are given in the next chapter.

Chapter 4

Results

This chapter is devoted to the results obtained with SAVOF. It is divided into two parts: one containing testcases for the features newly introduced, and one part with demonstrations of what SAVOF is capable of.

4.1 Testcases

Testing software is a science in itself, especially if there is little to compare the results to. That is the difficulty here. There is a lot of theory on potential flows. SAVOF however, is not based on potential flow, but actually solves the incompressible Navier-Stokes equations taking viscosity into account. When comparing SAVOF's results with examples that can be checked analytically, this is often visible.

4.1.1 In- and outflow tests

Steady flow through a pipe

The implementation of the in- and outflow condition was tested with the use of a pipe with one wall used for inflow, and one for outflow (see figure 4.1).

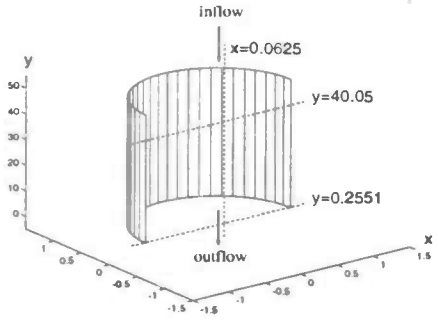


Figure 4.1: Axial symmetric pipe used for in- and outflow test

The vertical velocity was measured at two horizontal cross sections and at one vertical cross section. The velocities at the horizontal cross section indeed show the expected parabolic shape due to the no-slip conditions at the solid walls (see figure 4.2). The pressure difference Δp over the two horizontal sections results in a downward flow.

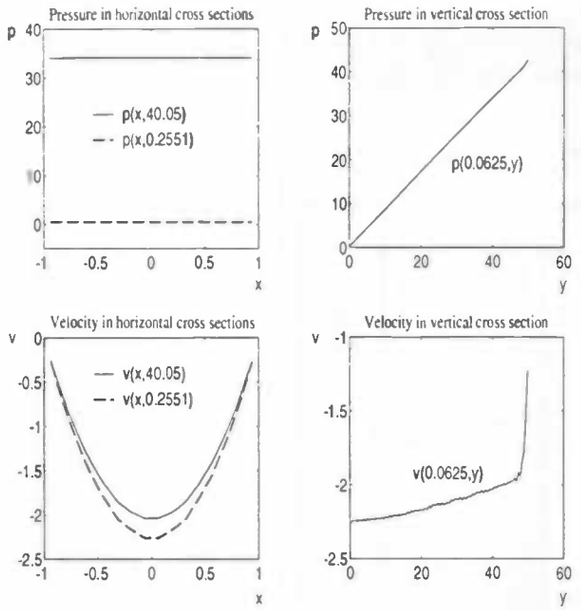


Figure 4.2: Measured pressure and vertical velocities

For a Newtonian fluid the resulting flowrate should be:

$$Q = \frac{\pi D^4 \Delta p}{128 \mu l}.$$

With D the diameter of the pipe (here 2.0), l the distance over which the pressure difference is measured (here 39.7949) and μ the viscosity (taken to be 0.01). The mean Δp calculated by SAVOF is 3.1684, resulting in a flowrate of 3.1266. Since the inflow velocity is -1, and the cross-sectional area is π, the flowrate should be 3.1415, which makes the above 3.1266 an acceptable result.

Falling drop

A very simple test was performed by just letting a drop of liquid fall and then check the height as a function of the elapsed time (see figure 4.3).

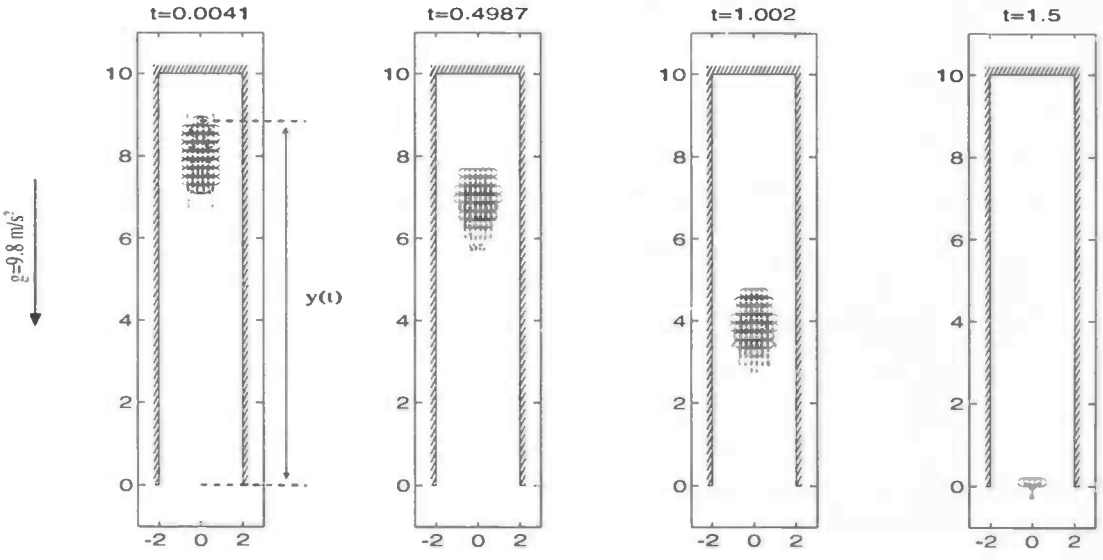


Figure 4.3: Falling drop

The location of the top of the drop should satisfy the elementary mechanics law $y(t) = y_0 + v_0 t + \frac{1}{2} g t^2$. In this case, $y_0 = 10$, $v_0 = 0$ and $g = -9.8$, resulting in graph 4.4, in which the location of the top according to SAVOF is plotted as well. The slight difference is caused by the deformation of the drop, due to the enabled surface tension.

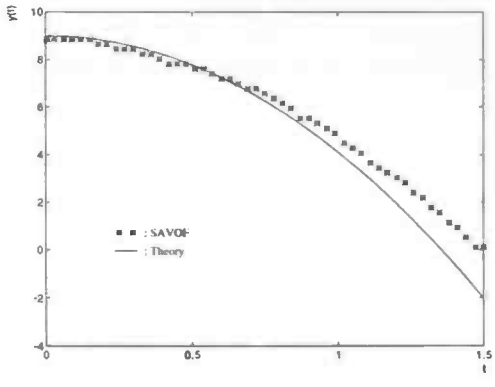


Figure 4.4: Location of the top

4.1.2 Toricelli's Theorem

Consider a container filled with fluid until a certain level. If the container has an outflow opening at a distance h below the surface of the fluid, then the velocity at which the fluid flows out of the container will satisfy (g is the gravitational acceleration):

$$v = \sqrt{2gh}.$$

This is called *Toricelli's Theorem* (see figure 4.5).

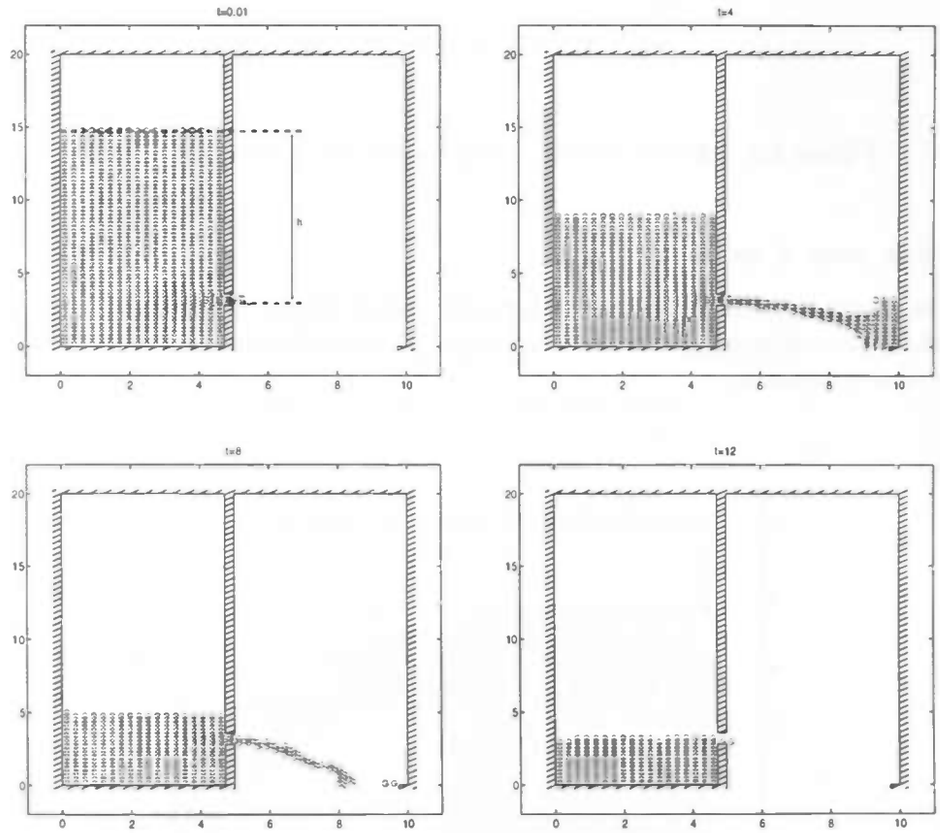


Figure 4.5: Toricelli's theorem

SAVOF's calculation of the velocity at the outflow opening is depicted in figure 4.6. Again, the influence of the no-slip condition is obvious: the velocity is lower due to the viscous effects in the opening. Also, a start-up phenomenon is visible: the velocity is evidently 0 in the beginning (with maximum h) and then rapidly increases towards the theoretical velocity.

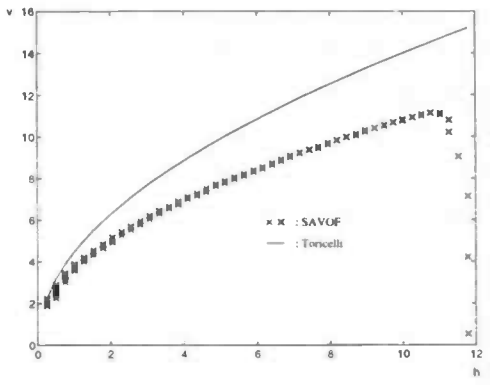


Figure 4.6: SAVOF's velocity compared to the theoretical velocity

4.1.3 Flow over a weir

A *weir* is an obstruction on a channel bottom over which the fluid must flow. Here, we use the so-called *sharp-crested weir*, a vertical, sharp-edged plate the fluid must flow over and then drop into the pool downstream.

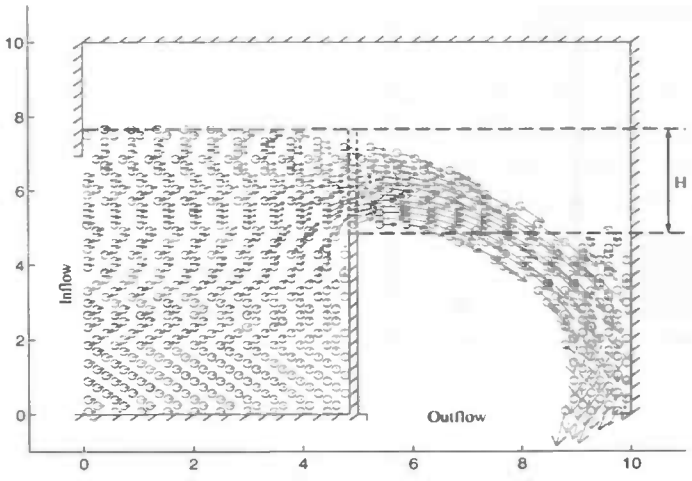


Figure 4.7: Flow over a sharp-crested weir

The Bernoulli equation dictates the horizontal velocity above the plate to be

$$u = \sqrt{2g\left(h + \frac{v_{in}^2}{2g}\right)}.$$

Where h is the distance from the maximal height of the free surface (upper dotted line in figure 4.7) to the point where the velocity is measured in (so it ranges from 0 at the top to H at the plate). g is the gravitational acceleration ($= 9.8$ m/s^2) and v_{in} the velocity at which the fluid

enters the geometry via the inflow opening (set at 1.5 m/s). Using this, a velocity profile can be drawn for the points in the dotted box above the plate in figure 4.7. The velocities emerging from SAVOF are plotted as well (see figure 4.8). We can clearly see the influence of the no-slip condition at for instance the top of the plate (where h is at its maximum): the velocity drops radically to satisfy $u = 0$ at the obstacle.

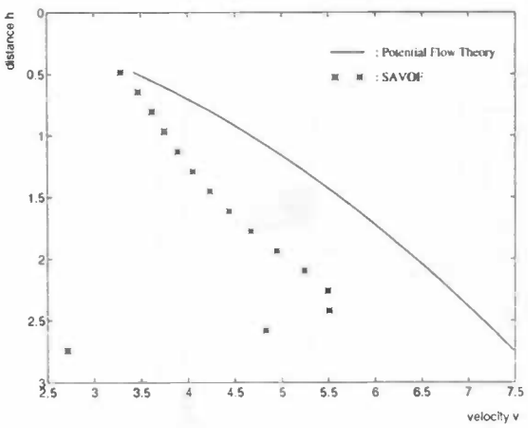


Figure 4.8: Velocity profile over a weir

4.2 Demonstrations

4.2.1 Dambreak problem

A demonstration that could be interpreted as a testcase of another nature is presented by the *dam-break* problem. In essence, we have a square filled with fluid, the gravitational acceleration downwards, and a vertical wall at the right side of the fluid (see figure 4.9).

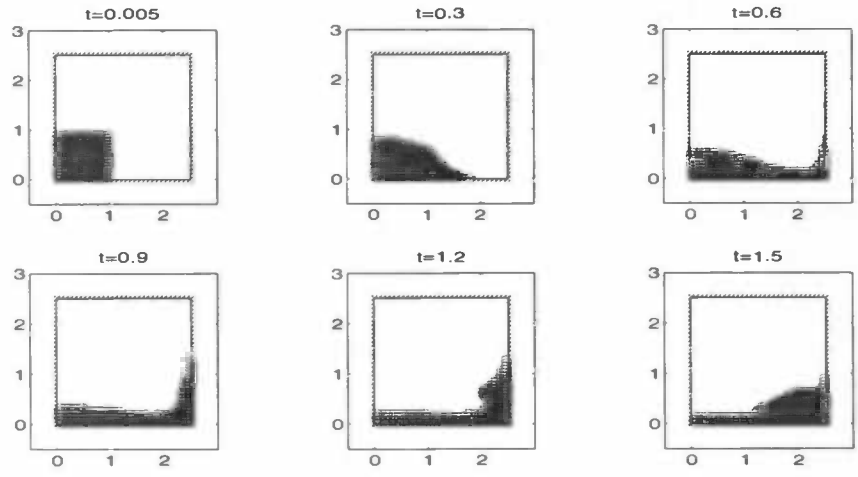
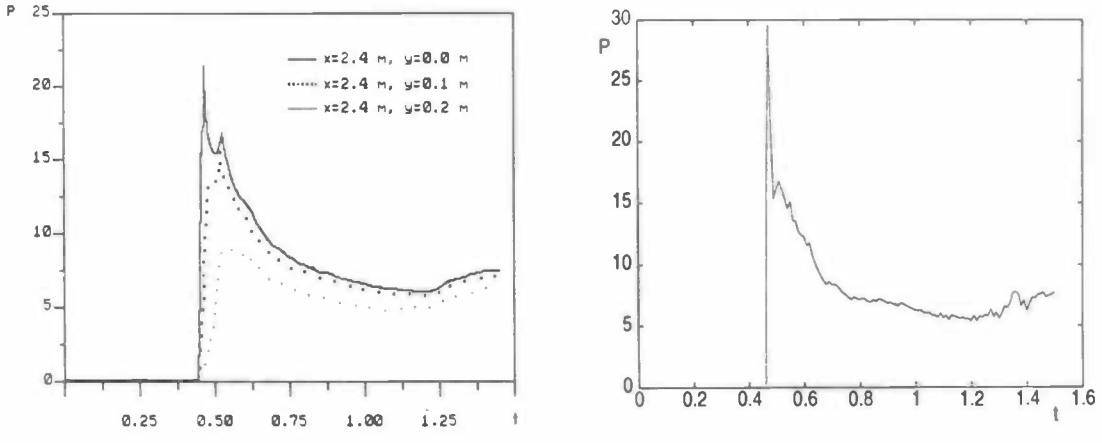


Figure 4.9: Simulation of a dam-break

At $t = 0$ the wall is removed, thus simulating a 'dam-break'. Since such a computation has been done before by Z.A. Sabeur in 1995 and by totally different means, it was an interesting testcase (see [4]). Sabeur measured the pressure in the lower right corner (see figure 4.10a). The same calculation was performed by SAVOF, resulting in a pressure depicted in figure 4.10b. The similarity in the two solutions is obvious.



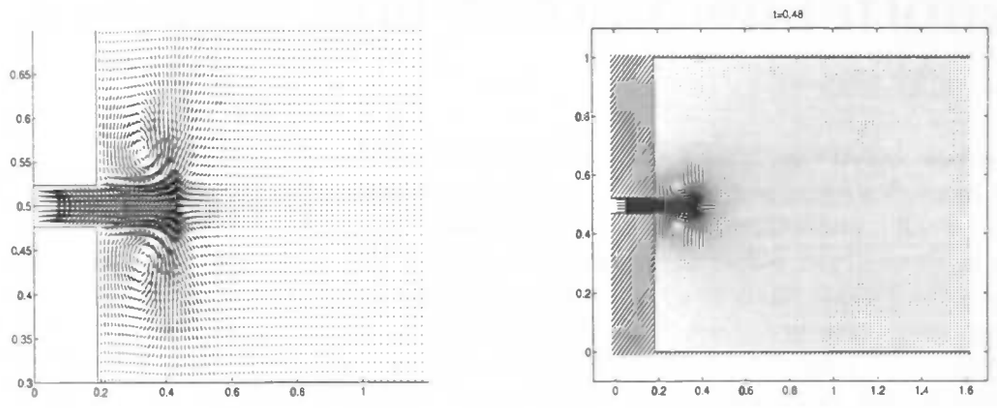
(a) Z.A. Sabeur

(b) SAVOF96

Figure 4.10: Comparison of pressures in the dam-break problem

4.2.2 Vortex formation

The flow of air through human vocal chords can be simulated by computer. The vortex formation involved is studied by fellow graduate student J.W. Phylipsen (see [3]). A special computer program was written to simulate a completely filled container to which fluid is added through a square-edged nozzle. With all the implemented additions it is simple to perform the same simulation by SAVOF. As can be seen in figure 4.11 the results look very similar, taking the different scales into account.



(a) J.W. Phylipsen

(b) SAVOF96

Figure 4.11: Vortex formation

4.2.3 Oscillating container

So far, all of the calculations were performed in a non-moving geometry. Next, consider a spherical, 2D container of 10x10 cm, with the letters SAVOF96 as obstacles in it. The container is partially filled and then moved up and down with a frequency of 10 Hz and an amplitude of 50 cm. Im impression of the resulting wild sloshing can be seen in figure 4.12.

Figure 4.12: Sloshing in 2D geometry

Because of the huge accelerations involved, a very small timestep is required, so the calculation time can reach the length of days, especially if the grid is suitably refined. E.g. 10^5 time steps on a grid with 10^4 grid points, will take about 14 CPU hours on a *Cray J90*; fortunately, the wall-clock time on this parallel computer will be significantly lower thanks to SAVOF's good vectorization and parallelization properties.

Chapter 5

Conclusions and recommendations

As can be concluded from this report, SAVOF has become a very easy-to-use and advanced program for simulating both two dimensional and axisymmetric free surface flows in complex geometries. Comprehensive testing increased confidence in its capabilities. SAVOF could in fact be used for commercial ends, all the more considering the rather poor supply of software packages in the area of free surface flow simulation. To make SAVOF fit for use in production or for educational ends, a few utilities could still be added. To name some, a mouse-driven application could be added to draw geometries, possibly completed by filters to import from several *CAD* programs. Also, with not too much effort, heat transfer could be integrated into SAVOF to even broaden its field of applicability. The creation of the grid could be made more sophisticated, allowing refinements at arbitrary places, that may eventually lead to savings in calculation time. The post processing in the form of movies should be available not only on Personal Computers, which is the case at the moment, but also on, for example, Unix workstations. In order to achieve this, the post processing TURBO PASCAL program should be cross-compiled to *C* or some other programming language supporting graphics.

A 3D version of SAVOF, is currently being developed at the R*u*G in the form of COMFLO (see [2]). This program is not going to replace SAVOF, but should be interpreted more as an extension towards fully three-dimensional flow. SAVOF remains a valuable tool in a large number of essentially 2D flow problems.

Appendix A

Program description

This appendix is devoted to the more practical description of SAVOF. The extended version of SAVOF (now called SAVOF96) consists of over 5000 lines of FORTRAN77 code. This code can be compiled to run on many computer systems varying from a personal computer to a powerfull supercomputer. The first section describes SAVOF96's calling sequence. Then, some important *common block* variables are discussed, and a brief explanation of the subroutines and input and output files is given.

A.1 Calling sequence

The calling sequence within SAVOF96 can be represented as follows:

```
initialisation   SETPAR   GRID
                 SETFLD   SURDEF
                          MKGEOM
                          LDSTAT
                          LABEL
                          BC       BCBND
time step        INIT     BCBND
                 PETCAL   BC       BCBND
                 TILDE    BDYFRC
                 SOLVEP   COEFF
                          CONVRT
                          PRESIT   SLAG
                                   SVSTAT
                          BC       BCBND
                 CFLCHK
                 DTADJ
                 VFCONV   LABEL
                 SVSTAT
```

In which the output routines are omitted for presentational reasons. There is a routine PRT that is invoked every `prtdt/delt` time steps, to be specified in the input file. This occurs after VFCONV is executed. Then PRT calls all the desired subroutines that produce specific output. These and the other subroutines are discussed in appendix A.3.

A.2 Common block variables

In FORTRAN, common blocks are used to make variables global. That is, the variables in a common block can be used in every subroutine in which this common block is declared, enabling subroutines to exchange data. Below, we state the variables of importance in alphabetical order.

/CASE/ contains external body forces and their controls:

`Omega, DelOme`	: rotation rate and initial rotation relative to `Omega`
`XO, YO`	: in the case of two dimensional rotation: point about which the rotation takes place
`TwUp, TwDown`	: time to start and end the rotation
`UO, VO`	: intitial velocities in x- and y-direction respectively
`Ampl, Freq, AAngle`	: amplitude, frequency and the angle under which an oscillation is performed
`GX, GY`	: accelerations in x-and y-direction respectively
`TxOn, TxOff, TyOn, TyOff`	: times to turn accelerations in x- and y-direction on and off

/COEFP/ contains the coefficients for the pressure in the Poisson equation:

`CC(I,J)`	: coefficient of $p_{i,j}$
`CN(I,J), CS(I,J), CE(I,J), CW(I,J)`	: coefficients of $p_{i,j+1}$,$p_{i,j-1}$,$p_{i+1,j}$ and $p_{i-1,j}$ respectively
`DIV(I,J)`	: right-hand side of the Poisson equation (3.3)

/GRIDAR/ contains parameters involving gridsize:

`X(I)`	: x-co-ordinates of the grid points
`XI(I)`	: x-co-ordinates of cell centers ($\frac{1}{2}(x_{i-1}+x_i)$)
`DelX(I)`	: distance in x-direction between two subsequent grid points ($\Delta x_i = x_i - x_{i-1}$)
`Y(J)`	: y-co-ordinates of the grid points
`YJ(J)`	: y-co-ordinates of cell centers ($\frac{1}{2}(y_{j-1}+y_j)$)
`DelY(J)`	: distance in y-direction between two subsequent grid points ($\Delta y_j = y_j - y_{j-1}$)
`RX(I), RXI(I), RDX(I)`	: inverse of `X(I)`, `XI(I)` and `DelX(I)` respectively i.e. $\frac{1}{x_i}$, $\frac{2}{x_{i-1}+x_i}$ and $\frac{1}{\Delta x_i}$
`Circum(I)`	: the circumference of a circle with radius x_i (in the axisymmetric case, 1 otherwise)
`RDY(J)`	: inverse of `DELY(J)` ($\frac{2}{y_{j-1}+y_j}$)
`CX,CY`	: stretch parameters in x- and y-direction respectively
`CYL,ICYL`	: floating-point and integer version of the 2D/axisymmetric switch (`CYL=0` for 2D and `CYL=1` for axisymmetric geometries)
`IMaxUs, JMaxUs, IM1Us, JM1Us, IM2Us, JM2Us`	: number of grid points in x- and y-direction (`IMaxUs=IM1Us+1=IM2Us+2` and `JMaxUs=JM1Us+1=JM2Us+2`)

/ORGA/ is used for cell labeling:

NF(I,J),	: contain cell labels for the current and previous time level
NFN(I,J)	respectively (see page 9)
PETA(I,J)	: coefficient used to interpolate the pressure (see page 11)

/PHYS/ contains pressures and velocities at the current and the previous time level. An N means 'at the previous time level':

F(I,J),FN(I,J)	: VOF (volume of fluid) / indicator function (see page 9)
U(I,J),UN(I,J)	: horizontal/radial velocity
V(I,J),VN(I,J)	: vertical/axial velocity
W(I,J),WN(I,J)	: azimuthal velocity
P(I,J),PN(I,J)	: pressure
PS(I,J)	: pressure at the free surface
VMAX	: maximum attained velocity
PMIN,PMAX	: minimum and maximum pressure

The latter three numbers are used for scaling during post-processing.

/TIMES/ contains parameters related to time levels and steps.

Cycle	: time step number
T	: current time
DelT	: time step
DelTMx	: maximum allowed timestep
TFin	: end time
TStart	: start time

A.3 Subroutines

Below, a short description of SAVOF96's subroutines is given.

- BC : sets the boundary conditions for the velocity components
- BCBND : handles influence on velocities from in- and outflow
- BDYFRC: computes the apparent body force
- CFLCHK: monitors the CFL number and sets a flag for time-step adjustment
- COEFF : defines coefficient matrix for Poisson equation including boundary conditions at the wall and free surface
- CONVRT: converts 2-dim datastructure into 1-dim datastructure for more efficient implementation of pressure solver
- DTADJ : halves or doubles the time step (old time step will be repeated)
- GRID : makes a (non-uniform) grid
- INIT : starts a new time step
- LABEL : labels empty, surface (preliminary) and full cells
- LABSUR: alternative labeling system for surface cells (for future use)
- PETCAL: labels surface cells and computes pressure at free surface
- PRESIT: solves Poisson equation and controls SOR relaxation factor
- PRT : prints and writes results
- SETFLD: initializes fluid configuration
- SETPAR: reads input file
- SLAG : performs one SOR sweep. It uses a 1-dimensional implementation
- SOLVEP: organizes pressure calculation and updates velocity
- SURDEF: generates liquid configuration
- TILDE : integrates momentum equations
- VFCONV: moves fluid, i.e. adjusts VOF-function, and re-labels

- AVS : generates output to be processed by AVS
- GNUPLT: writes data to a pipe to be displayed 'live' by GNUPLOT
- FILOUT: determines and saves the fill ratios of specified regions
- FLXOUT: determines and saves the fluxes through specified areas
- FRCOUT: integrates pressure on the walls in both x- and y-direction and writes the resulting forces to file
- LDSTAT: reads the restart file (produced by SVSTAT)
- MATLAB: produces MATLAB output (co-ordinates of 'particles' with their velocities and pressure
- MKGEOM: reads a geometry file and determines NF appropriately
- MKSTRL: keeps track of specified particles by integrating velocities
- MTLB : another routine for MATLAB output, especially for movies
- MPTOUT: interpolates velocities in points specified in the geometry file and saves them
- PRTFLD: produces a sequence of characters representing the entire geometry including obstacles (#=obstacle, *=fluid, I=inflow and O=outflow)
- PRTFLX: keeps track of the amount of in- and outflow
- SVSTAT: saves crucial variables to file for later use

The latter 14 subroutines were added to the 'original' version of SAVOF96.

A.4 Files

A.4.1 Input files

SAVOF96 uses at least one input file, the main input file. To create a special geometry, another input file can be used. The main input file is called SAVOF96.IN and has the following structure:

```
***** tank geometry ***************************************************
iCyl     Xmin      Xmax      Ymin      Ymax    SpGeom
   1      0.0       1.0       0.0      50.0         0

***** liquid configuration ********************************************
LiqCnf    xp    yp     r     xq     yq
     3   0.0   0.0   0.0    1.0   50.0

***** grid definition *************************************************
iMaxUs   jMaxUs     cx     cy
    10      100    0.0    0.0

***** liquid properties ***********************************************
Sigma    CAngle      Nu
  0.0      90.0     0.1

***** body forces and external motion: 2D *****************************
 Gx     TxOn     TxOff     u0        Gy     TyOn     TyOff     v0
0.0      0.0     100.0    0.0       0.0      0.0     100.0    0.0
Ampl Freq Angle
 0.0      0.0       0.0
Rpm     DelOme    TwUp     TwDown      x0      y0
0.0        0.0     0.0        0.0     0.0     0.0

***** body forces and external motion: axisymmetric *****************
  Gy     TyOn     TyOff       v0       Ampl     Freq
 0.0      0.0         0     -1.0        0.0      0.0
 Rpm    DelOme    TwUp    TwDown
 0.0       0.0     0.0       0.0

***** boundary conditions and inflow characteristics ***************
left     right    top    bottom    Amplin    Freqin
  2        2       8        7        1.0       0.0

***** upwind parameter and Poisson iteration parameters *************
Alpha       Epsi      ItMax    OmStrt
  1.0     1.0e-3        500       1.9

***** time step and restart control **********************************
TFin    DelT     PrtDt/DelT      svst      svdt
60.0     0.1              5         0     0.100

***** print/plot control **********************************************
gnu    matlab    avs   vofmat   velop   height   prtflx
  1         1      0        0       0        0        0

***** stream lines ****************************************************
nrx    nry    nrdt     xps     yps      xqs      yqs       t    dt/delt
```

```
  0     0      0   40.0   30.0    80.0    63.0    0.1          1

***** fluxes ****************************************************
number of fluxes to be printed
     1
    p1      p2      p3      hor
   0.0     1.0    25.0        1

***** fill ratios ************************************************
number of fill ratios to be printed
   2
  xpf     ypf     xqf     yqf
  0.0     0.0     1.0    25.0
  0.0    25.0     1.0    50.0

***** body forces ************************************************
integrate pressure in x direction or in y direction
                      0                 0
```

The input file will be explained following the above example.
In the *tank geometry* section the following parameters have to be set. `iCyl` is the switch between 2D and axisymmetric geometries (`0` for two dimensional, and `1` for axisymmetric calculations). `(Xmin,Ymin)` and `(Xmax,Ymax)` are the lower left and the upper right corner of the tank respectively (the same units should be used for all parameters, e.g. if the values here are in meters, then the velocities should be given in meters per second. The units don't necessarily have to be SI units, but have to be consistent with eachother !). `SpGeom` should be a number `0..9`. If greater than 0, the file `geo#.in`, with `#` the value of `SpGeom`, is used for geometry input (see section 3.2.2).

In the *liquid configuration* section the initial form and position of the fluid has to be specified. The meaning of the latter 5 arguments depends on the value of the first in the following manner:

LiqCnf	
0	: no fluid whatsoever
1	: lower part of the cylinder with the surface shaped as a semi-circle at average height `yp`
2	: drop with center `(xp,yp)` and radius `r`
3	: rectangle with nodes `(xp,yp)` and `(xq,yq)`
4	: drop (radius `r`), falling along vertical axis at $y =$ `yp` into a pool of depth `yq`
5	: fluid filament with width `r` and height `yp` including a semi-sphere

The third section serves for the *grid definition*. `iMaxUs` and `jMaxUs` are the number of grid points in x- and y-direction respectively. `cx` and `cy` are stretch parameters that can have three values: if `0`, no stretching is performed. Positive values result in smaller cells near the outer walls, and negative values in smaller cells near the x-axis and the y-center plane.

The *liquid properties* can be defined by `Sigma` (specifying the kinematic surface tension $\frac{\sigma}{\rho}$), `CAngle` (the contact angle) and `nu` (the kinematic viscosity $\frac{\mu}{\rho}$).

For an explanation of the next two sections (*external motions* for 2D and axisymmetric) we refer to section A.2 where in the common block CASE all the parameters occur.

Specified next are the *boundary conditions*. For every side, the desired treatment can be stated here. Values 1 or 2 set the boundary condition to *slip* or *no-slip* respectively, while values 7 or 8 make the entire side an opening for out- or inflow respectively. The rate of inflow is set at Amplin, and can eventually be made to oscillate with frequency Freqin.

The iteration process can be regulated by the parameters defined in the *numerical parameters* section. Alpha is the upwind parameter that should be 0 for central and 1 for upwind discretization. With Epsi, the Poisson convergence criterion can be controlled. ItMax is the maximum number of iterations that SAVOF96 is allowed to perform in one time step. Finally, the initial relaxation factor in the SOR iteration can be prescribed using OmStrt. This relaxation factor is automatically adjusted later on if necessary.

In the section *time step and restart control* the endtime TFin and the initial time step DelT can be specified. The latter may be reduced or doubled by SAVOF96 if necessary and possible. Also embedded in this section is the control of the frequency at which output is written to file. PrtDt is the time between two consecutive printouts to the screen and between two calls to the subroutines that produce all sorts of output (see section A.3). 20×PrtDt is the time between two consecutive large printouts to the screen. At the end of the line, parameters for the making of 'backups' are specified. SVST can have three values: 0 if no restart backups are required, 1 to save the program state every SVDT time units and 2 if a saved state from a previous run is to be read at startup, and after that the execution should proceed as with SVST=1.

The next five sections of the input file are devoted to SAVOF96's output.
First, there is the *print/plot control* section. It consists of 7 switches used to choose the kind of output SAVOF96 should produce. All the switches should be either 1, to enable, or 0, to disable the output option. The following abbreviations appear:

gnu : live GNUPLOT animation of liquid configuration
matlab : velocity and pressure data in MATLAB format
avs : velocity and pressure data in AVS format
vofmat : VOF-function (for use in MATLAB)
velop : additional velocity and pressure output in SAVOF96.OUT
height : height of free surface at several moments in time
prtflx : calculations of fluxes in in- and outflow opening

Next, there are the *streamlines* (see page 17). In the box defined by (xps,yps) and (xqs,yqs) as lower-left and upper-right corners, nrx*nry particles are placed: **nrx** in x-direction and **nry** in y-direction. These particles are followed, starting at t =t and after every dt/delt time steps, 'new' particles are released until the number of releases has reached **nrdt**. So, if any of the first three parameters nrx, **nry** or **nrdt** is zero, then no stream lines are created at all.

Below the streamlines section, there is the *fluxes* part of the input file. First, the number of fluxes to measure has to be specified. Then, for all of those fluxes a line is to be added to the input file specifying the location where to record the flux. Those locations are defined by horizontal or vertical lines. The line is defined by the first three co-ordinates, and whether the line should be horizontal or vertical depends on the fourth argument. If hor=1, then the line has start point (p1,p3) and end point (p2,p3) (resulting in a horizontal line); if hor=0, these

co-ordinates are (p3,p1) and (p3,p2) respectively (resulting in a vertical line). The resulting fluxes are written to FLUX##.OUT.

The *fill ratios* of a number of rectangular areas can be monitored and written to file by specifying the lower-left corner (xpf,ypf) and the upper-right corner (xqs,yqs) of the area. The number of areas is to be stated on the first input line of the section, and that amount of input lines is expected below. The files FILL##.OUT are created.

Finally, the switches for integrating the pressure in x- or y-direction can be set. The first number indicates whether (1) or not (0) the forces in x-direction have to be written to file, and the second does the same for the y-direction. If enabled, the files FORCE_X.OUT and FORCE_Y.OUT are made.

Rather important is the notion that all parameters *must* be specified in order for the input file to be accepted by SAVOF96. So, if no special geometry is used for example, the parameter SpGeom must be set to 0 and *not* be omitted, to avoid an error. Also, no extra lines should appear before the end of the last input section, because SAVOF96 expects all the lines to be at a constant place. If, for example, the *number of fill ratios to be printed* is set to 0, then all lines specifying possible previous 'fill ratio areas' must be removed. This concludes the description of the main input file.
For a description of the input file for the geometry, see section 3.2.2.

A.4.2 Output files

SAVOF96 can produce up to 22 different output files containing all sorts of information about the calculation. We shall discuss the most important ones briefly.

SAVOF96.OUT

This is SAVOF96's main output file. It contains the co-ordinates of the grid points, rough plots of the liquid inside the geometry at equidistant points in time (specified in the main input file) and comments from SAVOF96 about the convergence, evolution of the time step, etc. It also contains lines with the point in time, followed by the number of iterations that was required to obtain the desired accuracy, the relative change in volume and an array representing the level of the fluid.

GRID.OUT, COORD.CRD and IO.CRD

GRID.OUT is a text file containing a representation of the geometry and the fluid in it. The file is produced by the subroutine PRTFLD (see section A.3) and contains characters like # for obstacle cells, * for cells filled with fluid and I or O for in- or outflow cells respectively.
To be able to reconstruct the geometry with MATLAB or the TURBO PASCAL post processing program, the files COORD.CRD and IO.CRD are created. They contain the co-ordinates of the obstacles and in- and outflow cells respectively. Every line contains four numbers refering to the lower-left an upper-right co-ordinates of the obstacle or i/o cell respectively.

PIPE and VOF

These files are required by GNUPLOT to illustrate the calculation *on-line*.

M####.DAT and SAVOF.DAT

The subroutine MATLAB produces these files in order to be processed by MATLAB m-files or the TURBO PASCAL program. Every PrtDt/Delt time steps an M####.DAT is created containing as many lines of five numbers as there are cells filled with fluid. Every line exists of the x- and y-co-ordinate, horizontal and vertical velocity and pressure of such a cell. The last line of the file is used for the time, maximum velocity, minimum and maximum pressure and the number of the file.
Extra information about the calculation is given in SAVOF.DAT.

STREAM.OUT

Contains information about the streamlines requested in the main input file. There are two columns, to which every time step a number of lines is added. This results in a rather large file that can be seen as a series of 'time exposures' of the particles to be followed. Every time step must be interpreted as follows. On the first line, in the first column there is the point in time (in the second column appears a number of no importance -it represents the number of co-ordinates written at the current time point- but is added to allow the file to be read by MATLAB). The next line contains two numbers. The first is the number of particles currently traced (depending on the point in time, which determines the number of 'released' particles), the second is the number of requested streamlines in the input file (it is the product of **nrx**, **nry** and **nrdt** described above). The next lines contain the co-ordinates of the particles, beginning with the particles released first, and then -if they exist- the particles from the second release time, and so on. This file can be read by MATLAB producing figures like 3.2, and by a TURBO PASCAL program creating movies of particles leaving a trace behind them.

FLUX##.OUT and FILL##.OUT

Are, as described above, produced by subroutines FLXOUT and FILOUT respectively. The number in the filename corresponds with the line number in the input file defining the flux or fill ratio to be measured. Both types of file consist of two columns, the first one representing the time and the second the quantity under consideration.

FORCE_X.OUT and FORCE_Y.OUT

Contain three columns. The first is the time and the other two columns represent the forces exercised by the fluid on the solid walls and obstacles at the different points in time. In FORCE_X.OUT, the other columns are the forces directed to the right and to the left respectively. In FORCE_Y.OUT the columns represent the forces upwards and downwards respectively.

SPEED##.OUT

These files are created if, in the geometry file, one of the measure points is positive. On the first line the total number of measure points and the co-ordinates of the measure point of the particular file are given. Below, there are the time, the horizontal and vertical velocity in the measure point for every time step.

SAVOF96.SAV

Is SAVOF96's restart file (see **Saving** in section 3.2.4). This file can reach considerable proportions if the number of grid points is large.

List of symbols

Below, a list of the symbols used in this report is given with a short description. For some variables, a discrete version is given. The subscript ij refers to the number of the cell and the superscript n denotes the point in time.

x, y, x_i, y_j	= co-ordinates in horizontal and vertical direction respectively
Δx_i, Δy_j	= distance between two consecutive co-ordinates in x- and y-direction respectively ($\Delta x_i = x_i - x_{i-1}$ and $\Delta y_j = y_j - y_{j-1}$)
u, v, u_{ij}^n, v_{ij}^n	= velocity in horizontal and vertical direction respectively
p, p_{ij}^n	= pressure
ρ	= density
μ	= viscosity
ν	= kinematic viscosity ($\nu = \frac{\mu}{\rho}$)
ω	= rotation rate
$\boldsymbol{R}$	= abbreviation of $-(\boldsymbol{u} \cdot \text{grad})\boldsymbol{u} + \nu$ div grad $\boldsymbol{u} + \boldsymbol{F}$
$\boldsymbol{F}$	= vector containing external body forces

Bibliography

[1] E.F.F. Botta and M.H.M. Ellenbroek. A modified SOR method for the Poisson equation in unsteady free-surface flow calculations. *J. Comp. Phys.*, (60):119–134, 1985.

[2] J. Gerrits. Fluid flow in 3d complex geometries - a cartesian grid approach. Master's thesis, Rijksuniversiteit Groningen, 1996.

[3] J.W. Phylipsen. A simulation of vortex formation of a starting flow from a square-edged nozzle. Master's thesis, Rijksuniversiteit Groningen, 1996.

[4] Z.A. Sabeur. Development and use of an advanced numerical model using the volume of fluid method for the design of coastal structures. In K.W. Morton and M.J. Baines, editors, *Numerical Methods for Fluid Dynamics*, volume V, pages 565–573. Oxford Science Publications, 1995.

[5] A.E.P. Veldman. *Numerieke Stromingsleer*. Rijksuniversiteit Groningen, March 1994. Lecture notes.

[6] A.E.P. Veldman and M.E.S. Vogels. Axisymmetric liquid sloshing under low-g conditions - numerical simulation method. Reference manual TR 86057 L, NLR, 1986.

[7] R.W.C.P. Verstappen and A.E.P. Veldman. Data-parallel solution of the incompressible Navier-Stokes equations. In P. Wesseling, editor, *High Performance Computing in Fluid Dynamics*, pages 237–260. Kluwer Academic Publishers, 1996.

List of Figures

3.1 Screen dump of a .FLC movie 16
3.2 Streamlines around a sphere 18

4.1 Axial symmetric pipe used for in- and outflow test 20
4.2 Measured pressure and vertical velocities 21
4.3 Falling drop 22
4.4 Location of the top 22
4.5 Toricelli's theorem 23
4.6 SAVOF's velocity compared to the theoretical velocity 24
4.7 Flow over a sharp-crested weir 24
4.8 Velocity profile over a weir 25
4.9 Simulation of a dam-break 26
4.10 Comparison of pressures in the dam-break problem 26
4.11 Vortex formation 27
4.12 Sloshing in 2D geometry 27