Delft University of Technology
Faculty of Aerospace Engineering
Delft

Prins Maurits Laboratory
Organization for Applied
Scientific Research TNO
Rijswijk

Report LR - 513
Report PML 1987 - C17
SFCC PUBLICATION NO. 40

ON THE DIRECT SIMULATION OF VORTEX SHEDDING

C.W.M. van der Geld

Delft/Rijswijk, The Netherlands

February 1989

Delft University of Technology
Faculty of Aerospace Engineering
Delft

Prins Maurits Laboratory
Organization for Applied
Scientific Research TNO
Rijswijk

Report LR - 513
Report PML 1987 - C17
SFCC PUBLICATION NO. 40

ON THE DIRECT SIMULATION OF VORTEX SHEDDING

C.W.M. van der Geld

Delft/Rijswijk, The Netherlands

February 1989

ERRATA

page	location/lines	change
1	9th. f. below	exhausted
16	1th	not
15	3th f. below	propagates downstream
16	5th f. above	(J.B. Vos, private communications)
21	4th f. below	phenomenological
24	11th f. above	add : (x=0 at entrance of combustion chamber)
24	3th f. below	$p_2^- \exp(- i k_2 x)$ (add minus sign)
24	last line	add note : (*) If the injection chamber volume would have been 0,005 m^3 ,the additional length would have been 1,38 m rather than 1,22 m, showing that frequency and velocity of sound are of principal importance.
26	10th f. above	M_2 = 0,075
26	9th f. below	β_2 equals 0,985
27	7th f. below	B2 = EXP(-2*ALF*.45/795)
29	4th f. below	dependent
30	15th f. below	from the boundary
30	13th f. below	physics
30	7th f. below	omit "are"
32	10th f. below	induced
33	5th f. above	continuous
34	10th f. below	corresponding
35	formula	$\Gamma_1/\Gamma\ (x_1\ ,\ y_1)$
37	3th f. below	with the
39	first	(3.5)
39	7th f. below	in the flow
39 42	figures	immersed
42	middle	a discretized streamline
42	middle	method than (omit "is")
43	point 3	but has not always been
44	5th f. above	inhomogeneity
44	5th f. below	which
45	4th f. above	vortices

page	location/lines	change
45	2th f. below	complicated
45	3th f. below	zone flow structures are usually not
47	12th f. below	equal
47	13th f. below	velocity
48	6th f. below	Figure 3.14
50 55	figures	immersing
57	Figure 3.23	streamlines

TABLE OF CONTENTS

NOMENCLATURE

List of symbols

D	inner grain diameter (m)
d_{po}	initial port diameter (m)
L	grain length (m)
m_{air}	air mass flow rate (kg/s)
P	pressure (Pa)
P_c	mean combustion pressure in aft mixing chamber (Pa)
$\underline{v}$	velocity vector field
$\|\underline{v}\|$	magnitude of velocity (m/s)
x	coördinate in rectangular coordinate system
y	coördinate in rectangular coordinate system
η	function to determine Υ (Eq. 3.7b)
Υ	vortex blob shape function
φ	velocity potential (m^2/s)
ψ	stream function (m^2/s)
ω	vorticity (s^{-1})

List of superscripts

$\perp$	normal to a solid boundary

List of subscripts

po	initial value at port location, i.e. at the grain entrance
air	air or enriched or vitiated air
P	combustion chamber
wall	at or corresponding to a solid boundary
∞	at infinity

Acronyms

DEA	data exchange agreement
PE	polyethylene
PMMA	polymethylmethacrylate
PMLTNO	Prins Maurits laboratory of TNO
Re	Reynolds number
SFCC	solid fuel combustion chamber
Str	Strouhal number
TNO	Dutch orgaisation for pure and applied scientific research

1 INTRODUCTION

1.1 IMPORTANCE OF VORTEX FORMATION FOR SOLID FUEL COMBUSTORS

Solid fuel combustion chambers (SFCC's) are commonly applied in solid fuel ramjets and hybrid rocket motors, whereas other applications such as gas generation for the power industry are currently being investigated.

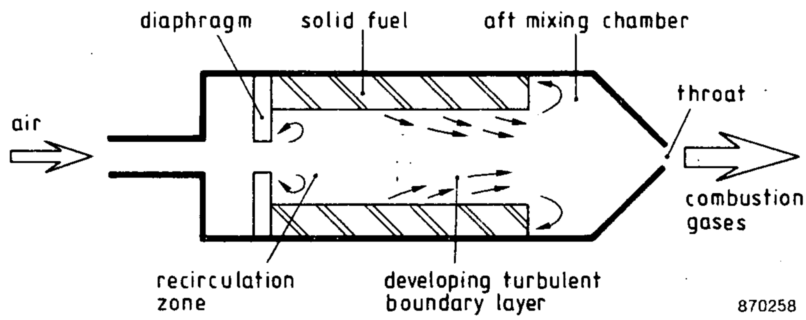


Figure 1.1
Schematics of a solid fuel combustion chamber

Figure 1.1 is a schematic of a SFCC. Air enters from the left and establishes a recirculation zone downstream of a sudden expansion. The hollow cylindrical fuel pyrolysis, product gases mix with the air and chemical reactions take place. Combustion products pass through an aft mixing chamber and are exhaused through a nozzle. During combustion, the inner grain surface of the fuel grain regresses until the grain is burned through or until the feeding of oxydant is stopped.
Flame stabilization is achieved by the rearward facing step, causing an area of elliptic flow at the entrance of the fuel grain. The importance of this recirculation zone is obvious since blow out would occur if the diaphragm would be omitted, but this importance is particularly well emphasized by the less familiar examples of the following two subsections.

1.1.1 Ignition problems

Ignition is usually established through additional supply of hydrogen and oxygen and a spark plug. If the additional supply is fed directly into the recirculation zone, ignition is reliable and almost instantly [1]. If, on the other hand, a spark plug is mounted in a mixing chamber upstream of the sudden expansion, ignition is troublesome. Much ignition gas is usually required. For inner fuel grain diamters of more than 60 mm, ignition is only achieved if the inlet temperature is raised above 600 K. In other circumstances ignition proved to be irreliable and was "hesitating".
A direct comparison between these ignition techniques was obtained in a joint research programme with DFVLR. Identical tests ware carried out with PE fuel grains in two different test rigs, and ignition was strikingly more difficult, if possible at all, in our test rig with the spark plug in the mixing chamber instead of the recirculation zone.

1.1.2 Extinction

In the course of a study of pyrolysis in a SFCC it became important to be able to create a sudden instantaneous stop of the combustion process. Because of remaining oxydant gases sustaining combustion it normally takes a few seconds after shutting the valves in the feed lines before the flame is extinguished.

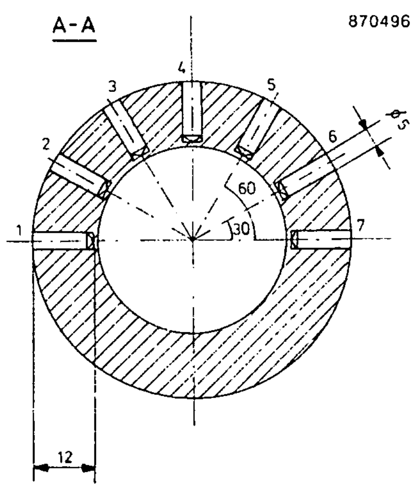


Figure 1.2
Cross-sectional view of fuel grain with holes

In order to shorten this after-burning holes were drilled in the fuel grain (see figure 1.2). As soon as the regressing surface of the grain reaches the bottom of a hole, the pressure in the chamber drops. Subsequently pyrolysis is extended to the surface of this hole. If pressure drops sufficiently, i.e. below ca. 0.3 MPa under standard test conditions (40 mm initial grain diamter, 150 g/s air mass flow rate), extinction is fully.

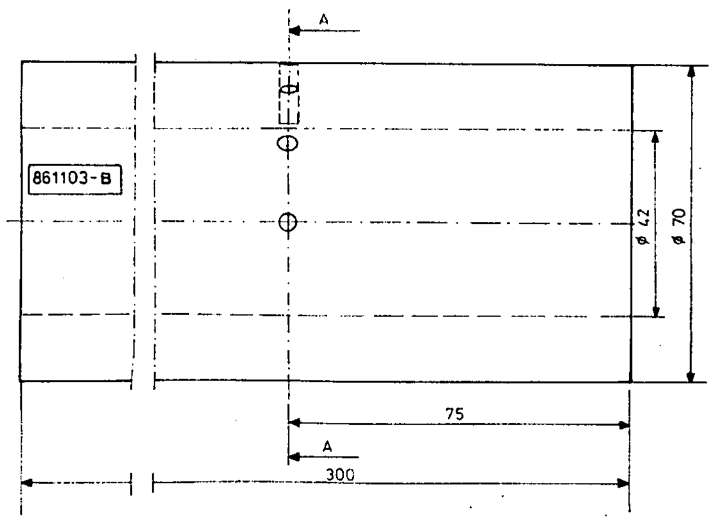


Figure 1.3
Side view of fuel grain with holes; #1

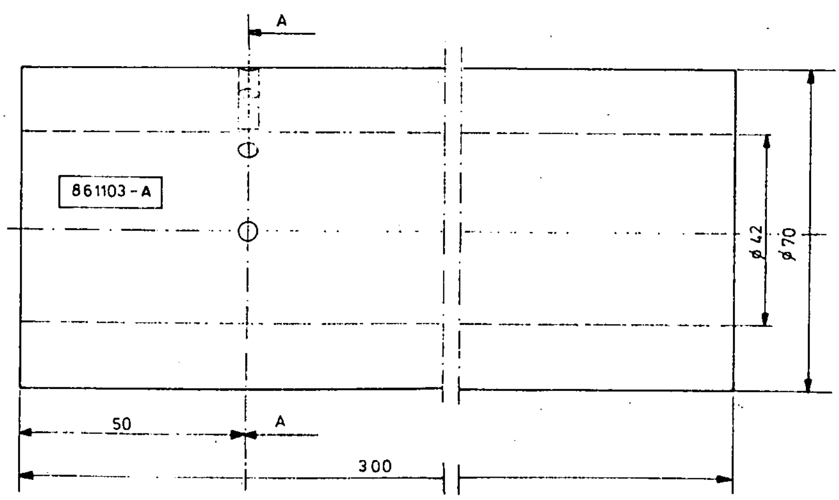


Figure 1.4
Side view of fuel grain with holes; #2

Two experiments were performed under standard test conditions:

- one with holes drilled far downstream of the recirculation zone (see figure 1.3);
- one with identical holes, but drilled close to the air inlet in the recirculation zone (see figure 1.4).

During the latter experiment, combustion was stopped immediately after one of the holes was burned through. As a consequence, the diameter of the holes remained almost as before the experiment; one hole even remained intact (see figure 1.5).

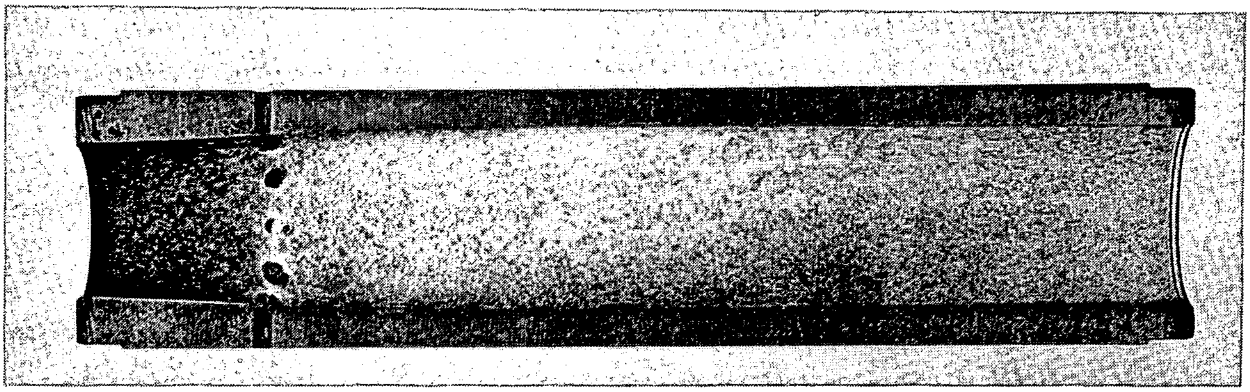


Figure 1.5a
Global view of grain #2 after burning

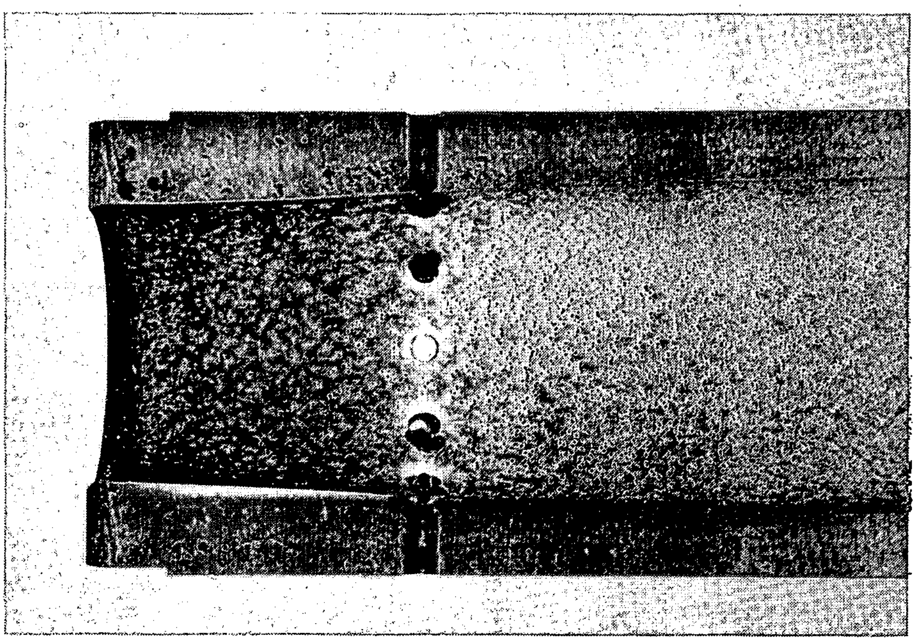


Figure 1.5b
Close-up of grain #2 after burning

During the first experiment, combustion was sustained for quite a long time. Because of this the holes were much larger than before the experiment (see figure 1.6).

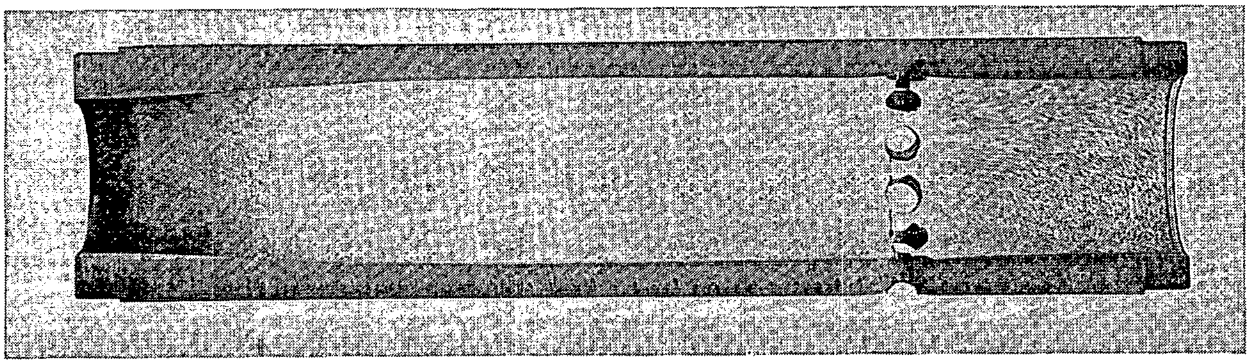


Figure 1.6a
Global view of grain #1 after sustained combustion

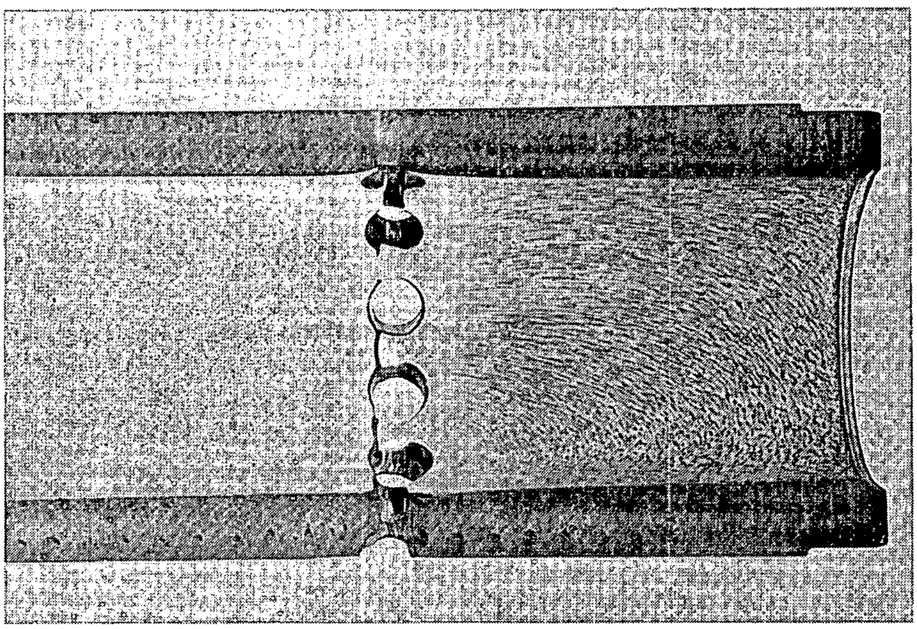


Figure 1.6b
Close-up of grain #1 after sustained combustion

This observation clearly indicates the importance of the recirculation zone, the region of elliptic flow downstream of the sudden expansion, for stabilizing the flame and sustaining combustion.

1.1.3 Influence of intrusive probes

The recirculation zone is a region where velocities are relatively low and where flame stabilization is achieved (see section 1.1.2).
All regions where the advection speed of fuel and the removal speed of products have the same order of magnitude as the reaction rate may serve as a flame stabilizing region.

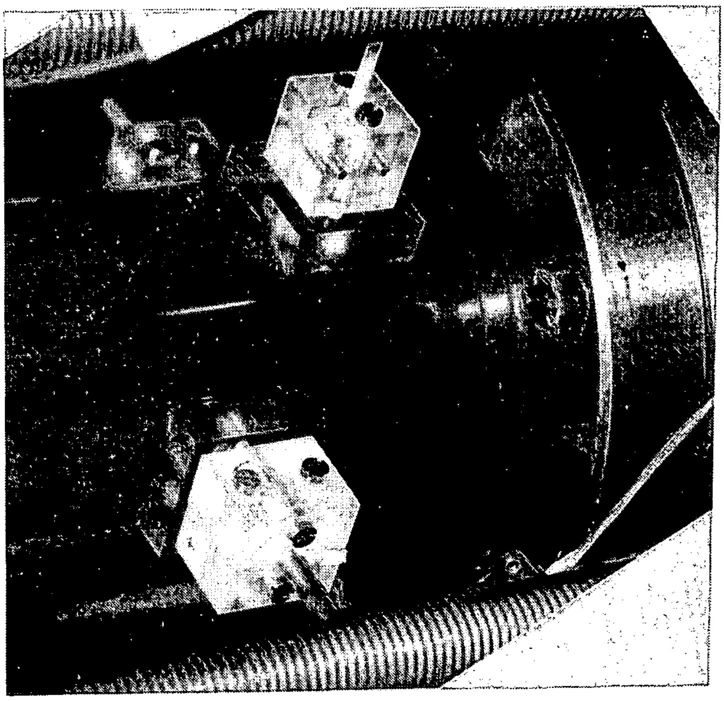


Figure 1.7
Thermocouple mounting on a fuel grain

This is also manifested by the results of experiments with intrusive ceramic tubes (see figures 1.7, 1.8 and 1.9). The horseshoe-vortices at the basis of these probes at the inner grain surface establish a region of increased regression rate. This was even apparent from the mean regression rate as measured from weight loss.

In the main part of the flow field, mixing and burning are incomplete as indicated by the presence of soot particles. The intrusive cylinders create a region of better mixing, more complete combustion and improved heat transfer to the wall. The latter controls regression rate, which is therefore enhanced.

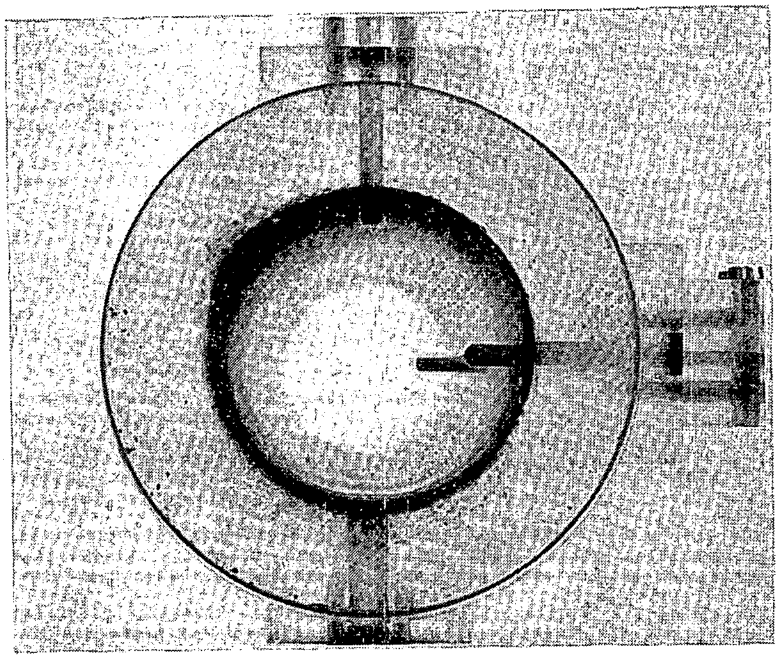


Figure 1.8

Thermocouple probes before burning

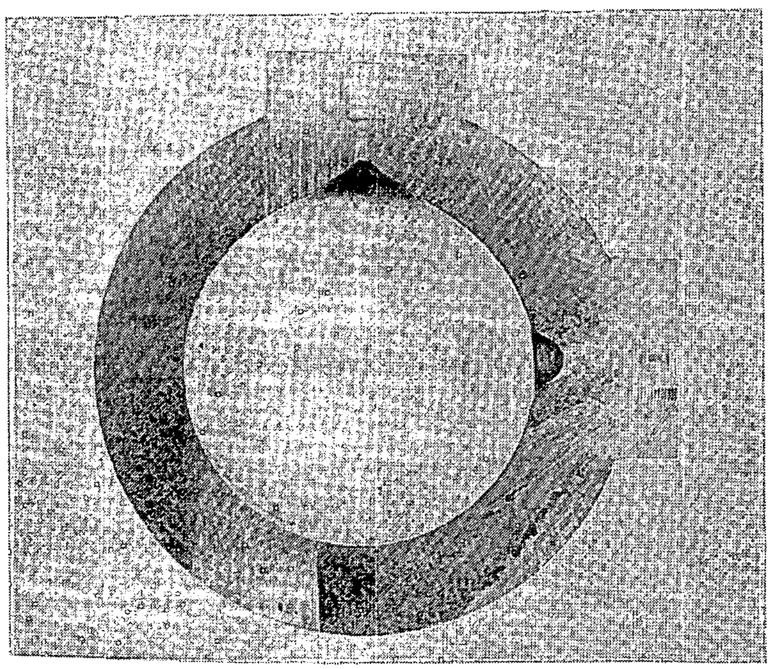


Figure 1.9

Thermocouple probes after burning

It is concluded that intrusive flow obstacles may serve as flame stabilizers and regression rate stimulators.

1.2 COOPERATION AND SUPPORT OF THE SFCC PROJECT

This research was started as part of a larger programme "Investigation of a Solid Fuel Combustion Chamber", which is financed by the Technology Foundation (Stichting voor de Technische Wetenschappen, STW) and the Management Office for Energy Research (Stichting Projectbeheerbureau Energie Onderzoek, PEO). In addition, money and manpower are made available by a special funding of Delft University of Technology (DUT) (Beleidsruimte), while also manpower, funding and computer facilities are provided by the Faculty of Aerospace Engineering (FAEDUT), and the Prins Maurits Laboratory TNO (PMLTNO).

In a Data Exchange Agreement, DEA, between PMLTNO and the Naval Weapon Centre, NWC, of the United States of America, vortex shedding research as described in this report is a major topic. NWC has already gained much experience with various inlet geometries under various conditions [2], and a joint experimental programme has been agreed upon. Theoretical modelling by means of the vortex method as described in this report is part of the DEA.

Some experimental results of the DEA with the Deutsche Forschungs- und Versuchsanstalt für Luft- und Raumfahrt e.V. (DFVLR), are also used in this report.

1.3 AIMS AND SCOPE OF THE PRESENT INVESTIGATION

The causes and effects are investigated of the oscillatory shedding of the large toroidal vortex structure that appears directly downstream of the rearward facing step in a circular channel.

Some of the effects on blow out and efficiency of a solid fuel combustion chamber are highlighted (chapters 1 and 2). The dependences of the shedding frequency on Reynolds number and inlet geometry are discussed (chapter 2).

The predictability of this vortex shedding is studied with the aid of a computational model based on the so-called Lagrangian vortex method. Some of the modelling assumptions are experimentally verified. Numerical computations are presented.

Not all the results of this report are fully conclusive. More work remains to be done, especially with respect to 3D-effects and diffusion aspects. Acoustical field triggering was separated from the flow field as a first step towards a proper simulation of the complicated, essentially time-dependent flow phenomena in a solid fuel ramjet.

2 EXPERIMENTS AND DEDUCTIONS

2.1 TEST FACILITY

The connected pipe test facility that was used in the experiments, is exhibited in figures 2.1 through to 2.3

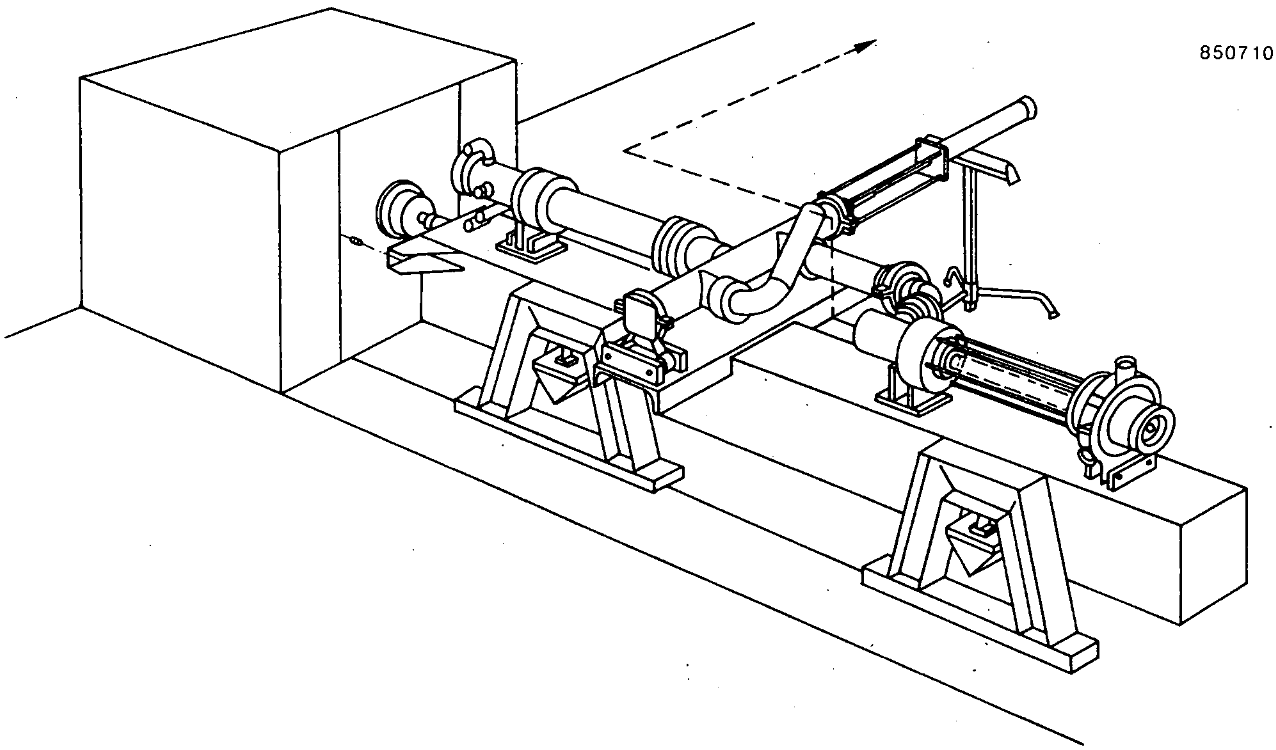


Figure 2.1
Schematic of connected pipe test facility (3D-view)

Air at precisely controlled temperature is produced in a vitiator in which methane is burned with oxygen enriched air. The oxygen content of the exhaust gases is computer controlled, just as the especially developed Sonic Control and Measuring Choke (SCMC) system. This SCMC system guarantees a constant oxidizer mass flow rate (± 3%) and allows for accurate mass flow rate measurements.

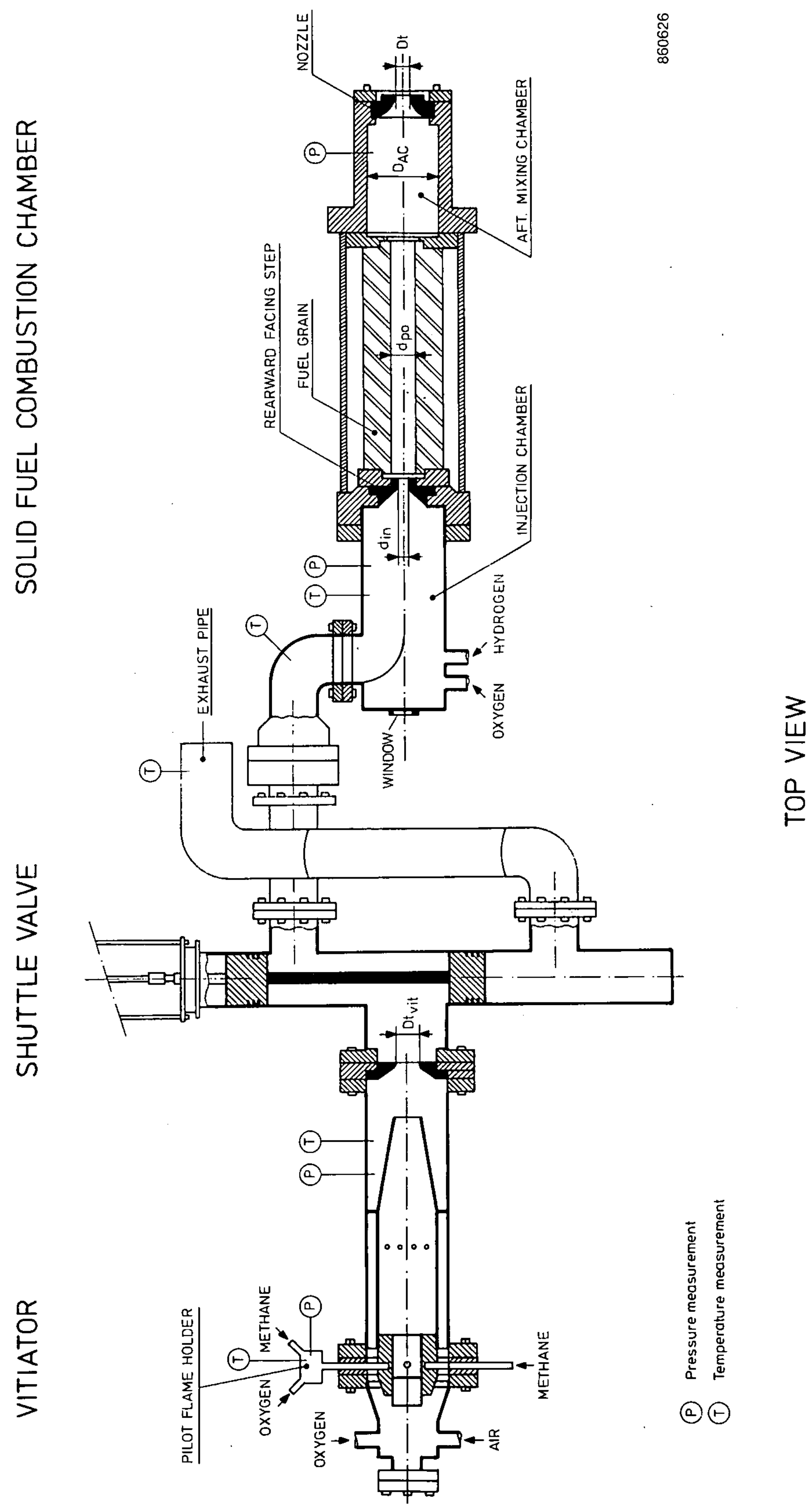


Figure 2.2

Schematic of connected pipe test facility (cross sectional view)

The shuttle valve (see figure 2.2) towards the solid fuel combustion chamber (SFCC) opens as soon as the vitiator exhaust gases are properly conditioned.

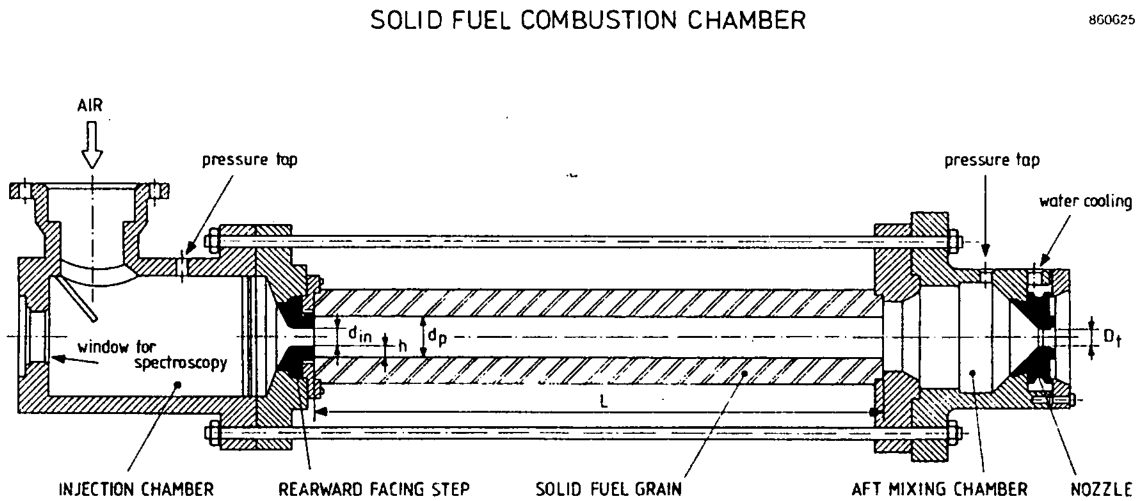


Figure 2.3

Schematic of combustion chamber (cross sectional view)

A schematic of the SFCC is shown in figure 2.3. Nominal sizes of the hollow cylindrical fuel grain were:

- length 300 mm
- inner diameter before burning 40 mm
- outer diameter 70 mm.

If the grain consists of transparant polymethylmethacrylate (PMMA), chordal beam maximum temperatures can be deduced from spectra recorded by a spectrographic system. One pyrometer registers intensity and a so-called "two-colour" pyrometer registers temperature of locally emitted radiation. Details of the optical system have been given by Wijchers [3].

Pictural observations were made on video and on high speed cinematographic film (10.000 frames/s).

After burning the grain inner surface was measured and analyzed (see figure 2.4).

Flame stabilization was achieved with a rearward facing step, usually consisting of a diaphragm (see figure 2.3). The sharp edges of this step represent a region of high shear in the flow, from which small-scale vortices emerge.

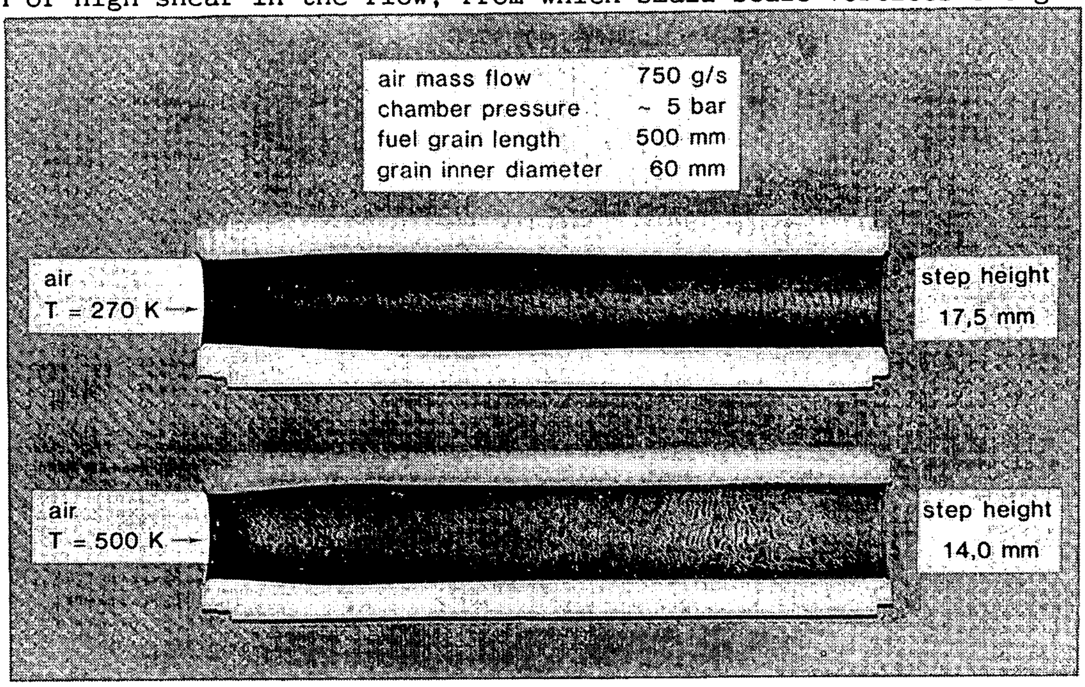


Figure 2.4

Polyethylene grain profiles after burning

Directly downstream of the entrance a recirculation zone with elliptic flow exists. Time-dependent flow phenomena in this zone are subject of this study.

2.2 EXPERIMENTAL RESULTS

2.2.1 Direct observation of the shedding of large vortex structures

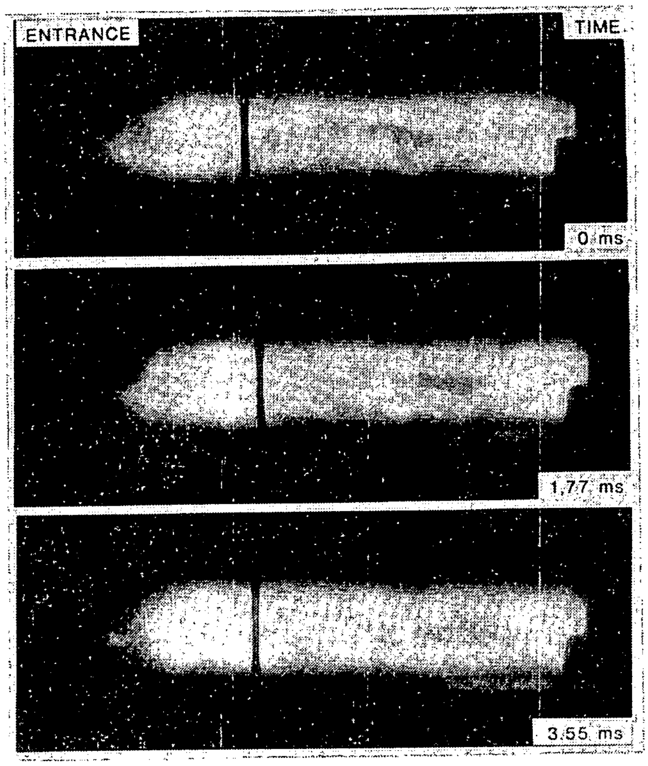


Figure 2.5a

High speed cinefilm recordings of a vortex structure being shed.

High speed cinefilms revealed the existence and oscillatory shedding of a large-scale toroidal vortex structure in the upstream end of the fuel grain, see figure 2.5. A careful analysis of several films yielded shedding rates between 80 and 100 Hz.

Shadowy parts on the pictures of figure 2.5 represent cooler spots in the flow. Initially they mark a part of the recirculation zone downstream of the rearward facing step. During the first phase of the shedding process this part seems to grow into the combustion chamber. Then suddenly it jumps off towards the centre of the grain, propagates at a speed low compared to the main stream, that has a velocity in the order of 200 m/s typically, and then diffuses in this main flow. This process repeats itself at a regular rate.

It is nog clear from these cinefilms whether the entire recirculation zone is shed or not. In the recirculation zone downstream of a rearward facing step always two toroidal vortices occur; one of them with a very small core, about 7% of the step height, right down in the corner of the sudden expansion. This small secondary eddy is known to increase in size with increasing blowing velocity. Possibly this smaller vortex is growing while the larger vortex structure slowly replaces itself away from the entrance.

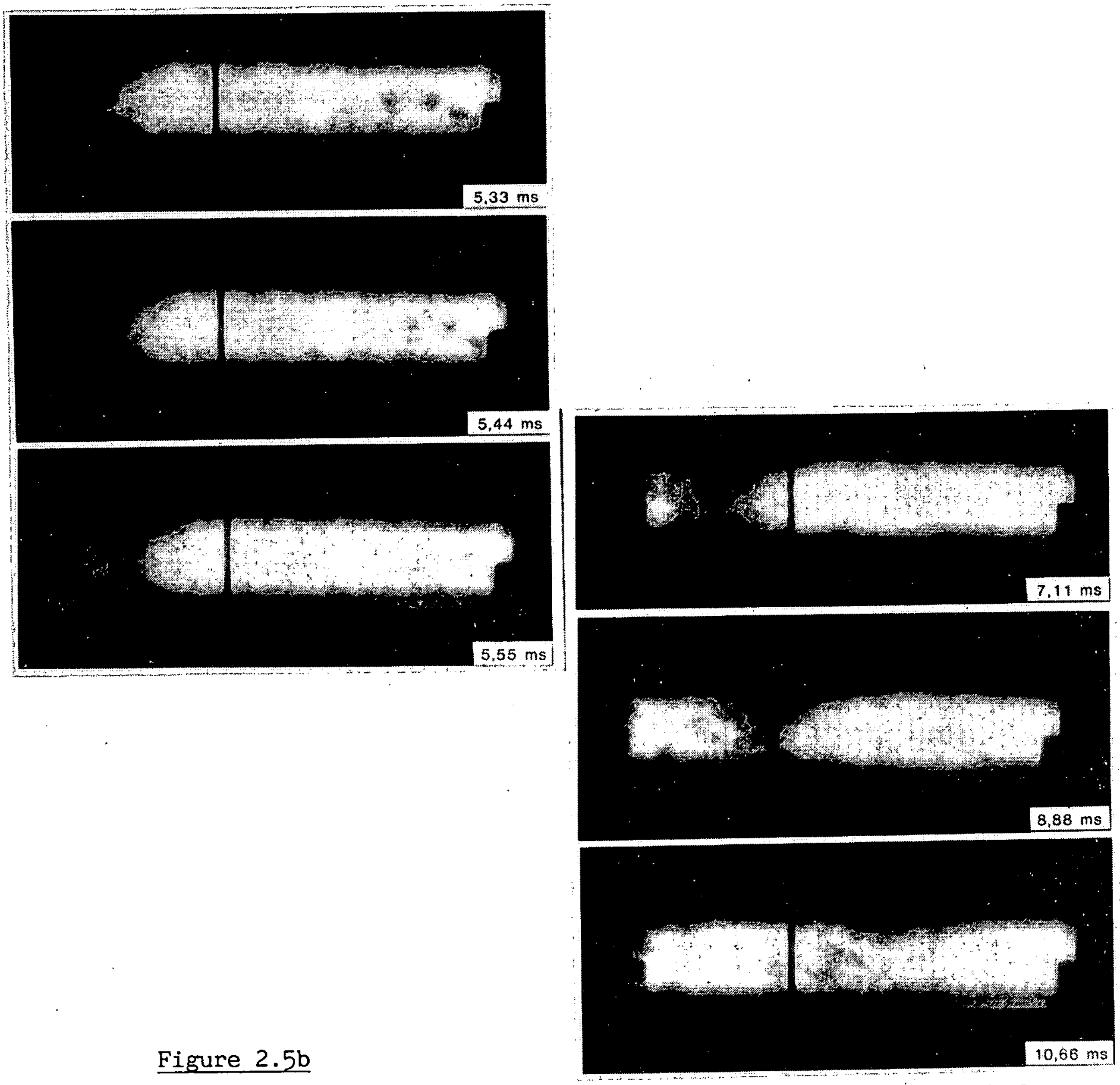


Figure 2.5b

High speed cinefilm recordings of a vortex structure being shed

Clearly the sweeping of the vortex structure through the fuel grain causes refreshment of the boundary layer and hence an increase in heat transfer.

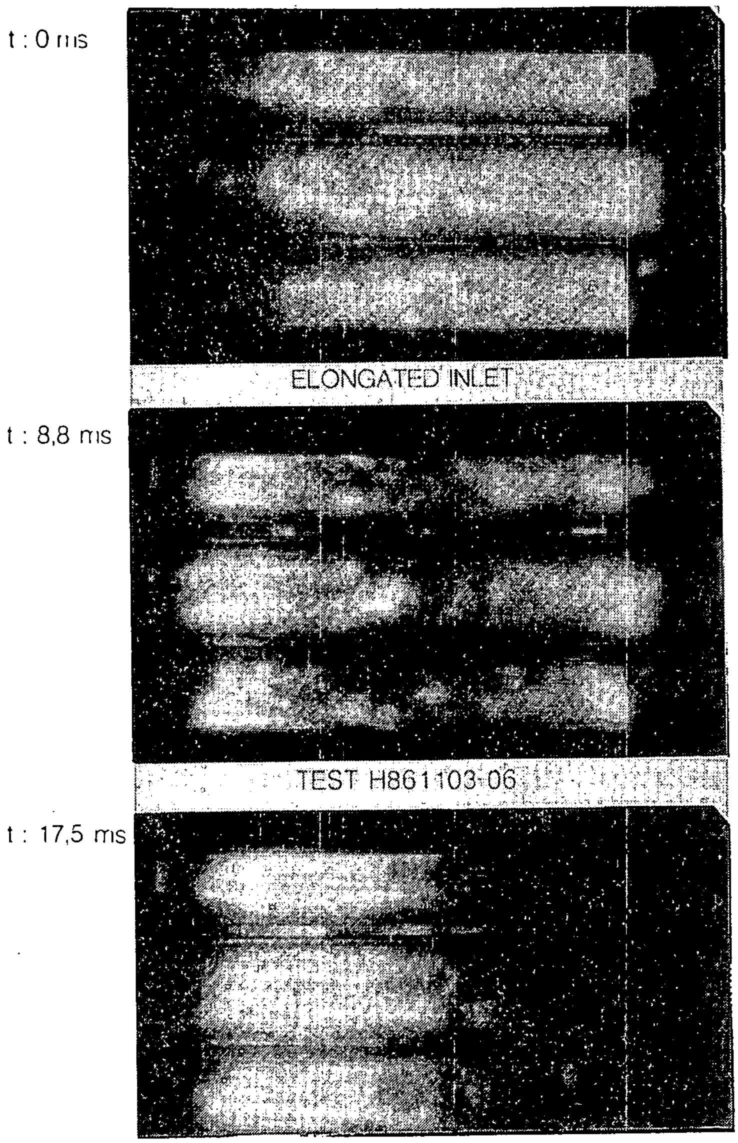


Figure 2.6

High speed cinefilm recordings with mirrors alongside the grain

Elongated mirror plates were mounted on both sides of the fuel grain. Cinefilms of this set-up revealed that the toroidal vortex structure remains essentially axis-symmetrical while propagating through the grain.

Oscillatory behaviour at the same frequencies was detected from pressure recordings (see figure 2.7), light emission measurements (see figure 2.8) and temperature measurements with the two-colour pyrometer. The amplitude of temperature fluctuations at 0,25 L from the entrance, L being the grain length, increased from ca. 300 K to ca. 600 K during a test of 24 s. At 0,75 L these fluctuations were less severe, about 200 K, probably as a result of the diffusion of the vortices in the main stream.

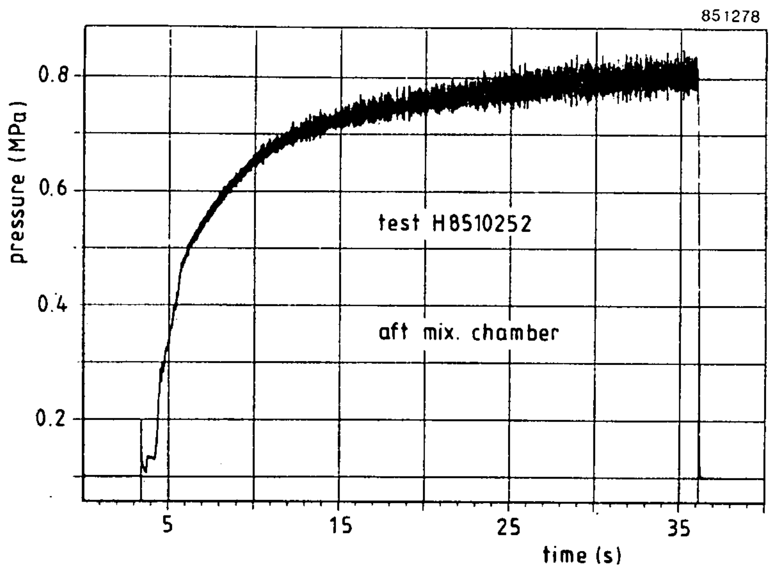


Figure 2.7
Specimen of pressure history

Small tuften wires were used to visualize the flow field in cold flow through a hollow cylindrical fuel grain, when no combustion takes place. High speed cinefilms again revealed an oscillatory flow behaviour at about 100 Hz, although some pictures were hard to interpret. The results are not really conclusive, but seem to indicate that combustion is not the primary cause of the oscillatory flow behaviour.

The inlet velocity and inlet temperature were varied to investigate the effect of the inlet Reynolds number on the shedding rate as measured from high speed film recordings. All results were in the range 80-100 Hz, and no systematic trend could be discerned.
The diameter of the inlet and the initial diameter of the fuel grain were also varied to investigate the effect of the inlet. These experiments will be discussed more fully in the next section. All frequencies measured for a certain inlet length were in the range 35-40 Hz. The inlet length was varied

to investigate the influence of acoustics and to improve the quality of the inlet turbulence.

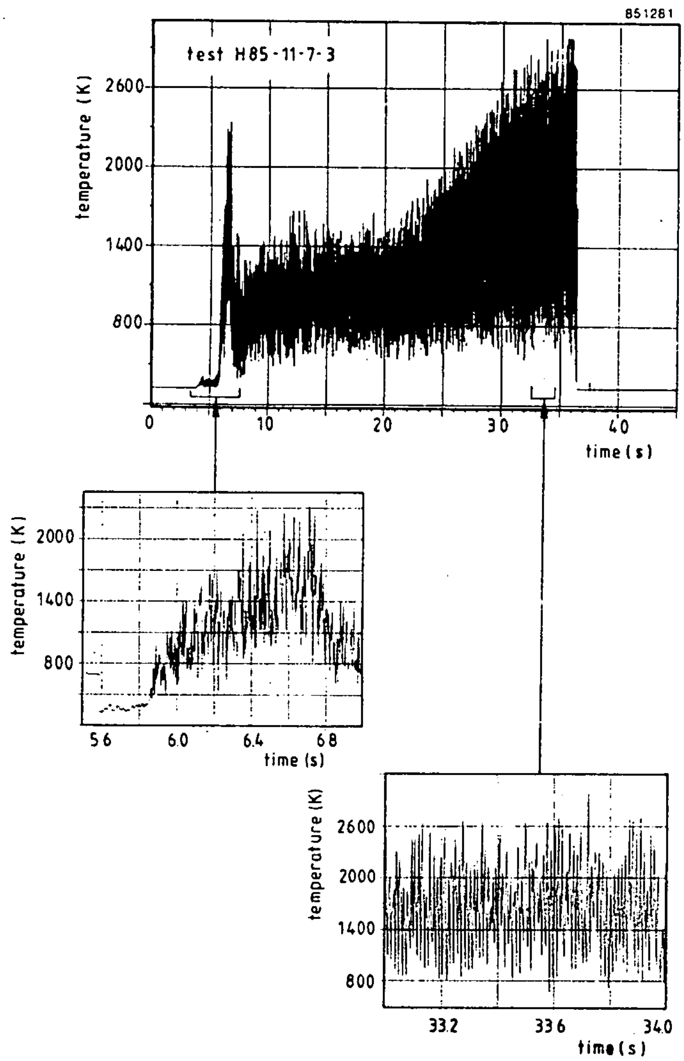


Figure 2.8

Radiation history; chamber pressure 0,9 MPa

From all these measurements it is concluded that the inlet Reynolds number has virtually no effect on the shedding frequency.

2.2.2 Elongated inlet section

The shear layer that extends from the inlet diaphragm into the combustion chamber responses to acoustic perturbations or perturbations that may origi-

nate from other sources. If incident acoustic waves are in the correct frequency range, a pressure fluctuation level may be provided that amplifies initial disturbances until breakdown of the shear layer. Large coherent structures of the recirculation zone may then detach and propagate into the combustion chamber.
The Strouhal number

$$S_a = 2\pi \, \nu_a \, \delta \, / \, U_o$$

determines the response of the shear layer to acoustical waves with frequency ν_a. Here δ denotes the momentum thickness of the shear layer and U_o, the maximum time-averaged flow velocity at the inlet.
The importance of S_a was investigated by adapting the length, L_o, of the inlet section. Note that for the first longitudinal acoustic mode ν_a is roughly equal to $\frac{1}{2} a_o/(L_c + L_o)$, where L_c represents the length of the combustion chamber and a_o the speed of sound. Note that cooler parts of the motor and the cool straightening section tend to decrease the average sonic speed. The actual average wave speed is a complicated function of the environment. Since oxidizer flow conditions were kept constant, U_o and δ remained unchanged.
To vary L_o, the test stand was adapted by inserting an inlet tube (see figure 2.9). Its inner diameter was 12, 15 or 18 mm. The length was 750 mm to guarantee well-developed turbulent flow at the inlet of the combustion chamber; in all cases the L/D ratio was larger than 40. The value of $(L_c + L_o)$ was approximately doubled by this adaptation.

With 18 mm inlet diameter, ignition was impossible even after increasing the supply time of H_2 and O_2 gases from 2 to 6 seconds. This is probably due to the fact that the spark ignitor was mounted in the mixing chamber at the upstream end of the inlet section (see section 1.1.1).

The frequency of the vortex shedding was again determined by counting frames of high speed cinefilms (recording speed 10.000 frames/second), eliminating excessively deviating numbers, and averaging. The following tests were performed with a 12 mm inlet section:

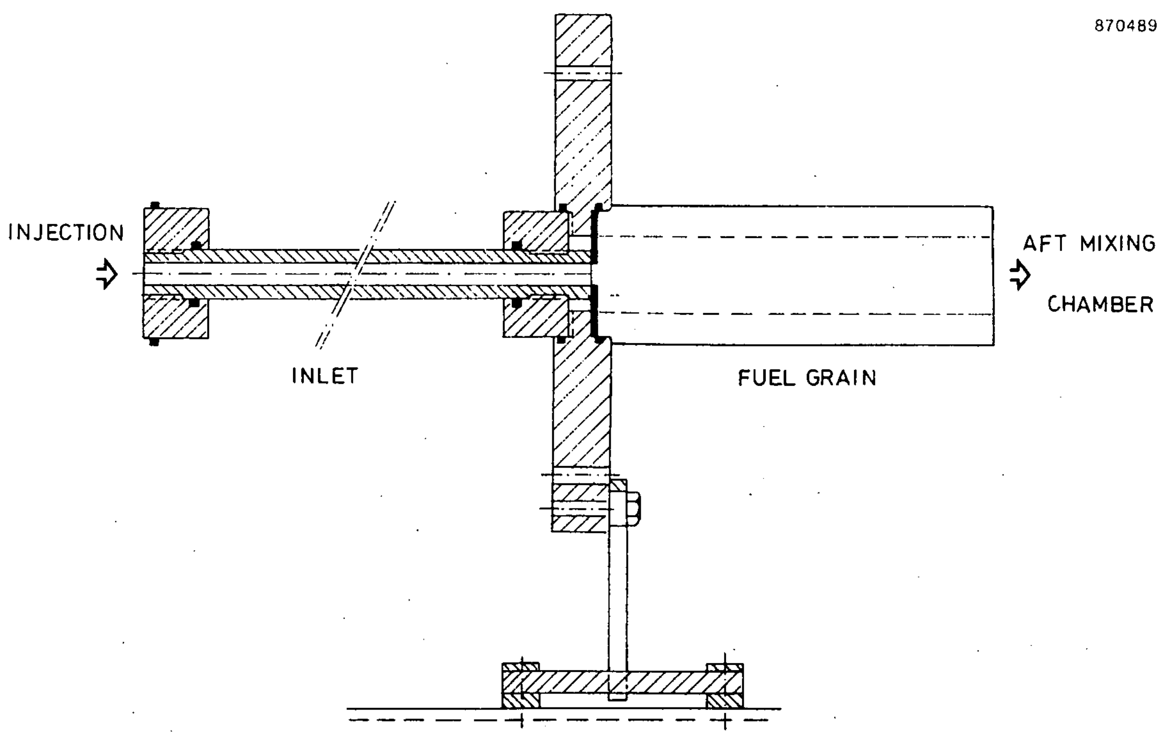


Figure 2.9
Schematics of chamber with elongated inlet section

H861103-3; H861103-4; H870121-03; H870121-04;
and the following tests with a 15 mm elongated inlet:
H861103-1; H861103-2; H861103-5; H861103-6; H870121-1; H870121-2.
All these tests yielded shedding frequencies in the range 35-40 Hz. The regularity of the oscillatory vortex shedding was increased with respect to the configuration with an inlet diaphragm. Clearly the quality of the turbulence at the inlet affects the sensitivity and susceptibility for pressure disturbances.

During the first phase of the shedding of a large vortex structure it again seemed to grow and gradually replace itself from the inlet. Then suddenly the structure jumped off and diffused in the main flow. No phenomelogical difference was therefore observed with the vortex shedding with a diaphragm instead of the long inlet section. Only the shedding frequency, ν_s, has altered from 67-100 Hz to 35-40 Hz, i.e. by a factor of roughly two. Since the frequency of the first acoustic mode, ν_a, was changed by the same factor it is concluded that ν_s is proportional to ν_a.

Acoustic waves may originate from the interaction of the vortical field with downstream impingement surfaces such as the nozzle [4, 5].
The above observations make clear that feedback may indeed exist from such pressure waves reflected by or generated at the downstream end of the combustion chamber.

A strong coupling between flow field and acoustical field would also explain the result of section 2.2, that the inlet Reynolds number hardly affects shedding frequency.
In the next section, the frequencies ν_a of the acoustical field will therefore be computed and compared with the observed shedding frequencies.

2.3 ACOUSTICAL FIELD FREQUENCY CALCULATIONS

In this section the main frequencies of standing acoustical waves in the ramjet will be calculated. The analysis is essentially an extension of the work of Clark and Humphrey [6].

To examine the unsteady behaviour of a two-dimensional ramjet combustor within the low frequency range, Yang and Culick [7] carried out an analysis in which both longitudinal and transverse mode oscillations were considered. Clark and Humphrey [6] simplified their treatment into a one-dimensional acoustic model that allows for the computation of perturbation pressure amplitudes and phase distributions in an idealized ramjet. They found very good agreement between predicted and experimentally determined frequencies and phase distributions. However, the prediction of the dependance of pressure amplitude on axial location turned out to be rather poor. In the following analysis, the value of this damping is shown to be inessential for the assessment of the dominant frequencies of lower order. This allows for a simplification of the calculation procedure by eliminating damping and pressure mode shape.

The model consists of an inlet section, indexed by 1, with known properties of the uniform flow of cold air. The injection chamber is treated as a resonance cavity, effectively enlarging the inlet section. A sudden expansion connects the inlet to the combustion chamber, indexed by 2, where flow properties are different (high temperatures). The entrance to the injection chamber is characterized by the complex reflection coefficient β_1, and the exit from the combustor by the reflection coefficient β_2.
Let g denote a reflected pressure wave, and f an incident one. Following Lighthill [8], the reflection coefficient is defined by:

$$\beta = g/f = (1 - Y_d/Y_u)/(1 + Y_d/Y_u)$$

in which Y_u is the upstream and Y_d the downstream admittancy of the channel joint:

$$Y = A/\rho\, c$$

Here A denotes the cross sectional area of the channel and c the velocity of sound.

A resonance cavity with volume V at the end of a channel effectively enlarges the length of the channel by the amount [8]

$$l = (c/\omega) \arctan (V \omega / c A)$$

The volume of the injection chamber used is approximately equal to $7{,}54.10^{-4}$ m^3. For a frequency of 55 Hz, the effective additional length of an inlet tube with a diameter of 15 mm amounts to 1,22 m if c equals 310 m/s. Note that $\omega = 2\pi\nu$.

Now suppose that the pressure field in the combustion chamber consists of one right (+) and one left (-) running acoustic wave:

$$p_2 = p_2^+ e^{i k_2 x} + P_2^- e^{-i k_2 x}$$

For the sake of brevity, the assumed sinusoidal time dependance has been omitted. The modified complex wave number k_2 is given by

$$k_2 = (\omega + i \alpha)/c_2(1 - M_2^2)$$

in which M_2 is the mean Mach number.

The damping is represented by α.

It is noted that p_2 can be multiplied with any function of x without affecting the analysis; Yang and Culick [7] multiplied p_2 by the term $e^{-i M_2 k_2 x}$.

The linearizing of the Navier Stokes equation immediately yields for the induced perturbation velocity

$$U_2 = \frac{1}{\rho_2 c_2} \left(p_2^+ e^{i k_2 x} - p_2^- e^{i k_2 x}\right)$$

Isentropic flow is assumed at the sudden expansion, and mass continuity yields $\rho_1 U_1 A_1 = \rho_2 U_2 A_2$, whence

$$\frac{c_1 A_2}{c_2 A_1} = \frac{p_1^+ - p_1^-}{p_2^+ - p_2^-} = \frac{p_1^+ - p_1^-}{p_1^+ + p_1^-} \cdot \frac{p_2^+ + p_2^-}{p_2^+ - p_2^-}$$

The last equality follows from $p_1(x = 0) = p_2(x = 0)$.
The definition of β yields on the exit

$$\beta_2 = \frac{g}{f} \left. \frac{p^- e^{-i k_2 x}}{p^+ e^{i k_2 x}} \right|_{x = L_2} = \frac{p^- e^{-i k_2 L_2}}{p^+ e^{i k_2 L_2}}$$

for the inlet entrance

$$\beta_1 = \frac{g}{f} = \left. \frac{p^+ e^{i k_1 x}}{p^- e^{-i k_1 x}} \right|_{x = -L_1} = \frac{p^- e^{-i k_1 L_1}}{p^+ e^{i k_1 L_1}}$$

Defining F_1 by exp $(2i\ k_1L_1)$ and F_2 by exp $(2i\ k_2L_2)$ we obtain

$$\beta_1 F_1 = p_1^+/p_1^- \quad \text{and} \quad \beta_2 F_2 = p_2^-/p_2^+$$

These expressions are substituted in the equation for c_1A_2/c_2A_1 to obtain

$$\frac{c_1A_2}{c_2A_1} = \frac{\beta_1F_1 - 1}{\beta_1F_1 + 1} \cdot \frac{1 + \beta_2F_2}{1 - \beta_2F_2} \tag{2.1}$$

This equation has also been derived, in a somewhat different manner, by Clark and Humphrey [6]. If the reflectances β_1 and β_2 are known, the only unknown in this equation is the complex frequency $\omega + i\alpha$. The real part of the equation can be further reduced to:

$$\left(1 + \frac{c_1A_2}{c_2A_1}\right) \left\{1 - \bar{\beta}_1\bar{\beta}_2 \cos\left(2\ k_1L_1 + 2k_2L_2\right)\right\} +$$

$$+ \left(\frac{c_1A_2}{c_2A_1} - 1\right) \left\{\bar{\beta}_1 \cos\left(2k_1L_1\right) - \bar{\beta}_2 \cos\left(2k_2L_2\right)\right\} = 0 \tag{2.2}$$

in which $\bar{\beta}_j$ is defined by β_j . exp $(-2.\alpha.L_j/c_j(1 - M_j^2))$.
To determine the reflectances, use can be made of the expression

$\beta = (1 - Y_d/Y_u)/(1 + Y_d/Y_u)$. The exit of the combustor is a nozzle for which

$$\beta_2 = \left(1 - \frac{M_2(\gamma_2 - 1)}{2}\right)/\left(1 + \frac{M_2(\gamma_2 - 1)}{2}\right) \quad (2.3)$$

where γ denotes the specific heat ratio. Because of the occurrence of a 90 degrees bend in the injection chamber, almost total reflection is assumed at its entrance: $\beta_1 \sim 0{,}97$. The results were found to be quite insensitive for 10% - changes in the value of β_1.

Calculations were performed with the following values, corresponding to a typical test run at 0,9 MPa with 200 g/s oxidizer mass flow rate:

$c_1 = 310$ m/s	$c_2 = 800$ m/s
$M_1 = 0{,}38$	$M_2 = 0{,}19$
$\rho_1 = 9{,}6$ kg/m^3	$\rho_2 \sim 2{,}1$ kg/m^3
$V_1 = 120$ m/s	$V_2 \sim 60$ m/s
$D_1 = 15$ mm	$D_2 = 45$ mm
	$L_2 = 45$ cm
	$\gamma_2 \sim 1{,}2$

The inlet section consisted either of a tube with a length of 75 cm or of a diaphragm. The extent of the inlet was effectively enlarged by the injection chamber, e.g. 1,22 m at 55 Hz.
Therefore two L_1-values were investigated: 0,8 m and 2 m. The latter length will be seen to yield 55 Hz for the lowest mode, and therefore corresponds correctly to the 0,75 m inlet tube. The value of β_2 was calculated with the aid of Eq. (2.3); β_2 equals 0,963, while c_1A_2/c_2A_1 amounts to 3,4875.

Predicted frequencies were found to be independent of the value of α in the α-range 0-80 Hz. It is important to note that firstly frequencies, ν_j, were determined where the LHS of Eq. (2.2) attained minimum values, normally very close to zero. By tuning the value of α a bit, this LHS could subsequently be made zero at the same frequencies ν_j. This makes clear that the value of α is inessential if only possible frequencies are to be considered. The predominance of some frequency in reality is of course still determined by the actual damping rate.

The predicted frequencies for L_1 = 2 m are:

ν_1 = 55 ± 2 Hz
ν_2 = 115 ± 2 Hz
ν_3 = 180 ± 2 Hz
ν_4 = 240 ± 2 Hz
.
.
etc.

The predicted frequencies for L_1 = 0,8 m are:

ν_1 = 125 ± 2 Hz
ν_2 = 270 ± 2 Hz
ν_3 = 415 ± 2 Hz
.
.
etc.

More accurate determination of ν_j is possible, but the achieved accuracy was found adequate in view of the present level of approximation. The simple BASIC source listing that allows for the computation of these frequencies is given below.

```
10 L1=2
20 INPUT "Guess alfa";ALF
30 BETI = .98
40 PRINT "inlaatpijp van 15 mm has length";L1;" m"
50 B1 = EXP(-2*ALF*L1/265.2)
60 B2 = EXP(-2*ALF*.45/771.1)
70 FOR NU = 10 TO 500 STEP 5
80   OHM = 2*3.14159*NU
90   K1L1 = (L1/265.2)*OHM
100   K2L2 = .58 *OHM/1000
110   STUK1 = 4.4875 * (1 - .963*BETI*B1*B2 * COS(2*K1L1+2*K2L2) )
120   STUK2 = 2.4875 * (BETI *B1* COS(2*K1L1) - .963*B2 * COS(2*K2L2) )
```

```
130   PRINT "1 = ";STUK1;" 2 = ";STUK2
140   PRINT "freq = ";NU;" result = ";STUK1 + STUK2
150   NEXT NU
160 END
```

It is concluded that the lowest order acoustic standing wave modes have a predicted frequency that is strikingly close to the experimentally observed vortex shedding frequencies (see section 2.2):

$\nu_1 = 55$ Hz , $\nu_{exp} \sim 40$ Hz (elongated inlet)

$\nu_1 = 125$ Hz , $\nu_{exp} \sim 80 - 100$ Hz (diaphragm)

Note that the aft mixing chamber acts as a resonance cavity, although it was only modelled as an extension of the fuel grain with 15 cm. A proper accounting for this aft mixing chamber will surely lower the predicted frequencies.

The above analysis makes clear that the experimentally observed vortex shedding frequencies can be identified as corresponding to the first longitudinal acoustic mode associated with pressure oscillations in the ramjet.

2.4 CONCLUSIONS WITH RESPECT TO THEORETICAL MODELLING

- Shedding occurs of large-scale vortex structures from the recirculation zone downstream of the rearward facing step.

- This shedding is at a regular rate; frequencies are identified as those of the first longitudinal acoustic mode associated with pressure oscillations.

- The large-scale toroidal vortex structure retains its axisymmetrical shape while diffusing in the main stream. The phenomenon is two-dimensional in this sense.

- Combustion itself has probably no bearings to this vortex shedding. In other words : chemistry is probably not vital.

The pressure oscillations trigger the flow field and vortex shedding, while, vice versa, the flow field generates acoustic waves. This complicated interaction will not be described in the sequel. The flow field in a ramjet being essentially time ependent, it will rather be tried to set up a numerical scheme to simulate turbulence and flow development behind two-dimensional flow obstacles. Modelling the interaction with acoustical waves can be the next step of these theoretical investigations.

3 ON THE DIRECT SIMULATION OF VORTEX SHEDDING

3.1 INTRODUCTION

3.3.1 Purpose modeling

The prime physical situation under study is the confined flow in a solid fuel ramjet with a rearward facing step, although flow around immerged objects is also investigated.
The theoretical model aims to mimic the physical phenomena occurring at macro-scale in truly time-dependent flows as good as possible. Of particular interest is the built-up, shedding and diffusion of large vortex structures in recirculating flows. It is not attempted to compute boundary layers accurately. In this sense, the modeling is a direct simulation of free stream turbulence.

3.1.2 Choice of numerical model

Supported by our experimental observations (see sections 2.2.1 and 2.4), the flow can be concidered as two-dimensional and adiabatic. The phenomena to be studied are essentially time-dependent in nature, and convection has to be modeled accurately in regions away from boundry (see section 3.1.1).

It was felt that some numerical k-ε and algebraic closure models, although frequently applied in technical research, to some extend mistify the phisics and also do not handle properly all features of particular importance for typical time-dependent phenomena. A basically different approach, a Lagrangian vortex method was therefore chosen (see [9] for example).

It is well known that the vortex type of modeling can treat well free convection and conditions at infinity, and is capable of predicting Strouhal numbers quantitatively correct [10]. Only pure viscous (parts of the) flows are demand much effort to account for properly.
No grid is needed, and the calculation procedures do not require a large memory capacity.

Essential features of the vortex model are discussed in section 3.2. The technical details of the numerical solution procedures that are typical for this study are discussed is section 3.3. Those details that are eather more common knowledge or rather straightforward are only briefly discussed in sections 3.2 and 3.3; the open literature (see [9] through to [11] for example) elaborates these parts.

3.2 GOVERNING EQUATIONS AND MAIN FEATURES OF THE COMPUTATIONAL MODEL

3.2.1 Helmholtz decomposition and vorticity transport

The velocity vector field, $\underline{v}$, is decomposed in the Helmholtz way:

$$\underline{v} = \underline{\nabla}\varphi + \underline{\nabla} \times \underline{\psi} \qquad (3.1.)$$

with $\underline{\nabla} \cdot \underline{\psi} = 0$

In some cases a unique decomposition exists; for example if the field $\underline{v}$ is defined in infinite space, is differentiable such that $\underline{\nabla} \cdot \underline{v}$ and $\underline{\nabla} \times \underline{v}$ vanish of order r^{-3}, the unique decomposition reads [12]:

$$\underline{v}(x) = -\underline{\nabla}_x \iiint_{R^3} \frac{\underline{\nabla}_y \cdot \underline{v}(\underline{y})}{4\pi\,|\underline{x} - \underline{y}|}\, dV(\underline{y}) + \underline{\nabla}_x \times \iiint_{R^3} \frac{\underline{\nabla}_y \times \underline{v}(\underline{y})}{4\,\pi\,|\underline{x} - \underline{y}|}\, dV(\underline{y}) \qquad (3.2)$$

In the limit, the second term on the RHS yields the Biot-Savart law for the velocity induces by a vortex filament. We extend our flow field into all space, and assume incompressible flow (for a start, although compressible flow will not be dealt with in this report), i.e.:

$$\underline{\nabla} \cdot \underline{v} = 0.$$

In this case the first term on the LHS of Eq. (3.2) vanishes.
Denoting the vorticity part of the flow by ω, defined as ($\underline{v} = U\hat{x} + V\hat{y}$)

$$\omega = V_{,x} - U_{,y} = -\nabla^2\psi = \underline{\nabla} \times \underline{v} \qquad (3.3)$$

the Navier-Stokes equation is written in a form that does not contain pressure any more:

$$\omega_{,x} + \underline{v} \cdot \underline{\nabla}\omega = \nu\, \Delta\omega$$

This transport equation for the vorticity is the main convection governing equation. It will primarily be solved with the neglection of the viscosity term on the RHS, after discretizing the vorticity.

3.2.2 Discretizing vorticity

The continous vorticity field is split up into a set of descrete vortices, each with the same shape function, γ:

$$\omega(\underline{r}) = \sum_{i=1}^{Nvort} \Gamma_i \, \gamma(|\underline{r} - \underline{r}_i|) \tag{3.5.}$$

The core shape γ was defined by $\sigma^2/\{\pi(r^2 + \sigma^2)\}^2$. This function is infinitely differentiable, and gives a smooth velocity field everywhere. The velocity at point (x,y) induced by the vortices situated at locations indicessed by i can be computed from the Biot-Savart law. The so-called undisturbed velocity, U_{undist}, at infinity is superposed. The discretizing of the vorticity makes it possible to derive the velocity from (see Appendix 1):

$$U(x,y) = U_{undist_x} + 0{,}5 \frac{1}{\pi} \sum_i (y_i - y) \, \eta(|\underline{r} - \underline{r}_i|) \tag{3.6}$$

and

$$V(x,y) = U_{undist_y} - 0{,}5 \frac{1}{\pi} \sum_i (x_i - x) \, \eta(|\underline{r} - \underline{r}_i|) \tag{3.7a}$$

where η is defined by

$$\frac{d}{dt} t^2 \eta = 2\pi t \, \gamma \tag{3.7b}$$

These formulae are used to derive the propagation speed of each individual vortex.
Near the wall the result is used to annihilate some given velocity, which leads to a set of algebraic equations that have to be solved simultaneously.

The convection of vortex blobs is accurately described by integrating the above velocity components over short intervals of time.

3.2.3 Discretizing time

Flow evolution is computed by integrating over consequetive time steps, where the consequences of time discretisation are to be considered.
After each timestep, at anomalous instants, immerged bodies yield a kind of impulsive start problem. It is well known for this problem that not all boundary conditions can be satisfied, resulting in vorticity being generated. The generation of vorticity in this respect is a mere consequence of the discretization of time in the presence of solid boundaries.

At other instants, i.e. during the timesteps, the vorticity diffuses. A typical thickness of a vortex sheet after time Δt is the square root of the product of the kinematic viscosity, ν, and Δt. In the first version of the model the diffusion was not accounted for, leading to results that will be discussed in section 3.4.

In this way the time-history has become piece-wise continuous, allowing for integration of the evolution equations. By such integration new locations of vortices are established.

3.2.4 Creation and annihilation of vortex blobs

Each solid boundary is discretized by defining a finite set of so-called wall points. To each wall point corresponds a creation point in the flow area. At each creation point after each timestep a vortex is initiated in a way that will be explained in section 3.3.4.
The distance between a wall point and the correspondin creation point is related to the diffusion time, Δt. There is some arbitrariness in defining this distance, although it can be compared to displacement or momentum boundary layer thicknesses. The establishing of a justified correspondance between wall- and creation point was one of the aims of this study.

If a vortex blob after being transferred appears in the region between the solid boundary and creation points or within the solid, it is simply deleted (see Figure 3.1). The strength of its vorticity is conserved, however, by spreading out all deleted vorticity over the 'new' vortices that are to be created at this instant of time. It implies an extra condition for the creation of vortices.

If two vortices, indicessed by 1 and 2 respectively, come close together they are merged into a single one. Preservation of total circulation and lineair momentum require that the resulting vortex with strength $\Gamma = \Gamma_1 + \Gamma_2$ should be located at:

$$(x,y) = \frac{\Gamma_2}{\Gamma} (x_1, y_1) + \frac{\Gamma_2}{\Gamma} (x_2, y_2)$$

Details of annihilation and merging will be discussed in section 3.3.3.2.

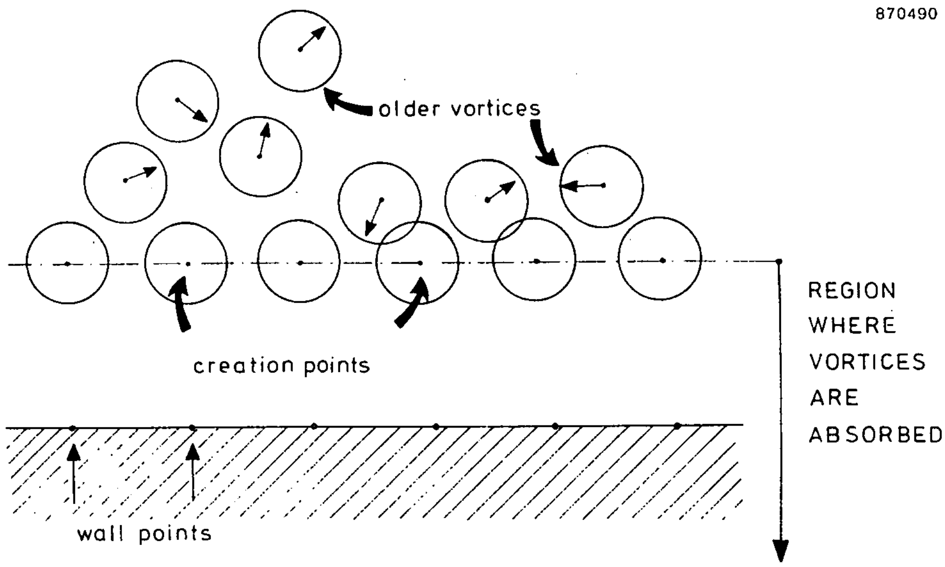


Figure 3.1 Schematic of vorticity manipulating near solid boundaries.

3.2.5 Calculation flow chart

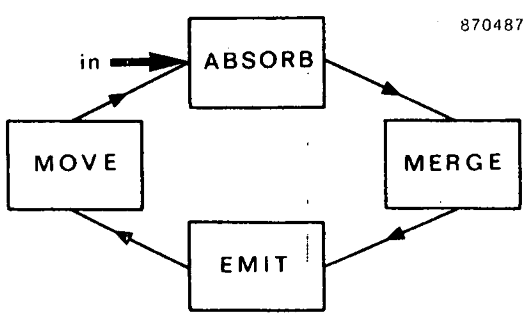


Figure 3.2 Sequence of the physical procedures in the model.

Figure 3.2 summaries the main procedures of the model. 'Absorb' keeps track of all deleted vorticity; in 'Merge' vortices are merged and the total number of vortices is adjusted, whereafter in 'Emit' new vortices are created satisfying conservation of total vorticity.
The last step of an iteration cycle is the transportation of vortex blobs towards new places in' Move'. To this end the evolution equation is integrated.

3.3 DETAILS OF THE MODEL

3.3.1 Main numerical solution procedures

A matrix solver is needed to simultaneously solve the set of independent algebraic equations for propagation speed of induced vortices (see section 3.2.2), completed with the equation for conservation of total circulation (section 3.2.4). Solvers for several applications are presented in appendix 2.
Appendix 3 contains explicit computation procedures for velocity components (see section 3.2.2 and appendix 1).
The structure of the programme (see section 3.2.5) is apparent from appendix 4.

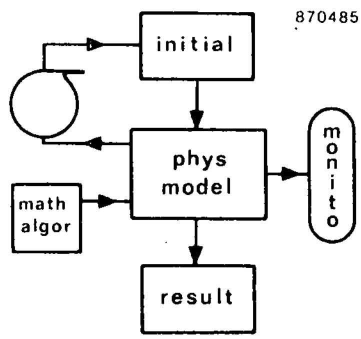


Figure 3.3 Schematic of main procedures in the model.

Since time evolution is computed step by step, the results of each computation may be the starting point of new computations. Therefore the frequent storage of data occurs in the schematic of main subroutine blocks around the physical model. (see Figure 3.3) Appendix 4 presents also the source listings for storing and reading.

3.3.2 Monitoring

3.3.2.1 Flow visualisation After each time step thousands of new locations and velocities are usually computed. For quick and accurate testing it is therefore important to have convenient ways for flow visualisation and data representation.
Calculation procedures were directly validated withthe aid of output data files (see Figure 3.4), whereas plotting capacity was exploiting by routines to compute and draw flowlines, routines to plot vortex blob locations with arrows

indicating direction and magnitude of the vortex velocity, and routines to depict velocity arrows at selected grid points (see appendix 5).

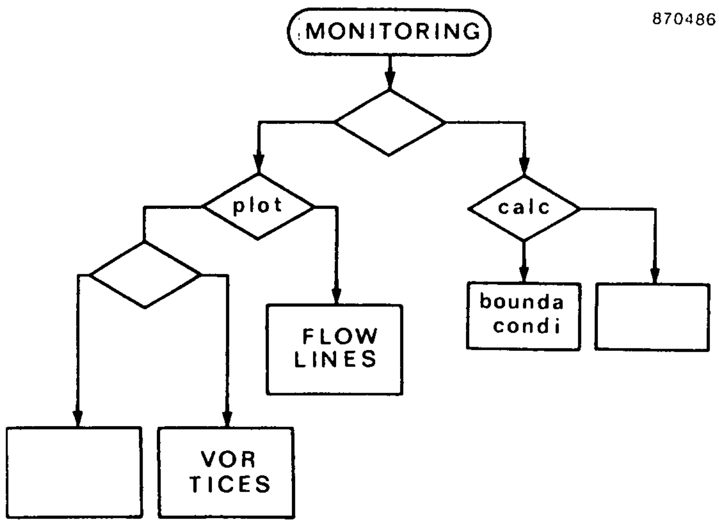


Figure 3.4 Schematic of monitoring utilities flow chart.

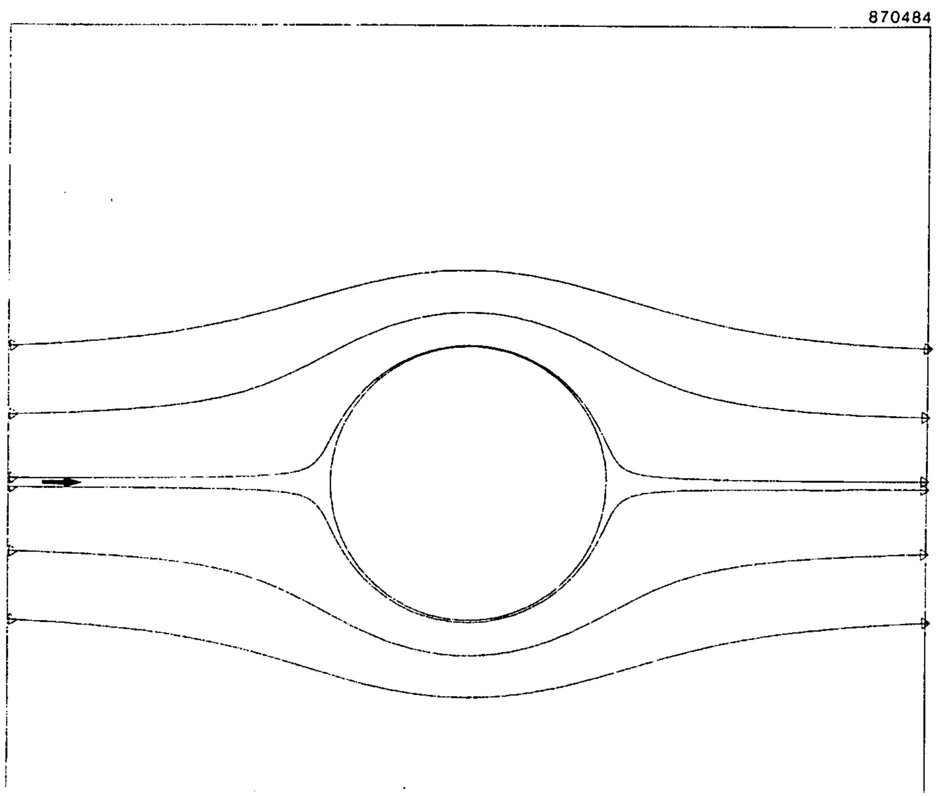


Figure 3.5 Potential flow lines around a cylinder.

In this figure 3.5 potential flow around a cylinder is indicated by computed flow lines. If only a part of the cylinder is retained one gets the circle segment or 'wing' that was often used as a test case in this study (see figure 3.6)

8704116

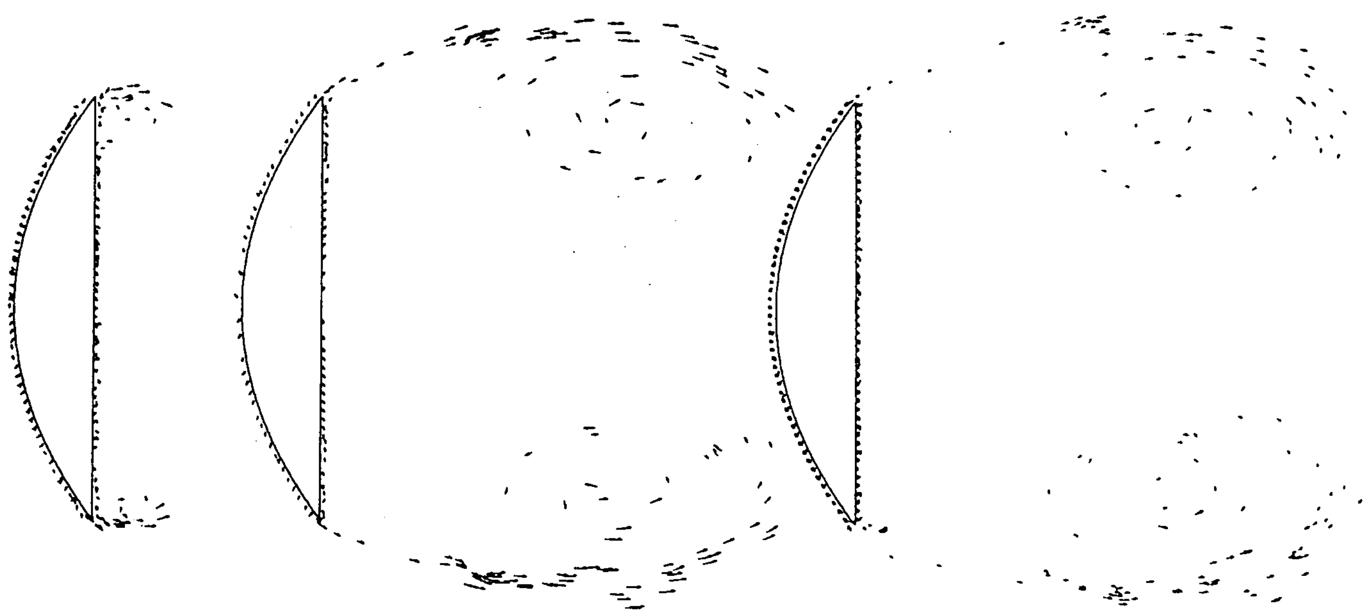


Figure 3.6 Flow development behing an obstacle suddenly immerged in a uniform flow.

From the sequence of three pictures it is clear that vortices are generated at points close to the solid boundary, and are transported by the uniform approach velocity from the left towards the right. Two hundred creation points were used. At time zero only the uniform velocity was present, and the object is suddenly introduced is the flow.

Clearly two large vortex structures are formed downstream of the object. The vortex blobs themselves serve as flow tracer in Figure 3.6.

Much later in the development of the flow, the distance between the object and vortices becomes that large, that rescaling of the plot is necessary. This results is a deformation of the object in the resulting picture (see Figure 3.7); more examples of the rescaling will follow in section 3.4.1.

If only some flow lines are retained in figure 3.7, the picture is more easily to interpret (see Figure 3.8). It is therefore convenient to be able to compute flow lines accurately and at low cost of computation time. Some alternative methods were therefore examined.

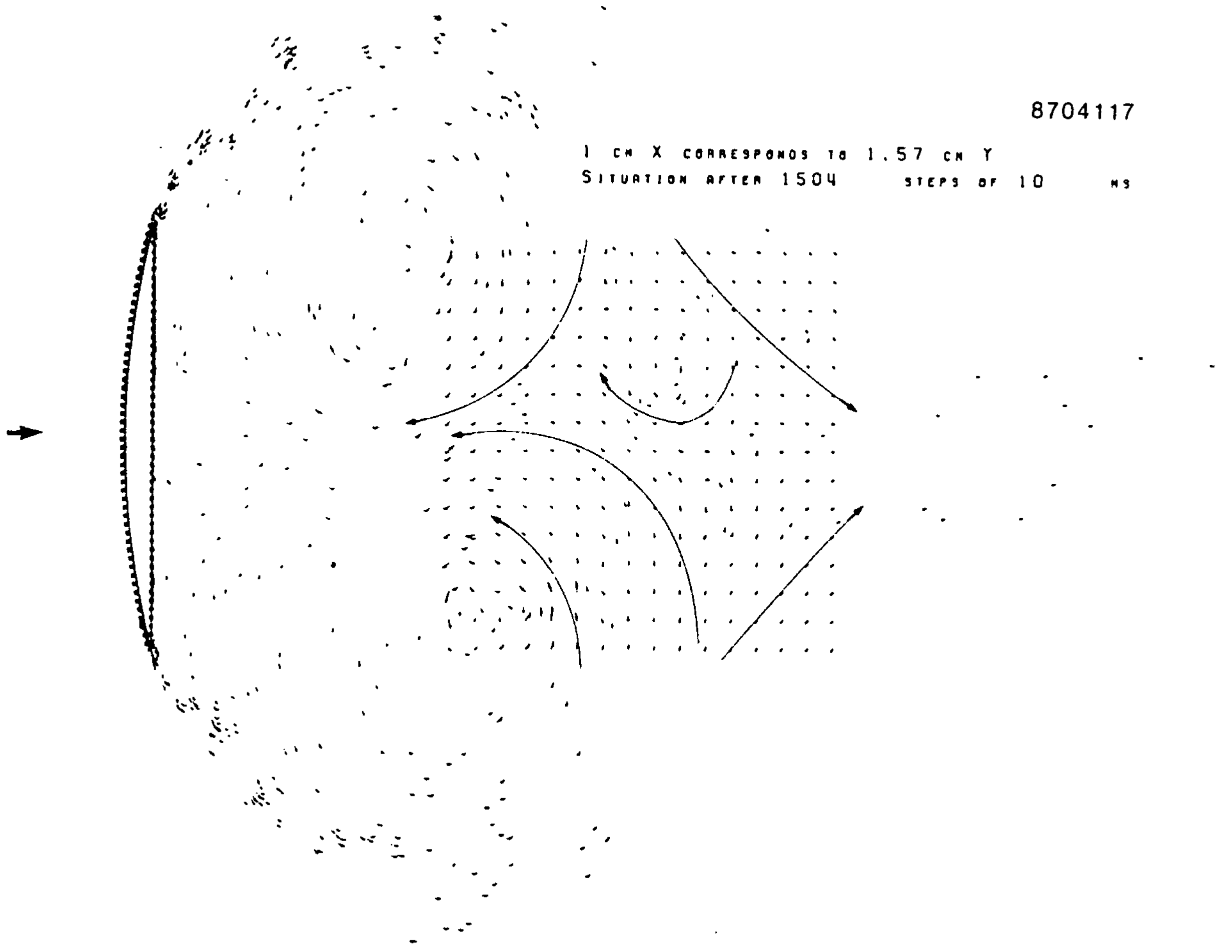


Figure 3.7 Vortices downstream of the object of Fig. 3.6. after 1504 computation steps; see also fig 3.8.

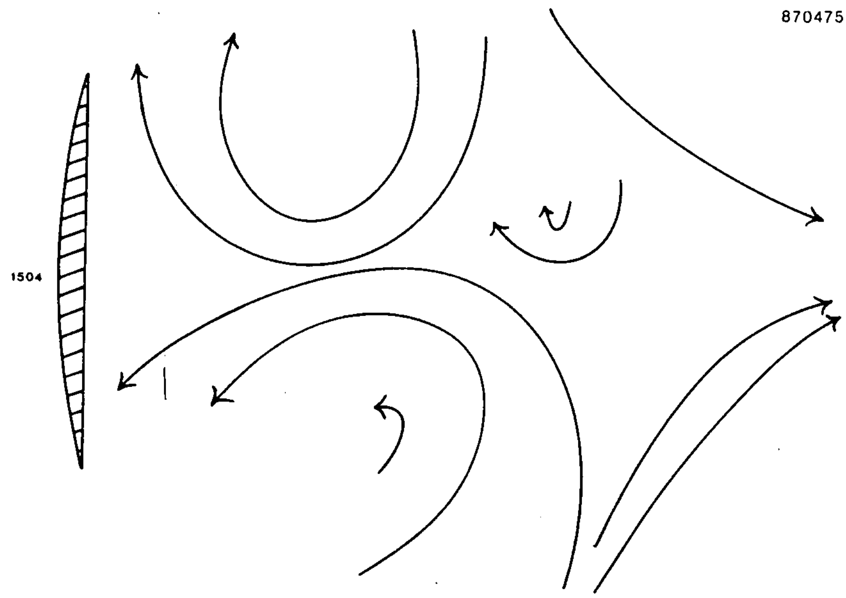
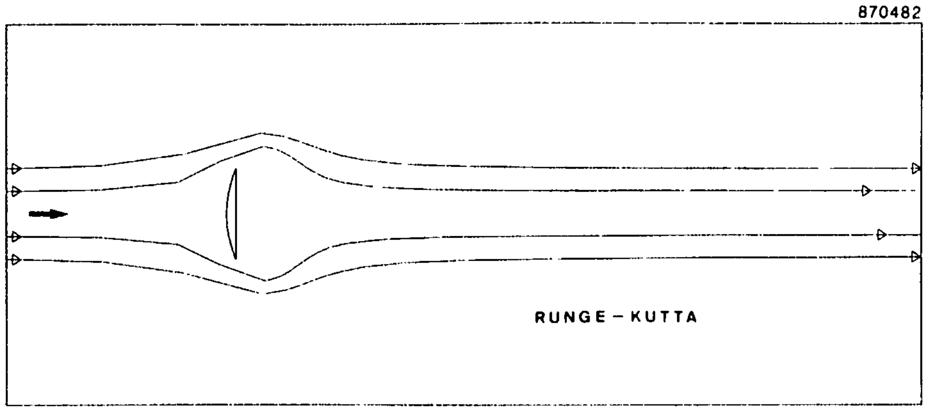


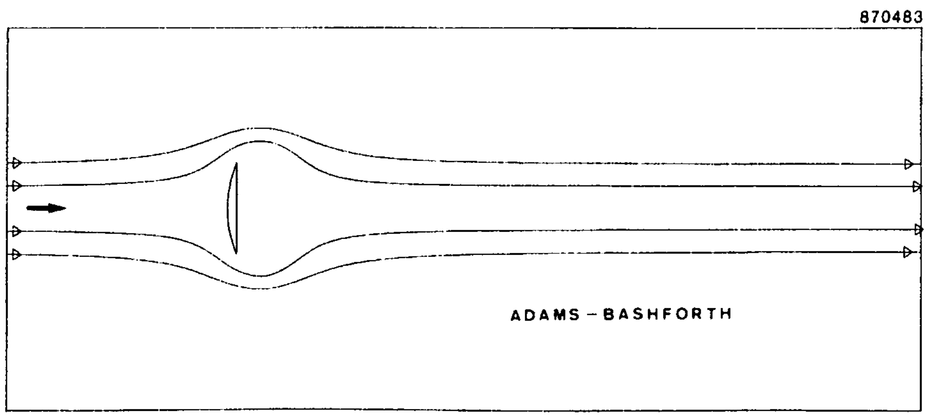
Figure 3.8 Flow lines downstream of an immerged object; see also fig. 3.7.

<u>3.3.2.2 Comparison of flow line computation methods</u> Flow lines can be computed for arbitrary starting points. From computations like the ones presented in Figures 3.9 and 3.10. the following conclusions are drawn.



<u>Figure 3.9</u> Flow lines around the immerged object of fig. 3.7.; Runge-Kutta method.

Each consequetive, individual point of a stream line is in most cases more accurately computed with a fourth order Runge-Kutta method is than with an Adams-Bashforth or Adams-Moulton method.



<u>Figure 3.10</u> Flow lines around the immerge object of Fig. 3.7; Adams-Bashforth method.

However, with a specified amount of computation time, fewer points are computed with the aid the Runge-Kutta methods. The total computation time for figures 3.9 and 3.10 was the same, resulting in 'broken' streamlines in a Runge-Kutta plot, and a smooth, realistic appearance in Adams-Bashforth plots. The latter method yields more plotting points per CPU-second.

This observation and the good recursive conditional stability of Adam-Moulton methods have led to the abandonment of Runge-Kutta methods for purpose of flow line visualization.

3.3.3 Computation time reduction

3.3.3.1 Some alternatives for computation time reduction After each time step a set of linear equations has to be solved (see section 3.3.2). The matrix inversion that is needed is carried out only once (see appendix 1), since the matrix is only dependant on the contour(s) of the object(s). Still the solving of the set of equations is very time consuming. The time is roughly proportional to N_{vort} squared, where N_{vort} denotes the total number of vortices.

There are four ways to substantially reduce computation time.

1 Limit the number of vortices

2 Optimize the time step

3 Optimize sequential order of actions; e.g. absorb vortices before merging them. This order seems obvious, but not always has been applied in similar computation programmes made elsewhere

4 Adapt matrix solver for spatial symmetry in the physical problem. Such symmetry leads to Toeplitz or Hankel blocks in the matrix, allowing for the economizing of the solver. This was prepared theoretically, but due to lack of time not implemented.

The time step cannot be increased at will. During a time step vortices are not allowed to be transported far, relative to one another, to prohibit an imbalance of interaction. Also the total number of absorbed vortices may not be too large since otherwise strong new vortices are created that disturb the inner region in the proximity of the wall. Generally the time step should therefore not exceed ds divided by a typical velocity, where ds denotes the spacing between consequetive wall points.

The number of vortices can be controlled by allowing them to diffuse by the action of viscosity. This is work in progress. The number can also be controlled by adjusting the merging parameter. This is subject of the next section.

3.3.3.2 Manipulating the merging process In order to limit the total number of vortices and the computation time, the merging parameter is adjusted in two ways:

- by making it time dependent;
- by allowing spatial inhomogeneouity, i.e. by making it locus dependent.

The merging parameter that is used for manipulating the merging process is a scalar. Let N_{vort} denote the actual number of vortices, and N_{des} the desired number of vortices. The merging parameter is solely dependent on the difference $N_{vort} - N_{des}$, and is automatically adjusted during computation. The bigger the parameter, the more vortices are merged.

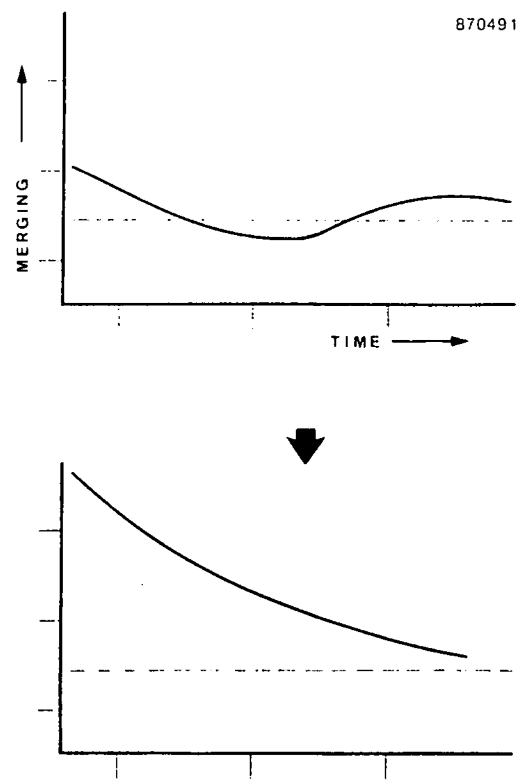


Figure 3.11 Schematic of two time evolutions of the merging parameter.

Figure 3.11 shows the actual time history of the merging parameter, It is seen to take up a value, indicated by the dotted horizontal line, around wyich it oscillates. The lower part of this figure shows how this oscillating is reduced by selecting another initial value of the merging parameter and by changing the automatic adjustment procedure. The initial value must be high as compared to the eventual, 'pseudo-stable' value. In order to create the 'damped' merging one must have some indication of the eventual 'pseudo-stable' merging parameter value. Which can be found by making a few explorative test runs for a given configuration of boundaries.

This stabilized, 'damped' merging has another advantage. The number of voritces is high if away from solid boundaries, while the number or vortices very close to these boundaries remains stable and low. Also the total number of vortices is reduced as compared to the case without 'damping'. This spatial distribution of vortices favours a correct representation of the free flow field, which is the main objective of this study.
The 'damped', initially high merging parameter stimulates vorticity spreading and keeps N_{vort} low, and was therefore applied.

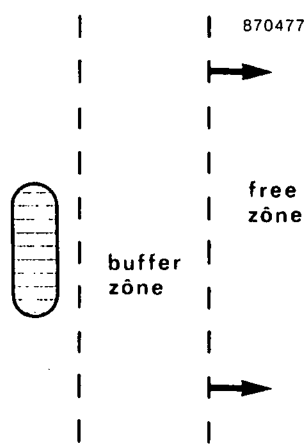


Figure 3.12 Schematic of flow regions behind an obstacle.

The region outside solid boundaries is divided into three regions (see Figure 3.12):
1 one so-called free zone, relatively far away from each boundary;
2 the region near to the wall;
3 the space inbetween, the so-called buffer zone.
From time to time extra merging is forced in the free zone by means of a 'sweep-over': after about 30 steps the value of the merging parameter is enlarged solely in the free zone and only for one timestep. In this way the total number of vortices in the free zone remains limited. In the free zone the flow structures usually are not very comlicated, whence not many vortices are needed to visualize the flow field there. The 'sweep-over' procedure entails blobs with

high vorticity strength, but hardly affects flow visualisation and, more important, has virtually no effect on the generation of vorticity at the boundaries. In fact, the general merging procedure is such that the velocity fields before and after merging are practically identical at large distances from the merging vortices.

3.3.4 Generation of vorticity at boundaries

3.3.4.1 General features A jump in velocity across a solid boundary can be simulated with help of a continual vortex sheet. It has been shown [11] that if a vortex sheet is used to satisfy the normal velocity condition globally along a simply connected region in two-dimensional space, the no-slip condition also is automatically and globally satisfied. This eliminates the need for image vortices to satisfy the no-slip condition in this case.
Imaging is often used to satisfy the normal velocity condition globally, but image vortices can only be used after comformally transforming the body, e.g. into a circle. This makes imaging difficult to apply in many situations.
Note that in reality the action of viscosity in a boundary layer takes account of the no-slip condition. It creates a continuous 2-D (or 3-D) distribution field of vorticity. Hence a one-layer model is a rather crude representation of reality, where in essence infinitely many layers occur.

It was found in this study that near sharp edges where much vorticity enters the free zone, the no-slip condition is not satisfied at locations inbetween the 'wall-points' due to the discretising of the vortex sheet.
This observation, together with the crudeness of the one-layer model, led to a study of how to better simulate vorticity generation at boundaries. Several ideas were examined, and results will be discussed in the next section.

It is noted that if vortex blobs are created at 200 creation points, only 199 distances and hence only 199 interaction equations have to be satisfied. An independent condition is therefore needed*).The conservation of total vorticity is used for that (see section 3.2.2).

So the physics of vorticity generation, for example at a cylinder, is mimiced by the conditional vortex generation given by total vorticity conservation and the normal velocity condition at a set of discrete wall points. No condition is used to define loci where boundary layers detach from the body. Since diffusion and

*) Only the case if a stream function is made constant on the surface.

transport of vorticity near the wall is simulated rather crudely, the model may not be expected to produce quantitatively correct results in the near-wall region.

<u>3.3.4.2 Improving the method to generate vorticity</u> In pure solenoidal flows it is possible to define a stream function ψ. Since a solid surface is a streamline, it is possible to satisfy the normal velocity condition by solving the set of N_{wall}-1 differential equations $d\psi = 0$.

A second way to generate vortices at the N_{wall} creation points is to satisfy the condition of zero normal velocity, $V^{\perp} = 0$, at all wall points (see also section 3.3.4.1), except for one (**).

The third modus of vortex generation is to make both the perpendicular and the transverse velocity components erqual to zero at all wall points. In this case the number of creation points exceeds the number of wall points by a factor two. In all three generation procedures the conservation of total vorticity is the N_{wall}'th equation.

Several alternatives to locate creation points relative to wall points were examined for both the first and second modus of vorticity generation. To compare the results two selection criterions were invented:

-1- in order to mimic the physics closely, vortex j should be determined mainly by the normal velicity condition at the nearest wall point j;

-2- in order to effectuate a smooth vortex generation without excessively large velocities close to the wall, time-discretizing should induce only low values of $\Delta\Gamma/\Delta t$, Γ being the local vorticity and Δt a time step. Let Γ_i be the strength of the vorticity created at point creation point i.
The above implies that the sum of all $|\Gamma_i|$ should be small.

With the aid of these criterions several configurations of creation points were examined. The configuration depicted in Figure 3.13 is rather trivial, and does a good job in simulating flow over flat plates.

However, it is easy to see that selection criterion 1 is not fulfilled: a vortex created at creation point i contributes only a longitudinal velocity component to wall point i. So if flow is perpendicular to the surface, only the creation points further away from the wall point i contribute.

(**) One equality must be dropped if total vorticity should be conserved.

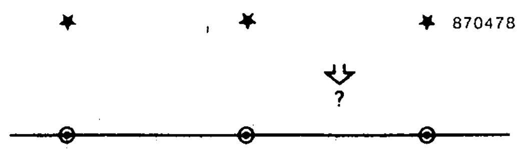


Figure 3.13 Schematic of configuration of wall- and creation points.

Close to slight curved surfaces the set of creation points can be considered to be a straight line. If all initiated vortices have the same strength γ and are positioned at a straight line at distance δ from each other, at $y = 0$ in a rectangular coordinate system (x, y), the x-velocity component, u, has at infinity the value $- \pm 0{,}5\ \gamma/\delta$ (Lamb, [13]) with the '±' sign corresponding to $y = \pm \infty$. This array of vortices simulates a continuous vortex sheet of uniform strength γ/δ. At other locations the velocity components u and v are obtained from:

$$u = -0{,}5\ (\gamma/\delta)\ \sinh(2\pi y/\delta)\ /\ \{\cosh(2\pi y/\delta) - \cos(2\pi x/\delta)\}$$
$$v = 0{,}5\ (\gamma/\delta)\ \sin(2\pi x/\delta)\ /\ \{\cosh(2\pi y/\delta) - \cos(2\pi x/\delta)\}$$

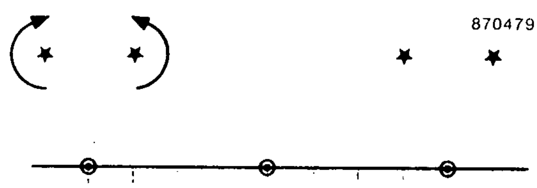


Figure 3.14 Schematic of configuration of wall- and creation points.

Suppose that two creation points are located directly above one wall point that has two neighbours without creation points above them, and that this configuration is repeated toward infinity (see Figure 3.13). Let each vortex located on the creation point 'on the left' of a wall point have strength k_1, and all other vortices have strength k_2. By superposition the normal velocity component at a wall point directly underneath two creation points is found to have the value

$$0{,}001867\ \{k_1 - k_2\}/(2d)$$

where d denotes the separation distance between two wall points. At neighbouring wall points the value

$$0,001317 \{k_1 - k_2\}/(2d)$$

is calculated. so purely longitudinal flow is simply dealt with by choosing k_1 equal to k_2, resulting in about the same flow pattern as generated in Figure 3.13.

However, purely perpendicular flow is now easily accounted for by tuning the values of k_1 and k_2.

In actual simulations the vortex layers are finite, whence the above computations are merely indicative.

In actual simulations the configuration of creation points of Figure 3.14 was found to have much better performance than the one of figure 3.13, in terms of the selection criteria described above. Further changing of the distance between creation points did not improve results.

Double vortex layers were also investigated, but without succes.

The configuration as depicted in Figure 3.14 favours the simulation of vorticity generation near walls, and was henceforward applied.

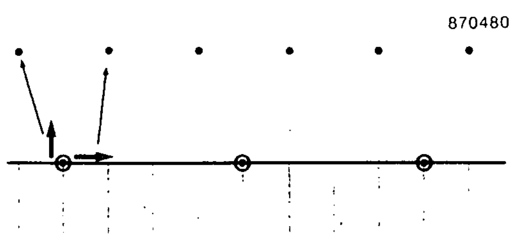


Figure 3.15 Schematic of configuration of wall- and creation points.

If high velocity gradients occur it might be wise to satisfy both the normal and the tangential velocity conditions at each wall point (see Figure 3.15). Such might be the case with penetrating shear layers, e.g. a 2-D channel with sudden expansion.

3.4 FLOW CALCULATIONS

3.4.1 Uniform flow disturbed by an immerged wing

The development of flow behind the circle section of Figures 3.6 and 3.7 annex 3.8 was further studied, employing the configuration of creation points of figure 3.14 (see section 3.3.4.2).

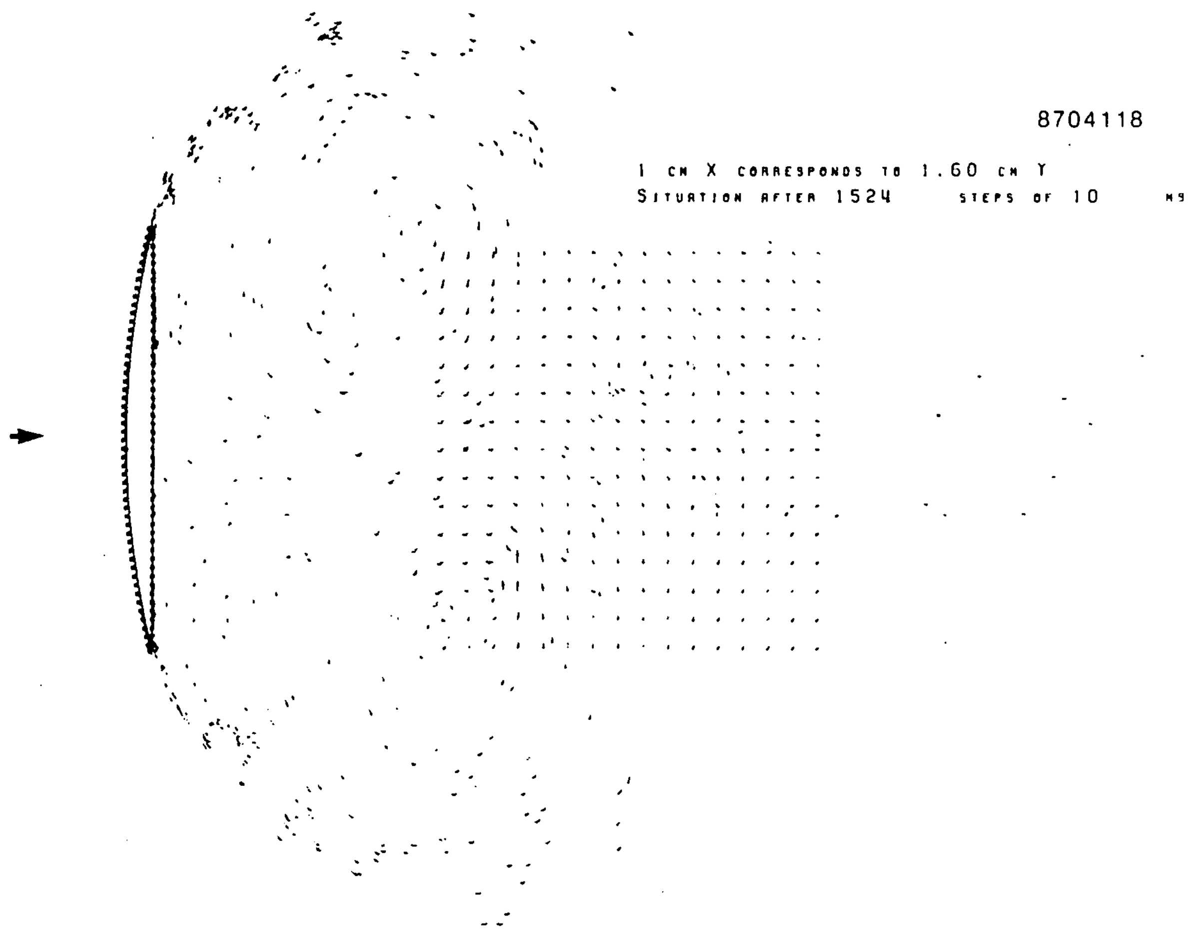


Figure 3.16 Flow due to immerging an object in a uniform flow; see Fig. 3.7. Location and velocity of vortex blobs after 1524 time steps.

In considering figures 3.16 through to 3.21 it should be remembered that rescaling occurs in the flow direction of the 'undisturbed' uniform approach velocity at infinity U_∞. The scaling is quantified in the figures via an x-y correspondance; in the x-direction perpendicular to the flow direction no scaling was applied.

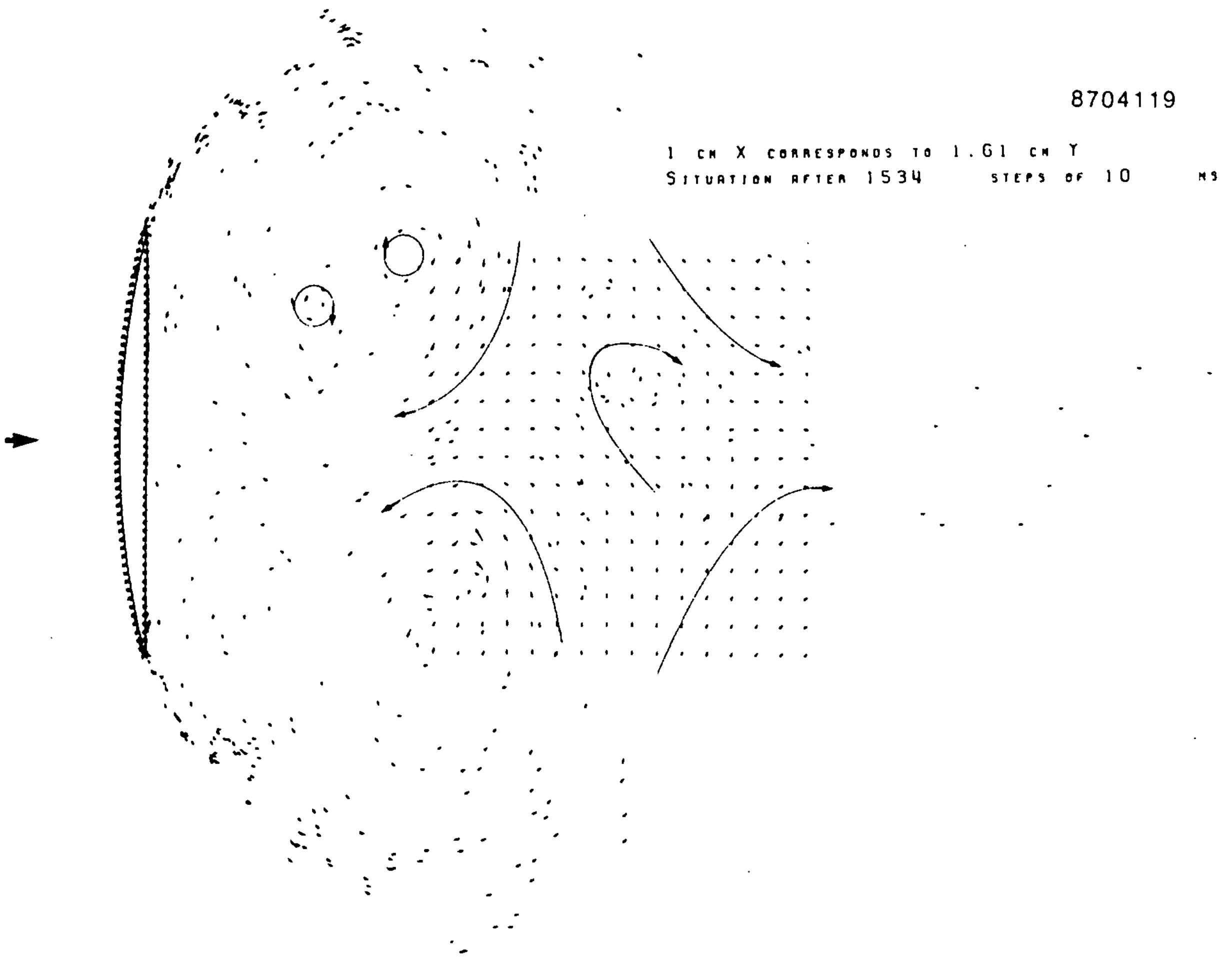


Figure 3.17 Flow due to immerging an object in an uniform flow; see fig. 3.7. Location and velocity of vortex blobs after 1534 time steps.

It is also noted that vortices serve as flow tracers and at the same time generate the flow field. The strength of a vortex can not be deduced from the figures, but the vortex velocity is proportional to the arrow length.
In some subsequent calculations the velocity was computed on the nodes of an arbitrary rectangular grid. The grid is easily recognized in the figures 3.16 through to 3.20. The grid arrows help to identify streamlines in the far field

zone, where due to merging vortices with high strengths appear ('sweep-over' see section 3.3.3). Such vortices are insufficient to serve as flow tracers.

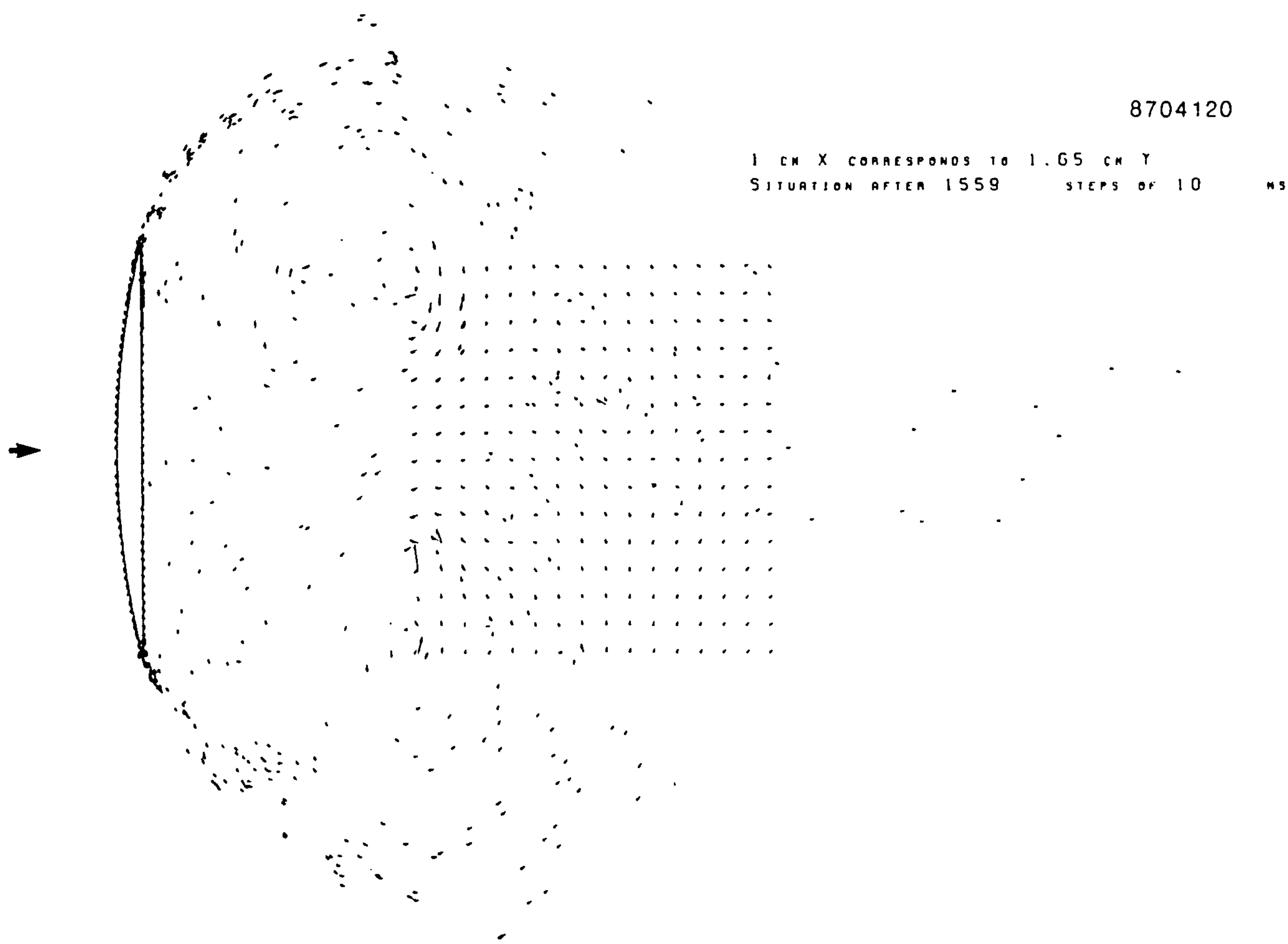


Figure 3.18 Flow due to immerging an object in a uniform flow; see Fig. 3.7. Location and velocity of vortex blobs after 1559 time steps.

Figure 3.16 clearly shows that emitted vortices, once away from the object, are transported downstream by the uniform approach velocity. Then the vortices may be catched by vortex structures with its centres, 'eyes', positioned behind the tips of the object. Further downstream a third, somewhat smaller vortex structure appears.

Figure 3.18 shows that at later times the eyes or the vortex structures shift relative to one another. The flow then resembles a Von Karman vortex street, indicating flow in the transitional Reynolds number region.

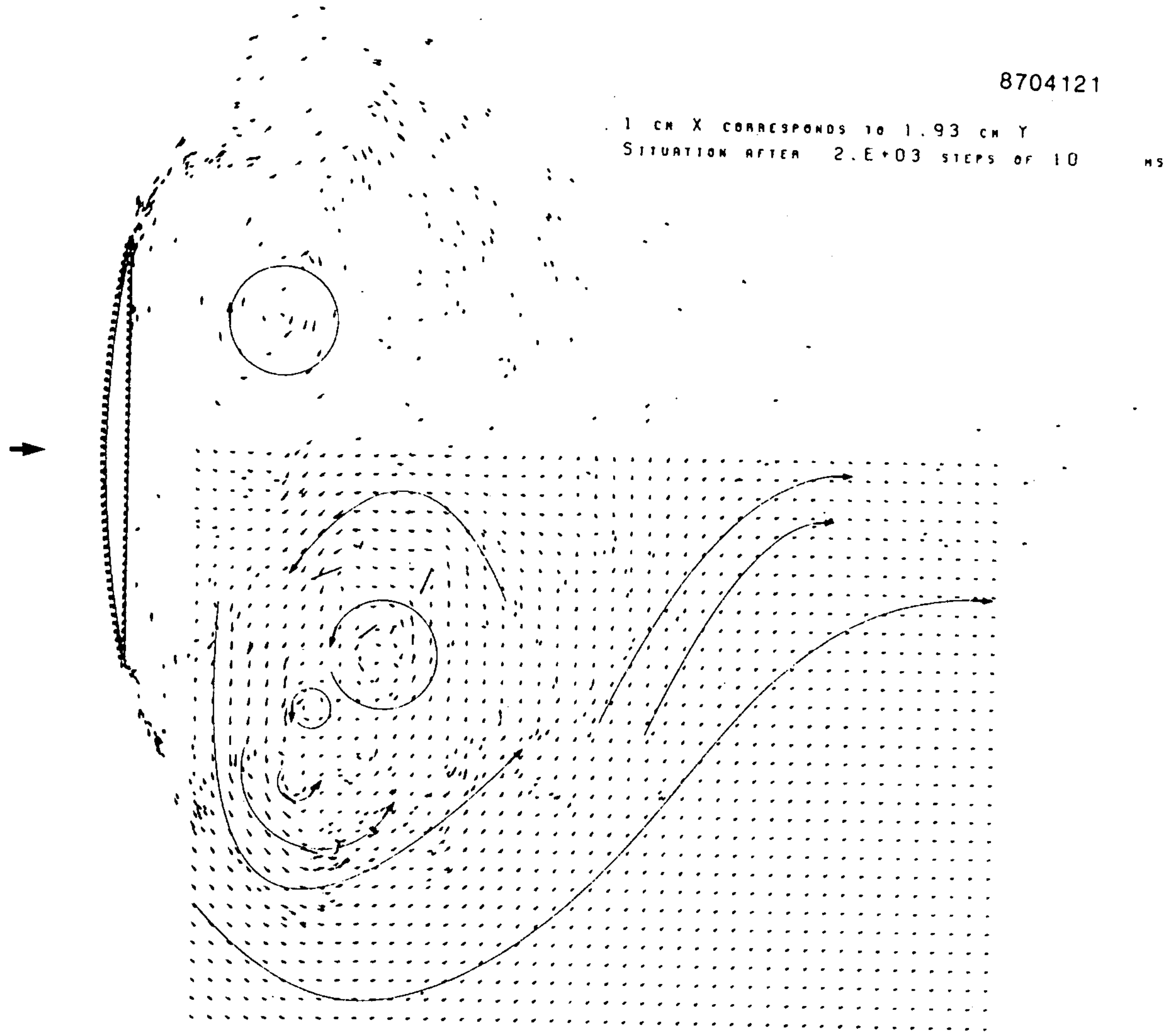


Figure 3.19 Flow due to immerging an object in a uniform flow; see Fig. 3.7. Location and velocity of vortex blobs after 2000 time steps.

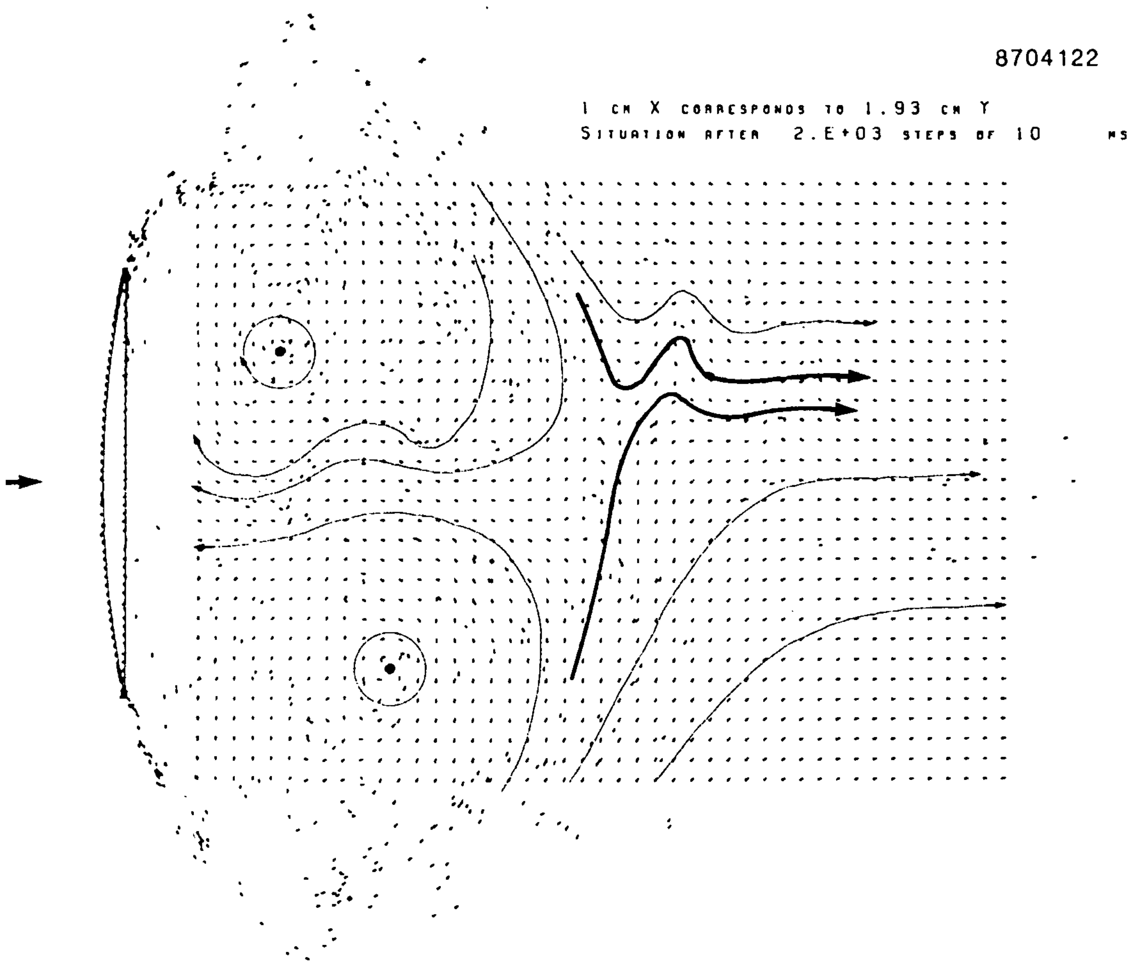


Figure 3.20 Flow due to immerging an object in a uniform flow; see Fig. 3.7. Grid with velocity vectors after 2000 time steps.

At later times, see e.g. Figure 3.21, the flow structure becomes manifold and even a bit chaotical. This is probably due to the fact that the diffusion and annihilation of vorticity by the action of viscosity was not yet incorporated in this model, although the incorporation is easily done. The only mechanism now available to redistribute the vorticity field is the merging of discrete vortices, but this procedure leads to stronger vortices. This mistifies the actual physics a bit, and does not lead to correct flow visualisation.
The diffusion by viscosity also controls the dependance on Reynolds number resulting in improved simulations. Lack of time prohibited the accounting for viscosity.

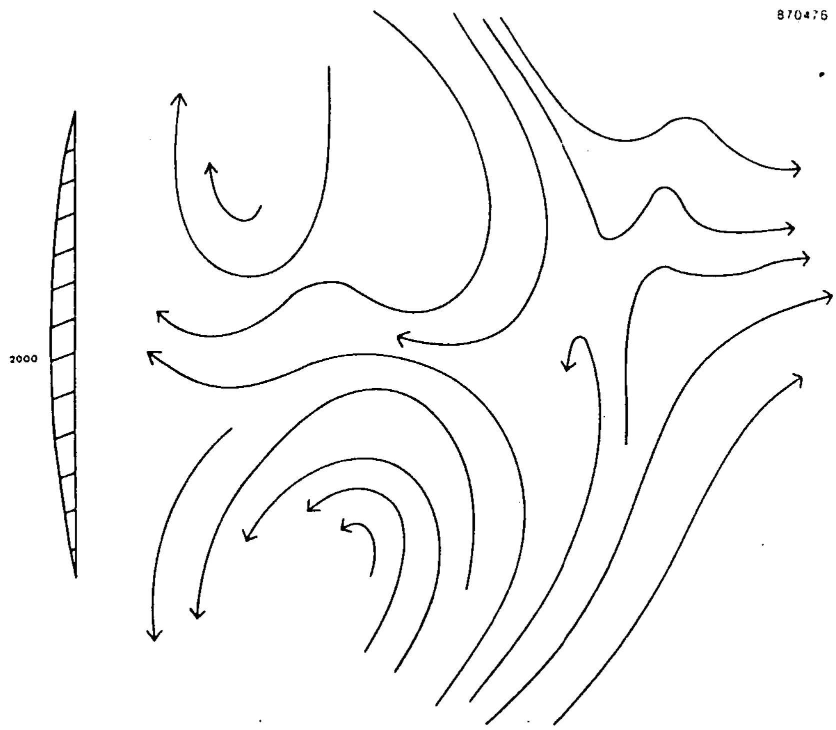


Figure 3.21 Flow due to immerging an object in a uniform flow; see Fig. 3.7. Stream lines after 2000 time steps.

3.4.2 Two-dimensional channel with rearward facing step

Irrotational flows can be characterized with a velocity potential. For two-dimensional irrotational and solenoidal flows a complex potential exist. In this case the flow in many configurations can be derived with the aid of the Schwartz-Christoffel transformation. However, this transformation is in terms of integrals. If an irrotational and solenoidal flow is to be used as 'undisturbed' flow part in a vortex model computation, it is desirable to analytically solve the integrals in order ot avoid excessive computation time.

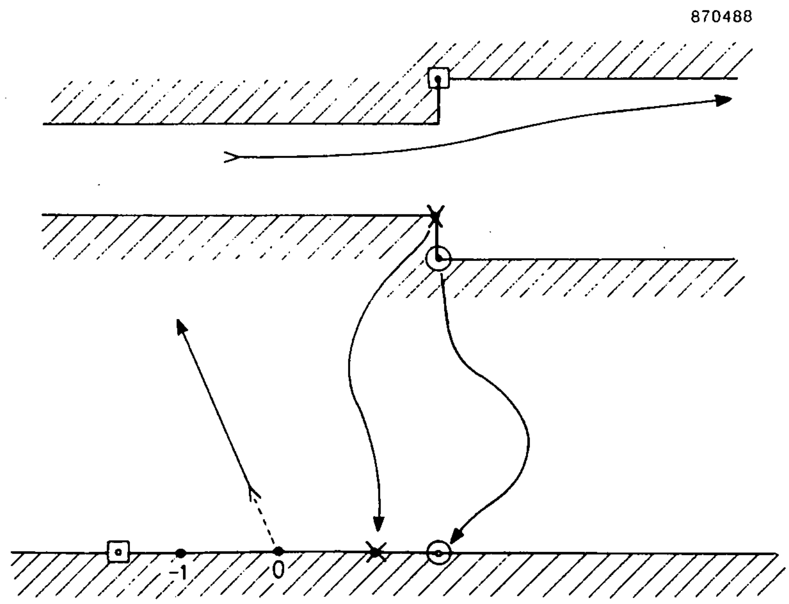


Figure 3.22 Schematic of two complex planes and transformation.

This was done for the case of a channel with a rearward facing step. Let z denote x + iy, and let p be a complex coordinate in the plane in which the transformed flow area of the pipe flow with a sudden expansion occupies exactly half the plane (see Figure 3.22). The flow at infinity in the z-plane is represented in the p-plane by a source in the centre, for which the complex potential is well known. The primitivating of the integrals in the Schwartz-Christoffel transformation yielded the following transformation from the p-plane, where the flow field is known, to the z-plane.
Scale the y-axis in the z-plane such that the smaller tuberadius equals 1. Let then R denote the larger tuberadius, and define

$$q = \sqrt{[(p^2 - R^2)/(p^2 - 1)]}$$

then the p-z correspondance is given by

$$z = \frac{R}{\pi}\left[\ln\{\frac{1+q}{1-q}\} - \frac{1}{R}\ln\{\frac{R+q}{R-q}\}\right] + R\ i\ \mathrm{sign}\{\mathrm{Re}(p)\}$$

where the last term, the constant of integration, is different in each quadrant in the p-plane due to the fact that the integration has to be carried out from the singularities on the wall, where Im(p) = 0, and due to the fact that the path of integration cannot cross the imaginary axis in the p-plane.

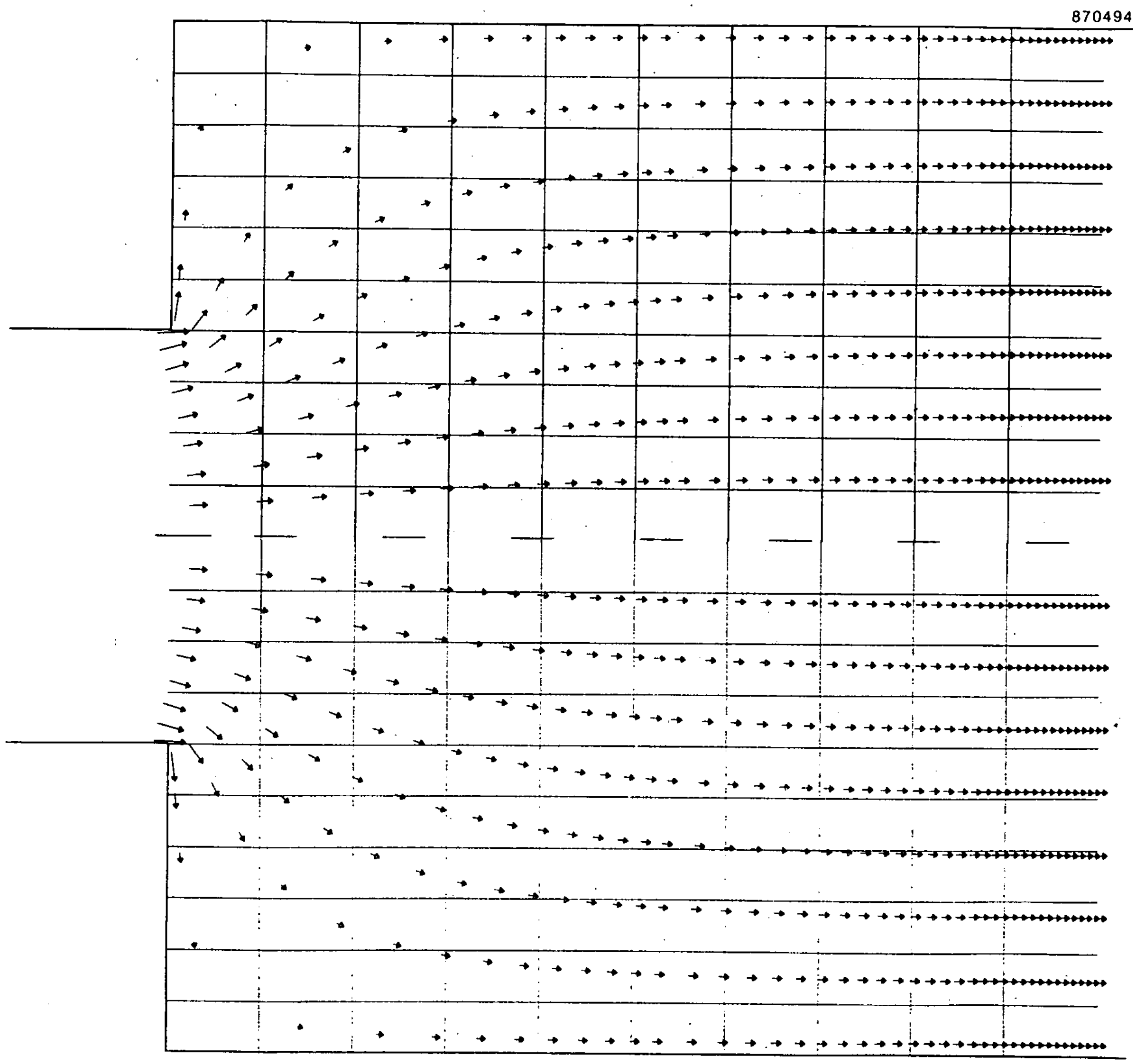


Figure 3.23 Stream lines of irrotational soleniodal flow in a two dimensional channel with sudden expansion.

Figure 3.23 shows the resulting flow lines in the z-plane. To each boundary a flow line is attached. Near the sudden expansion high velocity gradients appear. These gradients are to be dissolved by the initiation of vorticity in this region, leading to vortex structures downstream of the sudden expansion. Due to lack of time these computations have not yet been performed.

3.5 CONCLUSIONS FROM MODELING

For the simulation of time dependent two dimensional flows across solid boundaries a rather simple modeling with discrete vortices was found to be efficient and realistic. Flow phenomena like the von Karman Street can be reproduced.

Away from solid boundaries flow field computations are accurate. Close to boundaries vortices are initiated at so-called 'creation points'. The set-up of these creation points was examined closely, and a new and effective set-up was proposed.

The incorporating of diffusion of individual vortex blobs due to viscosity will make the flow simulation more realistic.
The visualisation of flow development in a sudden expansion was started, but has to be completed.

4 REFERENCES AND LITERATURE

1 Schulte, DFVLR
Private communication

2 Schadow, K.C., K.J. Wilson and M.J. Lee
Enhancement of mixing in ducted rockets with elliptic gasgenerator nozzles
AIAA/SAE/ASME 20th joint propulsion conference, Cincinatti, 1984

3 Wijchers, T.
A method for spectroscopic temperature measurements in a solid fuel combustion chamber
Memorandum M-530, report PML 1985-C7, DUT/PML-TNO, 1985

4 Culick, F.E.C.
Rotation axisymmetric mean flow and damping of acoustic waves in a solid propellant rocket
AIAA J., Vol. 4, page 1462, 1966.

5 Flandro, G.A.
Vortex driving mechanism in oscillatory rocket flows
J. Propulsion, Vol. 2, no. 3, page 206, 1986

6 Clark, W.H. and J.W. Humphrey
Identification of longitudinal acoustic modes associated with pressure oscillations in ramjets
J. of Propulsion, Vol 2, no. 3, 1986

7 Yang, V. and E.C. Culick
Analysis of low frequency combustion instabilities in a laboratory ramjet combustor
Combust. Sci. and Techn., Vol. 45, page 1, 1986

8 Lighthill, J.
Waves in fluids
Cambridge Univ. Press, 1978

9 Chorin, A.J.
Numerical study of slightly viscous flow
J. Fluid Mech., Vol. 57, part 4, page 785, 1973

10 Chorin, A.J.
Vortex sheet approximation of boundary layers
J. Computational Physics, Vol. 27, page 428, 1978

11 Spalart, P.R.
Numerical simulation of separated flows
Ph.D. Thesis, Stanford University, 1982

12 Morino L.
Helmholtz decomposition revisited: Vorticity generation and trailing edge condition. Part 1: incompressible flows
Computational Mechanics, Vol. 1, no. 1, page 65, 1986

13 Lamb, sir H.
Hydrodynamics
Cambridge University Press, 1963 (1879)

14 Roquemore, W.M., R.S. Tankin, H.H. Chiu and S.A. Lottes
A study of bluff-body combustor using laser sheet lighting
Experiments in Fluids, Vol. 4, page 205, 1986

15 Driver, D.M. and H.L. Seegmiller
Features of a reattaching turbulent shear layer in divergent channel flow
AIAA J., Vol. 23, no. 2, page 163, 1985

APPENDIX 1

Derivation of interaction equations for vortices from Biot-Savart law

Evaluation of the Biot-Savart integral :

The contribution to the velocity field at location $\bar{r}$ due to a vortex blob with shape function Υ at location $\bar{r}_i = \bar{r}(i)$ is computed from an integral, F :

$$F \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} dx^1 \; dy^1 \quad (y^1 - y) \; \frac{\Upsilon(|\bar{r}^1 - \bar{r}(i)|)}{|\bar{r} - \bar{r}^1|^2}$$

$$\bar{r} = (x, y)$$

This integral is now solved, firstly for the component in x-direction : Substitute

$$x^1 - x_i = r \cos \varphi \qquad ; \qquad x - x_i = t \cos \psi$$
$$y^1 - y_i = r \sin \varphi \qquad ; \qquad y - y_i = t \sin \psi$$

with φ and ψ choosen such that $r \geq 0$ and $t \geq 0$. This yields

$$|\bar{r} - \bar{r}^1|^2 = (r \cos \varphi - t \cos \psi)^2 + (r \sin \varphi - t \sin \psi)^2 =$$
$$= r^2 + t^2 - 2rt \cos (\varphi - \psi)$$

$$F = \int_0^{2\pi} d\varphi \int_0^{\infty} dr \quad r \; \frac{(r \sin \varphi - t \sin \psi) \; \Upsilon(r)}{[r^2 + t^2 - 2rt \cos(\varphi - \psi)]}$$

where the factor "r" appears as the determinant of the Jacobian of the transformation $(x^1, y^1) \rightarrow (r, \varphi)$

Substitute $\sigma = \varphi - \psi$

$$\int_{-\psi}^{2\sigma-\psi} \sin (\sigma + \psi) \frac{d\sigma}{r^2 + t^2 - 2r\,t \cos (\sigma)} = \left(\int_{-\psi}^{0} + \int_{0}^{2\sigma} - \int_{2\pi-\psi}^{2\pi}\right) (\sin \psi \cos \sigma +$$
$$+ \sin \sigma \cos \psi) \cdot$$

$$\cdot \left[\frac{d\sigma}{r^2 + t^2 - 2rt \cos \sigma}\right] = \int_0^{2\pi} \sin \psi \cos \sigma \quad d\sigma \; \frac{1}{r^2 + t^2 - 2r\sigma \cos \sigma}$$

($\sigma^1 = \sigma - 2\pi$ in the third integral that runs from $2\pi-\psi$ to 2π)

$$F = \lim_{\varepsilon \downarrow 0} \left(\int_0^{t-\varepsilon} + \int_{t+\varepsilon}^{\infty}\right) r\gamma(r) \sin\psi \left(\int_0^{2\pi} \frac{(r\cos\varphi - t)}{r^2 + t^2 - 2rt\cos\varphi}\, d\varphi\right) dr$$

The last integral is reduced as follows.

$$\int_0^{2\pi} d\varphi \frac{r\cos\varphi - t}{r^2 + t^2 - 2rt\cos\varphi} = \int_0^{2\pi} d\varphi \frac{r\cos\varphi - \frac{r^2+t^2}{2t} + \frac{r^2+t^2}{2t} - t}{r^2 + t^2 - 2rt\cos\varphi} =$$

$$= \int_0^{2\pi} d\varphi \left(-\frac{1}{2t} + \frac{r^2 - t^2}{2t\,(r^2 + t^2 - 2rt\cos\varphi)}\right) = -\frac{\pi}{t} + \left(\frac{r^2 - t^2}{4rt^2}\right) \int_0^{2\pi} d\varphi \frac{1}{a - \cos\varphi}$$

with $a = \dfrac{r^2 + t^2}{2rt} > 1$

At the end of this appendix 1 it shall be proved that $\displaystyle\int_0^{2\pi} \frac{d\varphi}{a - \cos\varphi} = \frac{2\pi}{\sqrt{[a^2 - 1]}}$ $(a>1)$

In standard tables of integrals (Gradsteyn and Ryzhik for example) the same result can be found.

The combination of the above results yields

$$F = \lim_{\varepsilon \downarrow 0} \left(\int_0^{t-\varepsilon} + \int_{t+\varepsilon}^{\infty}\right) r\,\gamma(r) \sin(\psi) \left\{-\frac{\pi}{t} + \frac{r^2 - t^2}{4rt^2} \cdot \frac{2\pi\,(2rt)}{\sqrt{[(r^2 + t^2)^2 - 4r^2t^2]}}\right\} dr =$$

$$= \lim_{\varepsilon \downarrow 0} \left(\int_0^{t-\varepsilon} + \int_{t+\varepsilon}^{\infty}\right) r\,\gamma(r) \sin(\psi) \frac{\pi}{t} \left\{-1 + \frac{(r - t)(r + t)}{\sqrt{[(r^2 + t^2 - 2rt)(r^2 + t^2 + 2rt)]}}\right\} dr$$

$=$

$$= \lim_{\varepsilon \downarrow 0} \left(\int_0^{t-\varepsilon} + \int_{t+\varepsilon}^{\infty}\right) r\,\gamma(r) \sin\psi \frac{\pi}{t} \left\{-1 + \mathrm{sign}(r^2 - t^2)\right\} dr =$$

$$= \lim_{\varepsilon \downarrow 0} \int_0^{t-\varepsilon} dr\, r\,\gamma(r) \sin\psi \frac{\pi}{t} (-2) = -\frac{2\pi}{t} \sin(\psi) \int_0^{t} dr\, r\,\gamma(r)$$

$$F = -\frac{2\pi}{t^2}(y - y_i)\int_0^t dr\, r\, \gamma(r) = (y_i - y)\frac{2\pi}{t^2}\int_0^t dr\, r\, \gamma(r)$$

$$\left.\frac{d}{dt}\right|_{t_0} \int_0^t dr\, r\, \gamma(r) = t_0\, \gamma(t_0)$$

So: $F = (y_i - y)\, \eta\,(|\bar{r} - \bar{r}_i|)$

with $\left.\frac{d}{dt}\right|_{t_0} t^2\, \eta(t) = 2\pi\, t_0\, \gamma(t_0)$

Biot-Savart: $\binom{u}{v}(x,y) = \bar{U}_\infty + \frac{1}{2\pi}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\binom{y^1 - y}{x - x^1}\frac{\omega(x^1, y^1)\, dx^1\, dy^1}{(x - x^1)^2 + (y - y^1)^2}$

$$\omega(\bar{r}) = \sum_{i=1}^{N_{vort}} \Gamma_i\, \gamma(|\bar{r} - \bar{r}_i|)$$

Hence, summarizing :

$$\underline{u\,(x,y) = U_{\infty_x} + \frac{1}{2\pi}\sum_{i=i}^{N_{vort}} \Gamma_i (y_i - y)\, \eta(|\bar{r} - \bar{r}_i|)}$$

$$\underline{\text{with } \left.\frac{d}{dt}\right|_{t_0} t^2\, \eta(t) = 2\pi\, t_0\, \gamma(t_0)}$$

v(x,y) can be evaluated in a strictly analoguous manner.

Analytical solution of $\int_0^{2\pi} \frac{d\varphi}{a - \cos\varphi}$ **:**

The calculation of $G \overset{def}{=} \int_0^{2\pi} \frac{d\varphi}{a - \cos\varphi}$ with a > 1 goes as follows.

Let Γ denote the set of complex numbers with magnitude 1

$$G = \int_0^{2\pi} \frac{d\varphi}{a - \frac{1}{2}(e^{i\varphi} + e^{-i\varphi})} = \int_0^{2\pi} d\varphi \frac{2e^{i\varphi}}{2a\, e^{i\varphi} - 1 - e^{2i\varphi}} = \frac{1}{i} \int_\Gamma dz \frac{2}{2az - z^2 - 1}$$

The second equality follows from the observation that the parametrization of Γ given by

$f: \varphi \to e^{i\varphi} \in \Gamma$

has the derivative $i\, e^{i\varphi}$

$$\frac{1}{2\pi i} \int_\Gamma g(z)dz = \Sigma \text{ (circulation number). residue (g, within area bounded by } \Gamma)$$

$2az - z^2 - 1 = -(z^2 - 2az + 1) = -\{z-(a + \sqrt{[a^2 - 1]})\}\cdot\{z - (a - \sqrt{[a^2 - 1]})\}$
Since a > 1, only the residue at $z = a - \sqrt{[a^2 - 1]}$ lies within the area bounded by Γ. This gives the result

$$G = 2\pi\left(\frac{1}{2\pi i} \int_\Gamma dz \frac{2}{2az - z^2 - 1}\right) = 2\pi\left(\frac{-2}{a - \sqrt{[a^2 - 1]} - a - \sqrt{[a^2 - 1]}}\right) = \frac{2\pi}{\sqrt{[a^2 - 1]}}$$

APPENDIX 2

Source listing of matrix solvers in Pascal

Real version; Gaussian elimination with rows

```
%PAGE;
(*==============================================================*)
(*                                                              *)
(*                         MATSOLVE                             *)
(*                                                              *)
(*==============================================================*)
PROCEDURE MATSOLVE (VAR A    : RA500500;
                    VAR IPAR : INTEGER;
                    VAR IFL  : INTEGER;
                    VAR ISUB : IA1100;
                    VAR XX   : RA500;
                        B    : RA01100;
                        IDIM : INTEGER);
VAR I, IDM1, IH, INN, J, K        : INTEGER;
    ABSA, AM, HULP,G,PIVOTR       : REAL;
(**************************************************************)
(*                                                            *)
(* This procedure solves the matrix equation                  *)
(*                         A.x = b                            *)
(* by means of a Gaussian elimination process with partial    *)
(* pivoting.                                                  *)
(*                                                            *)
(*                       REAL VERSION                         *)
(*                                                            *)
(* INPUT :                                                    *)
(*                                                            *)
(* A      Matrix                                              *)
(* B      Known vector                                        *)
(* IDIM   Dimension of the matrix                             *)
(* IPAR   Parameter which controls the elimination process, if *)
(*        IPAR = 1 the Gaussian elimination process has to be *)
(*        performed on A; if IPAR = 2 only vector B has to be *)
(*        modified to account for the elimination process. This *)
(*        means that the input matrix A has already been      *)
(*        Gaussian eliminated in the way corresponding to the *)
(*        modification of B. This can be useful if this routine *)
(*        is used in an iteration cycle.                      *)
```

```
(*                                                                  *)
(* OUTPUT :                                                         *)
(*                                                                  *)
(* IFL    Status of the Gaussian elimination process:               *)
(*            0 : Matrix solving is completed successfully          *)
(*            1 : Error occurred during Gaussian elimination        *)
(*                process                                           *)
(* XX     solution vector (only if IPAR = 2)                        *)
(*                                                                  *)
(* VARIABLES USED :                                                 *)
(*                                                                  *)
(* IDM1   IDIM - 1                                                  *)
(* ISUB   contains the row numbers of the partially pivoted         *)
(*        matrix                                                    *)
(*                                                                  *)
(********************************************************************)
begin
  IFL := 0;
  IDM1 := IDIM - 1;
(********************************************************************)
(*                                                                  *)
(* Perform the Gaussian elimination process on matrix A             *)
(*        j                                                         *)
(* i   A(11)  A(12)  A(13) ......                                   *)
(*     A(21)  A(22)  A(23) ......                                   *)
(*                                                                  *)
(********************************************************************)
  IF IPAR = 1 THEN
(********************************************************************)
(* Initialize array ISUB                                            *)
(********************************************************************)
    BEGIN
    FOR I := 1 TO IDIM DO      ISUB(.I.) := I;
(********************************************************************)
(* Calculate maximum value of A(I,J) at each column. The row        *)
(* number of the largest value of A(I,J) is stored in ISUB(J).      *)
(********************************************************************)
    FOR J := 1 TO IDM1 DO
      BEGIN
      AM := 0.0;
      FOR I := J TO IDIM DO
        BEGIN
        ABSA := ABS(A(.ISUB(.I.),J.));
        IF AM < ABSA THEN
          BEGIN
          AM := ABSA;
          INN := I;
          END;
        END;  (*  end of for I iteration  *)
(********************************************************************)
(* If the maximum value of A(I,J) equals zero, it is not            *)
(* possible to calculate in inverse of the matrix A                 *)
(********************************************************************)
      IF AM < 1.0E-10 THEN
        BEGIN
```

```
          WRITELN;
          IFL := 1;
          WRITE ('AM= ', AM, 'INN = ', INN);
          WRITELN ('J = ', J);
          WRITE ('STOP, THIS RUN OF THE PROGRAMME "VORTEX"');
          WRITELN (' IS IMPOSSIBLE DUE TO FOUT IN MATSOLVE');
          END;  (*  end of if AM<  situation  *)
        IF AM < 1.0E-30 THEN
          BEGIN
          WRITE ('STOP, THIS RUN OF THE PROGRAMME "VORTEX"');
          WRITELN (' IS IMPOSSIBLE DUE TO SEVERE FOUT IN MATSOLVE');
          HALT;
          END;  (*  end of if AM<  situation  *)
(*****************************************************************)
(* Het J-de veld van ISUB gaat het rijnummer bevatten waar de    *)
(* maximale waarde van de J-de kolom staat.                      *)
(*****************************************************************)
        IH := ISUB(.J.);
        ISUB(.J.) := ISUB(.INN.);
        ISUB(.INN.) := IH;
        PIVOTR := -1.0 / A(.ISUB(.J.),J.);
(*****************************************************************)
(* Set the column elements with absolute values less than AM     *)
(* to zero by subtracting the row with index J from it. The      *)
(* multiplication factor used in this subtraction is stored in   *)
(* place of the zero, i.e. in element A(ISUB(I),J)               *)
(*****************************************************************)
        FOR I := J+1 TO IDIM DO
            BEGIN
            A(.ISUB(.I.),J.) := PIVOTR * A(.ISUB(.I.),J.);
            FOR K := J+1 TO IDIM DO
                BEGIN
                G := A(.ISUB(.I.),J.) * A(.ISUB(.J.),K.);
                A(.ISUB(.I.),K.) := G + A(.ISUB(.I.),K.);
                END;
            END;  (*  end of for I iteration  *)
        END;   (*  end of for J iteration  *)
    END;   (*  end of IPAR = 1 situation  *)

IF IPAR = 2 THEN
  BEGIN
(*****************************************************************)
(* Solve the vector XX : first, set values of B less than 1.0E-30*)
(* to zero, then modify the vector B by accounting for the       *)
(* Gaussian elimination process                                  *)
(*                                                               *)
(*   (( D ))  =  (( A op Jordanvorm ))                           *)
(*                                                               *)
(*   (( A ))  =  (( J )) (( D ))                                 *)
(*                                                               *)
(*   (( A )) (XX)  =  (B)        is the same as                  *)
(*                                                               *)
(*                                 -1                            *)
(*   (( D )) (XX)  =  (( J ))  (B)                               *)
(*                                                               *)
```

```
(*   In A(ISUB(I),J) the multiplication factor of the rows was    *)
(*   stored in the appropriate way.                                *)
(*****************************************************************)
    FOR I := 1 TO IDIM DO
      BEGIN
      HULP :=MAX(ABS(B(.I.)) - 1.0E-60, 0.0);
      IF HULP <> 0 THEN    HULP := HULP/ABS(HULP);
      B(.I.) := HULP * B(.I.);
      END;  (*  end of for I iteration  *)
    FOR J := 1 TO IDM1 DO
      BEGIN
      XX(.J.) := B(.ISUB(.J.).);
      B(.ISUB(.J.).) := B(.J.);
      FOR I := J+1 TO IDIM DO
          BEGIN
          G := A(.ISUB(.I.),J.) *  XX(.J.);
          B(.ISUB(.I.).) := G + B(.ISUB(.I.).);
          END;
      END;  (*  end of for J iteration  *)
(*****************************************************************)
(* Solve the equation by back substitution, starting with the    *)
(* last row of the Jordan form of the matrix (A).                *)
(*****************************************************************)
    XX(.IDIM.) := B(.ISUB(.IDIM.).) / A(.ISUB(.IDIM.),IDIM.);
    FOR J := IDM1 DOWNTO 1 DO
        BEGIN
        FOR I := J+1 TO IDIM DO
            BEGIN
            G := A(.ISUB(.J.),I.) *  XX(.I.);
            XX(.J.) := XX(.J.) - G;
            END;
        XX(.J.) := XX(.J.) / A(.ISUB(.J.),J.) ;
        HULP := MAX(ABS(XX(.J.)) - 1.0E-60, 0.0);
        IF HULP<>0 THEN           HULP := HULP / ABS(HULP);
        XX(.J.) := HULP * XX(.J.);
        END;
   END; (* end of IPAR=2 situation  *)
END;  (* end of routine MATSOLVE  *)
```

Complex version of matrix inverting and equation solving subroutines

```
(*****************************************************************)
(*                                                               *)
(*   FUNCTIONS AND PROCEDURES FOR COMPUTING IN COMPLEX NUMBERS   *)
(*                                                               *)
(*****************************************************************)

FUNCTION CABS (A : COMPLEX) : REAL;
(*****************************************************************)
(*  Deze functie berekent de modulus van een complex getal       *)
(*****************************************************************)
BEGIN
  CABS := SQRT(A.RE*A.RE + A.IM*A.IM);
END;

FUNCTION CMPLX (HULP,PLOP : REAL) : COMPLEX;
(*****************************************************************)
(*  CMPLX is een functie die van twee reeele getallen a en b een *)
(*  complex getal maakt :  a + bi                                *)
(*****************************************************************)
BEGIN
  CMPLX.RE := HULP;
  CMPLX.IM := PLOP;
END;

FUNCTION CONJG (A : COMPLEX) : COMPLEX;
(*****************************************************************)
(*  CONJG berekent de geconjugeerde van een complex getal        *)
(*****************************************************************)
BEGIN
  CONJG.RE := A.RE;
  CONJG.IM := - A.IM;
END;

PROCEDURE CADD (A,B : COMPLEX; VAR ZZ : COMPLEX);
(*****************************************************************)
(*  CADD maakt het mogelijk twee complexe getallen bij elkaar op *)
(*  te tellen. Vb.  CADD (C1,C2,C3) betekent : C3 = C1 + C2      *)
(*****************************************************************)
BEGIN
  ZZ.RE := A.RE + B.RE;
  ZZ.IM := A.IM + B.IM;
END;

PROCEDURE CSUB (A,B : COMPLEX; VAR ZZ : COMPLEX);
(*****************************************************************)
(*  CSUB maakt het mogelijk twee complexe getallen van elkaar af *)
(*  te trekken. Vb.  CSUB (C1,C2,C3) betekent : C3 = C1 - C2     *)
(*****************************************************************)
BEGIN
```

```
  ZZ.RE := A.RE - B.RE;
  ZZ.IM := A.IM - B.IM;
END;

PROCEDURE CMUL (A,B : COMPLEX; VAR ZZ : COMPLEX);
(*******************************************************************)
(*  CMUL maakt het mogelijk twee complexe getallen met elkaar te *)
(*  vermenigvuldigen.                                            *)
(*        Vb. CMUL (C1,C2,C3) betekent : C3 = C1 * C2            *)
(*******************************************************************)
BEGIN
  ZZ.RE := A.RE * B.RE - A.IM * B.IM;
  ZZ.IM := A.RE * B.IM + A.IM * B.RE;
END;

PROCEDURE CMULR (A : COMPLEX; B : REAL; VAR ZZ : COMPLEX);
(*******************************************************************)
(*  Bij de aanroep van CMULR wordt een complex getal met een     *)
(*  reeel getal vermenigvuldigd.                                 *)
(*        Vb. CMULR (C1,R,C2) betekent : C2 = C1 * R             *)
(*******************************************************************)
BEGIN
  ZZ.RE := A.RE * B;
  ZZ.IM := A.IM * B;
END;

PROCEDURE CMULI (A : COMPLEX; B : INTEGER; VAR ZZ : COMPLEX);
(*******************************************************************)
(*  Bij de aanroep van CMULI wordt een complex getal met een     *)
(*  Integer vermenigvuldigd.                                     *)
(*        Vb. CMULI (C1,I,C2) betekent : C2 = C1 * I             *)
(*******************************************************************)
BEGIN
  ZZ.RE := A.RE * B;
  ZZ.IM := A.IM * B;
END;

PROCEDURE CINV (A : COMPLEX; VAR ZZ : COMPLEX);
(*******************************************************************)
(*  CINV zorgt ervoor dat de inverse van een complex getal wordt *)
(*  berekend. Vb. CINV(C1,C2) betekent : C2 = 1 / C1             *)
(*******************************************************************)
VAR P : REAL;
BEGIN
  P := SQR(A.RE) + SQR(A.IM);
  ZZ.RE := A.RE / P;
  ZZ.IM := -A.IM / P;
END;

PROCEDURE CDIV (A,B : COMPLEX; VAR ZZ : COMPLEX);
(*******************************************************************)
(*  Procedure CDIV deelt een complex getal met een ander complex *)
(*  getal. Vb. CDIV (C1,C2,C3) betekent : C3 = C1 / C2           *)
(*******************************************************************)
BEGIN
```

```
  CINV(B,ZZ);
  CMUL(A,ZZ,ZZ);
END;

PROCEDURE CDIVR (A : COMPLEX; B : REAL; VAR ZZ : COMPLEX);
(*****************************************************************)
(*  Procedure CDIVR deelt een complex getal door een reeel getal *)
(*        Vb. CDIVR (C1,R,C2) betekent : C2 = C1 / R             *)
(*****************************************************************)
BEGIN
  ZZ.RE := A.RE / B;
  ZZ.IM := A.IM / B;
END;

PROCEDURE CDIVI (A : COMPLEX; B : INTEGER; VAR ZZ : COMPLEX);
(*****************************************************************)
(*  Procedure CDIVI deelt een complex getal door een integer.    *)
(*        Vb. CDIVI (C1,I,C2) betekent : C2 = C1 / I             *)
(*****************************************************************)
BEGIN
  ZZ.RE := A.RE / B;
  ZZ.IM := A.IM / B;
END;

PROCEDURE CSQRT (A : COMPLEX; VAR ZZ : COMPLEX);
(*****************************************************************)
(*  CSQRT berekent de wortel van een complex getal.              *)
(*        Vb. CSQRT (C1,C2) betekent : C2 = wortel (C1)          *)
(*****************************************************************)
VAR P : REAL;
BEGIN
  IF (A.RE = 0) AND (A.IM = 0) THEN ZZ := A
  ELSE BEGIN
    P := SQRT ((ABS(A.RE) + CABS(A))/2);
    IF (A.RE < 0) AND (A.IM < 0) THEN P := - P;
    IF A.RE <0 THEN BEGIN
        ZZ.IM := P;
        ZZ.RE := A.IM / (2*P);
        END
    ELSE
        BEGIN
        ZZ.RE := P;
        ZZ.IM := A.IM / (2*P);
        END;
  END;  (*  end of else begin  *)
END;  (* end of procedure   *)

PROCEDURE CEXP (A : COMPLEX; VAR ZZ : COMPLEX);
(*****************************************************************)
(*  CEXP berekent de e-macht van een complex getal               *)
(*        Vb. CEXP (C1,C2) betekent : C2 = e ** (C1)             *)
(*****************************************************************)
VAR P : REAL;
BEGIN
  P := EXP(A.RE);
```

```
    ZZ.RE := P * COS(A.IM);
    ZZ.IM := P * SIN(A.IM);
  END;

  PROCEDURE CLN (A : COMPLEX; VAR ZZ : COMPLEX);
  (*****************************************************************)
  (*  CLN berekent de natuurlijke logaritme van een complex getal  *)
  (*        Vb. CLN (C1,C2) betekent : C2 = Ln(C1)                 *)
  (*  In het programma VORTEX wordt CLOG i.p.v. CLN gebruikt!!     *)
  (*****************************************************************)
  BEGIN
    ZZ.RE := LN(CABS(A));
    IF ABS(A.IM) < ABS(A.RE) THEN ZZ.IM := ARCTAN( ABS(A.IM / A.RE) )
    ELSE ZZ.IM := PI/2 - ARCTAN( ABS(A.RE / A.IM) );
    IF A.RE < 0 THEN ZZ.IM := PI - ZZ.IM;
    IF A.IM < 0 THEN ZZ.IM := -ZZ.IM;
  END;

  PROCEDURE CLOG (A : COMPLEX; VAR ZZ : COMPLEX);
  (*****************************************************************)
  (*  Het veld PI ( = 3.14159 ) dient globaal geinitialiseerd te   *)
  (*  zijn.                                                         *)
  (*  CLOG rekent exact hetzelfde uit als CLN maar is iets korter  *)
  (*  Waarschijnlijk zal CLOG daarom minder rekentijd vergen dan   *)
  (*  CLN. In het programma VORTEX wordt alleen CLOG aangeroepen.  *)
  (*****************************************************************)
  VAR P : REAL;
  BEGIN
    P := CABS(A);
    ZZ.RE := LN(P);
    ZZ.IM := 2 *  ARCTAN( A.IM / (P + A.RE) )  ;
    IF (A.RE<0.0) AND (A.IM = 0.0) THEN BEGIN
       WRITELN ('CLOG(-1000 OF ZO) + 0,0 i = ');
  (* het volgende is a matter of choise, maar komt in GEOMETRY goed uit *)
       ZZ.IM := PI;
       WRITE(ZZ.RE, ZZ.IM);
       END;   (*  end of log(negatief getal) - situatie  *)
  END;   (*  end of routine CLOG   *)

  %PAGE;
  (*===============================================================*)
  (*                                                               *)
  (*                          CSIGN                                *)
  (*                                                               *)
  (*===============================================================*)
  FUNCTION CSIGN (A : REAL; B : COMPLEX) : COMPLEX;
  (*****************************************************************)
  (* CSIGN (R,C) ZET HET TEKEN VAN R VOOR C                         *)
  (*        VB. CSIGN (-6.2, 5-I) = -5+I                            *)
  (*****************************************************************)
  VAR H  : INTEGER;
      G  : COMPLEX;
  BEGIN
    IF A = 0 THEN H := 0
    ELSE H := ROUND(A / ABS(A));
```

```
  CMULI (B, H, G);
  CSIGN := G;
END;

%PAGE;
(*===============================================================*)
(*                                                               *)
(*                       CMATINV                                 *)
(*                                                               *)
(*===============================================================*)
PROCEDURE CMATINV(VAR A    : CAIDIM;
                      IDIM : INTEGER;
                  VAR AINV : CAIDIM);
(***************************************************************)
(* In the procedure header the following fields should occur :  *)
(*      VAR AINV : CAIDIM      IDIM : INTEGER   and, optionally *)
(*      matrix VAR A : CAIDIM , that shall be rewriten          *)
(***************************************************************)
VAR I,IC,IDM1,IH,INN,J,K : INTEGER;
    ABSA,AM              : REAL;
    G, PIVOTR            : COMPLEX;
    ISUB                 : IA500;
    E                    : CA500;
(***************************************************************)
(*                                                              *)
(* This procedure has been developed to calculate the inverse   *)
(* of a matrix by means of a Gaussian elimination process with  *)
(* partial pivoting                                             *)
(*                      COMPLEX VERSION                         *)
(*                                                              *)
(* INPUT :                                                      *)
(*                                                              *)
(* A      Matrix to be inverted                                 *)
(*        Note : The original form of this matrix is destroyed  *)
(*               in the course of performing "MATINV".          *)
(* IDIM   Dimension of the matrix                               *)
(*                                                              *)
(* OUTPUT :                                                     *)
(*                                                              *)
(* AINV   Inverted matrix                                       *)
(* IFL    Status of the inversion :                             *)
(*          0 : Matrix inversion is completed successfully      *)
(*          1 : Error occurred during matrix inversion          *)
(*                                                              *)
(* VARIABLES USED :                                             *)
(*                                                              *)
(* E      Unity vector                                          *)
(* IDM1   IDIM - 1                                              *)
(* ISUB   contains the row numbers of the partially pivoted     *)
(*        matrix                                                *)
(***************************************************************)
BEGIN
  IFL := 0;
  IDM1 := IDIM - 1;
(***************************************************************)
```

```
(* Initialize array ISUB                                              *)
(*    /   \                                                           *)
(*   |  1  |                                                          *)
(*   |  2  |                                                          *)
(*   |  .  |    =   ISUB                                              *)
(*   |  .  |                                                          *)
(*   |  N  |                                                          *)
(*    \   /                                                           *)
(*                                                                    *)
(*  A(1,1)  ... ... ... ... ... ... ... ...   A(1,n)                  *)
(*    .       .   .   .   .   .   .   .   .      .                    *)
(*    .       .   .   .   .   .   .   .   .      .                    *)
(*    .       .   .   .   .   .   .   .   .      .                    *)
(*  A(n,1)  ... ... ... ... ... ... ... ...   A(n,n)                  *)
(*                                                                    *)
(*  n = IDIM                                                          *)
(*********************************************************************)
  FOR I := 1 TO IDIM DO    ISUB(.I.) := I;
(*********************************************************************)
(* Calculate maximum value of A(I,J) at each column.                  *)
(* The row number of the largest value of A(I,J) is stores in         *)
(* ISUB(J)                                                            *)
(*********************************************************************)
  FOR J := 1 TO IDM1 DO
    BEGIN
    AM := 0.0;
    FOR I := J TO IDIM DO
      BEGIN
      ABSA := CABS(A(.ISUB(.I.),J.));
      IF AM < ABSA THEN
        BEGIN
        AM := ABSA;
        INN := I;
        END;
      END;  (* end of I iteration  *)
(*********************************************************************)
(* If the maximum value of A(I,J) equals zero, it is not pos-         *)
(* sible to calculate in inverse of the matrix A                      *)
(*********************************************************************)
    IF AM < 1.0E-10 THEN
      BEGIN
      IFL := 1;
      WRITELN ('Stop this run of the program "VORTEX", matrix ');
      WRITE ('inversion is impossible');
      HALT;
      END;  (*  end of AM< condition  *)
(*********************************************************************)
(*  Stop in het J-de veld van ISUB het rijnummer waar de maxi-        *)
(*  male waarde van de J-de kolom staat                               *)
(*********************************************************************)
    IH := ISUB(.J.);
    ISUB(.J.) := ISUB(.INN.);
    ISUB(.INN.) := IH;
    CINV (A(.ISUB(.J.),J.), PIVOTR);
    CMULI (PIVOTR,-1,PIVOTR);
```

```
(*******************************************************************)
(* Set the column elements with absolute values less than AM to  *)
(* zero by subtracting the row with index J from it. The multi-  *)
(* plication factor used in this subtraction is stored in place  *)
(* of the zero, i.e. in element A(ISUB(I),J)                      *)
(*******************************************************************)
    FOR I := J+1 TO IDIM DO
      BEGIN
      CMUL (A(.ISUB(.I.),J.), PIVOTR, A(.ISUB(.I.),J.) );
      FOR K := J+1 TO IDIM DO
        BEGIN
        CMUL (A(.ISUB(.I.),J.), A(.ISUB(.J.),K.), G);
        CADD (G, A(.ISUB(.I.),K.), A(.ISUB(.I.), K.) );
        END;  (*  end of K iteration  *)
      END;  (* end of I iteration  *)
  END;  (*  end of J iteration  *)
(*******************************************************************)
(* Calculate the inverted matrix by solving the equation          *)
(*                           AX = E                                *)
(* where E is the unity vector, and the solution vector is        *)
(* the IC-th column of the inverted matrix                         *)
(*******************************************************************)
  FOR IC := 1 TO IDIM DO
(*******************************************************************)
(* IC : the column of the inverted matrix being determined        *)
(*******************************************************************)
    BEGIN
    FOR I:= 1 TO IDIM DO   E(.I.) := CMPLX(0.0,0.0);
    E(.IC.) := CMPLX(1.0,0.0);
(*******************************************************************)
(* Modify the vector E by accounting for the Gaussian elimina-    *)
(* tion process.                                                    *)
(*                                                                  *)
(*   (( D ))  =  (( A op Jordanvorm ))                             *)
(*                                                                  *)
(*   (( A ))  =  (( J )) (( D ))                                   *)
(*                                                                  *)
(*   (( A )) (X)  =  (E)        is the same as                     *)
(*                                                                  *)
(*                          -1                                      *)
(*   (( D )) (X)  =  (( J ))  (E)                                  *)
(*                                                                  *)
(*   In A(ISUB(I),J) the multiplication factor of the rows was    *)
(*   stored in the appropriate way.                                *)
(*******************************************************************)
    FOR J := 1 TO IDM1 DO
      BEGIN
      AINV(.J,IC.) := E(.ISUB(.J.).);
      E(.ISUB(.J.).) := E(.J.);
      FOR I := J+1 TO IDIM DO
        BEGIN
        CMUL (A(.ISUB(.I.),J.), AINV(.J,IC.),G);
        CADD(G, E(.ISUB(.I.).), E(.ISUB(.I.).));
        END;  (* end of I iteration  *)
      END;  (*  end of J iteration  *)
```

```
(*****************************************************************)
(*  Solve the equation by back substitution, starting with the   *)
(*  last row of the Jordan-form of the matrix (A).               *)
(*****************************************************************)
   CDIV (E(.ISUB(.IDIM.).), A(.ISUB(.IDIM.),IDIM.),
          AINV(.IDIM,IC.));
   FOR J := IDM1 DOWNTO 1 DO
     BEGIN
     FOR I := J+1 TO IDIM DO
       BEGIN
       CMUL (A(.ISUB(.J.),I.), AINV(.I,IC.),G);
       CSUB (AINV(.J,IC.),G,AINV(.J,IC.));
       END;
     CDIV (AINV(.J,IC.), A(.ISUB(.J.),J.),AINV(.J,IC.));
     END;  (* end of J iteration  *)
   END;  (*  end of IC iteration  *)
END;  (*  end of subroutine CMATINV  *)

% PAGE
(*===============================================================*)
(*                                                               *)
(*                        CMATSOLVE                              *)
(*                                                               *)
(*===============================================================*)
PROCEDURE CMATSOLVE(VAR A    : CAIDIM;
                    VAR IPAR : INTEGER;
                    VAR X    : CA1000;
                    VAR ISUB : IA500;
                        B    : RA1000);
(*****************************************************************)
(* In the procedure header the following fields should appear :  *)
(*    VAR A : CAIDIM             VAR IPAR : INTEGER              *)
(*    VAR X : CA2000             VAR ISUB : IA500                *)
(* If not, they should be added.                                 *)
(* In the header of CMATSOLVE declaration VAR B should not ap-   *)
(* pear in the header. If it does, it should be eliminated.      *)
(*****************************************************************)
VAR I, IDM1, IH, INN, J, K : INTEGER;
   ABSA, AM                : REAL;
   G, PIVOTR               : COMPLEX;
(*****************************************************************)
(*                                                               *)
(* This procedure is developed to solve the matrix equation      *)
(*                       A.x = b                                 *)
(* by means of a Gaussian elimination process with partial       *)
(* pivoting.                                                     *)
(*                                                               *)
(*                    COMPLEX VERSION                            *)
(*                                                               *)
(* INPUT :                                                       *)
(*                                                               *)
(* A      Matrix                                                 *)
(* B      Known vector                                           *)
(* IDIM   Dimension of the matrix                                *)
(* IPAR   Parameter which controls the elimination process, if   *)
```

```
(*        IPAR = 1 the Gaussian elimination process has to be   *)
(*        performed on A; if IPAR = 2 only vector B has to be   *)
(*        modified to account for the elimination process. This *)
(*        means that the input matrix A has already been        *)
(*        Gaussian eliminated in the way corresponding to the   *)
(*        modification of B. This can be useful if this routine *)
(*        is used in an iteration cycle.                        *)
(*                                                              *)
(* OUTPUT :                                                     *)
(*                                                              *)
(* X      Solution vector                                       *)
(*                                                              *)
(* VARIABLES USED :                                             *)
(*                                                              *)
(* IDM1   IDIM - 1                                              *)
(* ISUB   contains the row numbers of the partially pivoted     *)
(*        matrix                                                *)
(*                                                              *)
(****************************************************************)
BEGIN
  IDM1 := IDIM - 1;
(****************************************************************)
(* Perform the Gaussian elimination process on matrix A         *)
(****************************************************************)
  IF IPAR = 1 THEN
(****************************************************************)
(* Initialize array ISUB                                        *)
(****************************************************************)
    BEGIN
    FOR I := 1 TO IDIM DO  ISUB(.I.) := I;
(****************************************************************)
(* Calculate maximum value of A(I,J) at each column. The row    *)
(* number of the largest value of A(I,J) is stores in ISUB(J).  *)
(****************************************************************)
    FOR J := 1 TO IDM1 DO
      BEGIN
      AM := 0.0;
      FOR I := J TO IDIM DO
        BEGIN
        ABSA := CABS(A(.ISUB(.I.),J.));
        IF AM < ABSA THEN
          BEGIN
          AM := ABSA;
          INN := I;
          END;  (*  end of AM< condition  *)
        END;  (*  end of I iteration  *)
(****************************************************************)
(* If the maximum value of A(I,J) equals zero, it is not        *)
(* possible to calculate in inverse of the matrix A             *)
(****************************************************************)
      IF AM < 1.0E-10 THEN
        BEGIN
        WRITE ('STOP THIS RUN OF THE PROGRAM "VORTEX", CMATSOLVE');
        WRITE ('is impossible.');
        HALT;
```

```
      END;  (*  end of AM< condition  *)
(*****************************************************************)
(* ISUB(J) is going to contain the number of the row where the   *)
(* maximum value of the J th. column occurs                      *)
(*****************************************************************)
     IH := ISUB(.J.);
     ISUB(.J.) := ISUB(.INN.);
     ISUB(.INN.) := IH;
     CINV (A(.ISUB(.J.),J.), PIVOTR);
     CMULI (PIVOTR, -1, PIVOTR);
(*****************************************************************)
(* Set the column elements with absolute values less than AM     *)
(* to zero by subtracting the row with index J from it. The      *)
(* multiplication factor used in this subtraction is stored in   *)
(* place of the zero, i.e. in element A(ISUB(I),J)               *)
(*****************************************************************)
     FOR I := J+1 TO IDIM DO
       BEGIN
       CMUL (A(.ISUB(.I.),J.), PIVOTR, A(.ISUB(.I.),J.));
       FOR K := J+1 TO IDIM DO
         BEGIN
         CMUL (A(.ISUB(.I.),J.), A(.ISUB(.J.),K.), G);
         CADD (A(.ISUB(.I.),K.), G, A(.ISUB(.I.),K.));
         END;  (*  end of K iteration  *)
       END;  (*  end of I iteration  *)
     END;  (* end of J iteration  *)
(*****************************************************************)
(* End of IPAR=1 situation                                       *)
(*****************************************************************)
  END;  (*  end of IPAR condition  *)

  IF IPAR = 2 THEN
    BEGIN
(*****************************************************************)
(* Solve the vector X : First, set values of B less than 1.0E-30 *)
(* to zero, then modify the vector B by accounting for the       *)
(* Gaussian elimination process                                  *)
(*                                                               *)
(*   (( D ))  =  (( A op Jordanvorm ))                           *)
(*                                                               *)
(*   (( A ))  =  (( J )) (( D ))                                 *)
(*                                                               *)
(*   (( A )) (X)  =  (B)    which is the same as                 *)
(*                                                               *)
(*                              -1                               *)
(*   (( D )) (X)  =  (( J ))  (B)                                *)
(*                                                               *)
(*   In A(ISUB(I),J) the multiplication factor of the rows was   *)
(*   stored in the appropriate way.                              *)
(*****************************************************************)
    FOR I := 1 TO IDIM DO
      B(.I.) := CSIGN(MAX(CABS(B(.I.)) - 1.0E-60, 0.0), B(.I.));
    FOR J := 1 TO IDM1 DO
      BEGIN
      X(.J.) := B(.ISUB(.J.).);
```

```
      B(.ISUB(.J.).) := B(.J.);
      FOR I := J+1 TO IDIM DO
        BEGIN
        CMUL (A(.ISUB(.I.),J.), X(.J.), G);
        CADD(B(.ISUB(.I.).), G, B(.ISUB(.I.).));
        END;  (*  end of I iteration  *)
      END;  (*  end of J iteration  *)
(******************************************************************)
(* Solve the equation by back substitution, starting with the    *)
(* last row of the Jordan form of the matrix (A).                *)
(******************************************************************)
    CDIV (B(.ISUB(.IDIM.).), A(.ISUB(.IDIM.),IDIM.), X(.IDIM.));
    FOR J := IDM1 DOWNTO 1 DO
      BEGIN
      FOR I := J+1 TO IDIM DO
        BEGIN
        CMUL (A(.ISUB(.J.),I.), X(.I.), G);
        CSUB (X(.J.), G, X(.J.));
        END;  (*  end of I iteration  *)
      CDIV (X(.J.), A(.ISUB(.J.),J.), X(.J.));
      X(.J.) := CSIGN(MAX(CABS(X(.J.)) - 1.0E-60, 0.0), X(.J.));
      END;  (*  end of J iteration  *)
    END;  (*  end of IPAR=2 condition  *)
END;  (*  end of subroutine CMATSOLVE  *)
```

Real version; Gaussian elimination with columns

```
%PAGE;
(*===============================================================*)
(*                                                               *)
(*                         TAMSOLF                               *)
(*                                                               *)
(*===============================================================*)
PROCEDURE TAMSOLF  (    A    : RA500500;
                    VAR IPAR : INTEGER;
                    VAR IFL  : INTEGER;
                    VAR ISUB : IA1100;
                    VAR XX   : RA500;
                        B    : RA01100;
                        IDIM : INTEGER);

VAR I, IDM1, IH, INN, J, K        : INTEGER;
    ABSA, AM, HULP,G,PIVOTR       : REAL;
(*****************************************************************)
(*                                                               *)
(* This procedure solves the matrix equation                     *)
(*                      A.x = b                                  *)
(* by means of a Gaussian elimination process with partial       *)
(* pivoting.                                                     *)
(*                                                               *)
(*                    REAL VERSION                               *)
(*                                                               *)
(* Rows will be cleaned to zeroes, one row by one.               *)
(* Multiplication factors are stored instead of these zeroes.    *)
(*                                                               *)
(*****************************************************************)
begin
  IFL := 0;
  IDM1 := IDIM - 1;
  IF IPAR = 1 THEN
    BEGIN
    FOR I := 1 TO IDIM DO      ISUB(.I.) := I;
(*****************************************************************)
(* Calculate maximum value of A(I,J) at each row. The column     *)
(* number of the largest value of A(i,j) is stored in ISUB(j).   *)
(*        j                                                      *)
(* i   A(11)  A(12)  A(13) ......                                *)
(*     A(21)  A(22)  A(23) ......                                *)
(*                                                               *)
(*****************************************************************)
    FOR J := 1 TO IDM1 DO
      BEGIN
      AM := 0.0;
      FOR I := J TO IDIM DO
        BEGIN
        ABSA := ABS( A(.J,ISUB(.I.).) );
```

```
       IF AM < ABSA THEN
         BEGIN
         AM := ABSA;
         INN := I;
         END;
       END;  (*  end of for I iteration  *)
(*****************************************************************)
(* If the maximum value of A(I,J) equals zero, it is not         *)
(* possible to calculate in inverse of the matrix A              *)
(*****************************************************************)
     IF AM < 1.0E-10 THEN
       BEGIN
       WRITELN;
       IFL := 1;
       WRITE ('AM= ', AM, 'INN = ', INN);
       WRITELN ('J = ', J);
       WRITE ('STOP, THIS RUN OF THE PROGRAMME "VORTEX"');
       WRITELN (' IS IMPOSSIBLE DUE TO FOUT IN MATSOLVE');
       END;  (*  end of if AM<  situation  *)
     IF AM < 1.0E-30 THEN
       BEGIN
       WRITE ('STOP, THIS RUN OF THE PROGRAMME "VORTEX"');
       WRITELN (' IS IMPOSSIBLE DUE TO SEVERE FOUT IN MATSOLVE');
       HALT;
       END;  (*  end of if AM<  situation  *)
(*****************************************************************)
(* Het J-de veld van ISUB gaat contain the column number where   *)
(* the maximale waarde van de J-de row occurred.                 *)
(*****************************************************************)
     IH := ISUB(.J.);
     ISUB(.J.) := ISUB(.INN.);
     ISUB(.INN.) := IH;
     PIVOTR := -1.0 / A(.J,ISUB(.J.).);
     FOR I := J+1 TO IDIM DO
         BEGIN
         A(.J,ISUB(.I.).) := PIVOTR * A(.J,ISUB(.I.).);
         FOR K := J+1 TO IDIM DO
             BEGIN
             G := A(.J,ISUB(.I.).) * A(.K,ISUB(.J.).);
             A(.K,ISUB(.I.).) := G + A(.K,ISUB(.I.).);
             END;
         END;  (*  end of FOR I iteration  *)
     FOR I := 1 TO IDIM DO BEGIN
         WRITELN;
         FOR K := 1 TO IDIM DO WRITE ('A(', I:3, ',', K:3, ')=',
                                   A(.I,K.):7:3, ' ');
         END;
     WRITELN (' J = ',J:4);
     END;   (*  end of for J iteration  *)
  END;   (*  end of IPAR = 1 condition  *)

IF IPAR = 2 THEN
  BEGIN
(*****************************************************************)
    FOR I := 1 TO IDIM DO
```

```
      BEGIN
      HULP :=MAX(ABS(B(.I.)) - 1.0E-60, 0.0);
      IF HULP <> 0 THEN    HULP := HULP/ABS(HULP);
      B(.I.) := HULP * B(.I.);
      END;  (*  end of for I iteration  *)
    FOR J := 1 TO IDM1 DO
      BEGIN
      XX(.J.) := B(.ISUB(.J.).);
      B(.ISUB(.J.).) := B(.J.);
      FOR I := J+1 TO IDIM DO
          BEGIN
          G := A(.J, ISUB(.I.).) *  XX(.J.);
          B(.ISUB(.I.).) := G + B(.ISUB(.I.).);
          END;
      END;  (*  end of for J iteration  *)
(******************************************************************)
(* Solve the equation by back substitution, starting with the    *)
(* last column of the Jordan form of the matrix (A).             *)
(******************************************************************)
    XX(.IDIM.) := B(.ISUB(.IDIM.).) / A(.IDIM, ISUB(.IDIM.).);
    FOR J := IDM1 DOWNTO 1 DO
        BEGIN
        FOR I := J+1 TO IDIM DO
            BEGIN
            G := A(.I, ISUB(.J.).) *  XX(.I.);
            XX(.J.) := XX(.J.) - G;
            END;
        XX(.J.) := XX(.J.) / A(.J, ISUB(.J.).) ;
        HULP := MAX(ABS(XX(.J.)) - 1.0E-60, 0.0);
        IF HULP<>0 THEN           HULP := HULP / ABS(HULP);
        XX(.J.) := HULP * XX(.J.);
        END;
   END; (* end of IPAR=2 situation  *)
END;  (* end of routine TAMSOLF  *)
```

APPENDIX 3

Source listing of velocity component calculation procedures

```
%PRINT ON;
%PAGE;
(****************************************************************)
(*                                                              *)
(*      HERE THE ESSENTIAL SUBROUTINES OF THE PROGRAM BEGIN     *)
(*                                                              *)
(****************************************************************)

(*==============================================================*)
(*                                                              *)
(*                         BOUNDCHECK                           *)
(*                                                              *)
(*==============================================================*)
(*  This procedure checks the boundary condition that should be *)
(*  met by the matrix manipulations in GEOMETRY and EMIT.       *)
(****************************************************************)
```

```
PROCEDURE BOUNDCHECK;
VAR I,K,L,P,I0             : INTEGER;
    SSUM,HULP              : REAL;
    G                      : COMPLEX;
BEGIN
  I0 := 0;
  FOR L := 1 TO NBDIES DO
    BEGIN
    FOR K := 1 TO NWALL(.L.) DO
      BEGIN
      FOR I := 1 TO NVORT DO
          BEGIN
          CSUB (WALL(.K,L.), Z(.I.), G);
          HULP := SIGMA2 + SQR(G.RE) + SQR(G.IM);
          CDIVR (G, HULP, G);
          CMUL (G, CMPLX(0.0, GAMMA(.I.) / (2*PI) ), G);
(*  Follows projection onto the normal in the wall point           *)
          HULP := G.RE *ZZ(.K,L.).RE + G.IM * ZZ(.K,L.).IM;
          HULP := HULP / CABS( ZZ(.K,L.) );
          B(.I.) := HULP;
          END;  (*  end of I iteration  *)
      SSUM := 0;
      FOR P := 1 TO NVORT DO SSUM := SSUM + B(.P.);
      HULP := UINF.RE * ZZ(.K,L.).RE + UINF.IM * ZZ(.K,L.).IM;
      HULP := HULP / CABS( ZZ(.K,L.) );
      PSI(.K,L.) := HULP + SSUM;
      END;  (* end of FOR K iteration  *)
(*******************************************************************)
    WRITELN;
    WRITELN ('Velocity components normal to the wall :');
    FOR K := 1 TO NWALL(.L.) DO
      BEGIN
      IF (K-1) MOD(4) = 0 THEN WRITELN (' ');
      HULP  := PSI(.K,L.);
      XS(.I0+K.) := HULP;
      WRITE ('VEL(', K:3,')= ', HULP:9:5, ' ' );
      END;  (*  end of FOR K iteration  *)
    WRITELN;
    WRITELN ('Dit was het ', L,'-de lichaam');
    I0 := I0 + NWALL(.L.);
    END;   (*  end of FOR L iteration  *)
(*******************************************************************)
END;   (*  end of routine BOUNDCHECK  *)
%PAGE;
(*===============================================================*)
(*                                                               *)
(*                          VELOCALC                             *)
(*                                                               *)
(*===============================================================*)
PROCEDURE VELOCALC (ZEE: COMPLEX);
    VAR J                    : INTEGER;
        DELZ2                : REAL;
        G,DELZ,ZSUM          : COMPLEX;
(*******************************************************************)
(*                                                               *)
```

```
(*   This subroutine calculates and prints velocity-components   *)
(*            induced at location ZEE                            *)
(*                                                               *)
(*****************************************************************)
  BEGIN
  ZSUM := CMPLX(0.0,0.0);
(*****************************************************************)
(*  NVORT discrete wervels worden verdisconteerd                 *)
(*****************************************************************)
  FOR J := 1 TO NVORT DO
      BEGIN
      CSUB(ZEE,Z(.J.),DELZ);
      DELZ2 := SIGMA2+SQR(DELZ.RE)+SQR(DELZ.IM);
      CDIVR(DELZ,DELZ2,DELZ);
      CMUL(DELZ,CMPLX(0.0, GAMMA(.J.)/(2*PI) ),G);
      CADD(ZSUM,G,ZSUM);
      END;
  CADD(ZSUM, UINF, ZSUM);
  WRITELN('Locatie: X = ', ZEE.RE:8:6,' Y = ', ZEE.IM:8:6,
          ' Velocity: VX = ', ZSUM.RE:8:6, ' VY = ', ZSUM.IM:8:6);
END;    (*  end of routine VELOCALC  *)
%PAGE;
(*==============================================================*)
(*                                                              *)
(*                           VELOCT                             *)
(*                                                              *)
(*==============================================================*)
PROCEDURE VELOCT (VAR VE      : CA1100;
                  VAR MERTST  : RA1100;
                  VAR B       : RA1100);
    VAR I,J,P             : INTEGER;
        DELZ2,SSUMX,SSUMY : REAL;
        G,DELZ,CHULP      : COMPLEX;
(*****************************************************************)
(*                                                              *)
(* Biot Savart interaction of vortices, positions Z(I),         *)
(* circulation GAMMA(I).  Velocity at infinity = UINF           *)
(* RC is the characteristic radius in the cut-off:              *)
(*              U(R) = (GAMMA/2PI) * R/(R*R + RC*RC)            *)
(*                                                              *)
(* Several different types of CORE's can be used                *)
(*                                                              *)
(*****************************************************************)
  BEGIN
    CMUL(UINF,CMPLX(0.0,-2*PI),CHULP);
    FOR I := 1 TO NVORT DO      VE(.I.) := CHULP;
(*****************************************************************)
(*  Compute interactions                                        *)
(*  Loop on first vortex                                        *)
(*****************************************************************)
    FOR I:=2 TO NVORT DO
(*****************************************************************)
(*  Loop on second vortex                                       *)
(*****************************************************************)
      BEGIN
```

```
      FOR J := 1 TO I-1 DO
        BEGIN
        CSUB(Z(.I.), Z(.J.), DELZ);
        DELZ2 := SIGMA2 + SQR(DELZ.RE) + SQR(DELZ.IM);
        CDIVR(DELZ, DELZ2, DELZ);
(****************************************************************)
(* MERTST fungeert als X component van de snelheid             *)
(* B fungeert als Y component van de snelheid                   *)
(****************************************************************)
        MERTST(.J.) := DELZ.RE * GAMMA(.J.);
        B(.J.) := DELZ.IM * GAMMA(.J.);
        CMULR(DELZ,GAMMA(.I.),G);
        CSUB(VE(.J.),G,VE(.J.));
        END;  (*  end of J iteration  *)
      SSUMX := 0;
      SSUMY := 0;
      FOR P := 1 TO I-1 DO
         BEGIN
         SSUMX := SSUMX + MERTST(.P.);
         SSUMY := SSUMY + B(.P.);
         END;
      CADD(VE(.I.),CMPLX(SSUMX,SSUMY),VE(.I.));
      END;  (*  end of for I iteration  *)
(****************************************************************)
(*   Multiply by i / 2Pi                                        *)
(****************************************************************)
    FOR I := 1 TO NVORT DO
      CMUL(VE(.I.),CMPLX(0.0,0.5/PI),VE(.I.));
END;  (*  end of subroutine VELOCT  *)

%PAGE;
(*==============================================================*)
(*                                                              *)
(*                            MOVE                              *)
(*                                                              *)
(*==============================================================*)
PROCEDURE MOVE (VAR MERTST     : RA1100;
                VAR B          : RA1100;
                VAR VM,Z       : CA1100);
  VAR I   : INTEGER;
      G,H : COMPLEX;
      VE  : CA1100;
(****************************************************************)
(*                   Motion of the vortices.                    *)
(*  Old vortices use ADAMS-BASHFORTH-2, new ones use EULER      *)
(*  explicit.                                                   *)
(****************************************************************)
  BEGIN
(****************************************************************)
(*             Compute velocities of the vortices.              *)
(****************************************************************)
   VELOCT (VE,MERTST,B);
   WRITELN('EINDE VELOCT');
(****************************************************************)
(*                        Move vortices                         *)
```

```
(*              ADAMS-BASHFORTH-2 FOR THE OLD VORTICES.                 *)
(**********************************************************************)
    FOR I := 1 TO NOLD DO
        BEGIN

(* VM(.I.) representeert de "oude" waarde van de snelheid              *)

        G := CMPLX(VE(.I.).RE * 1.5 * DELT,VE(.I.).IM * 1.5 * DELT);
        H := CMPLX(VM(.I.).RE * 0.5 * DELT,VM(.I.).IM * 0.5 * DELT);
        CSUB(G,H,G);
        CADD(Z(.I.),G,Z(.I.));
        VM(.I.) := VE(.I.);
        END;  (*  end of for I iteration  *)
(**********************************************************************)
(*                  EULER explicit for the new vortices                *)
(**********************************************************************)
    WRITELN('FOR (NOLD + 1) TO NVORT');
    FOR I := NOLD + 1 TO NVORT DO
        BEGIN
        G := CMPLX(VE(.I.).RE * DELT,VE(.I.).IM * DELT);
        CADD(Z(.I.),G,Z(.I.));
        VM(.I.) := VE(.I.);
        END;
END;  (*  end of routine MOVE  *)
```

APPENDIX 4

Main block of VORTEX and saving and reading procedures

```
PROGRAM VORT2 (INPUT, FILE2, FILE3, OUTPUT);

(**************************************************************)
(*                                                            *)
(*                    V O R T 2                               *)
(*                    =========                               *)
(*                                                            *)
(*                                                            *)
(*                                                            *)
(*     FILE allocated on disk to read parameters and to read  *)
(*     and write locations and strengths of vortices :        *)
(*              WERVEL(file3), of WERFTW(file2)               *)
(*              VORTICES of VORTJE2 voor het plotten          *)
(*              INPUT en OUTPUT zijn de standaard lees en     *)
(*                    schrijf files (achter de source)        *)
(*     Bij switchen tussen different versions the following   *)
(*     parameters should be accounted for properly :          *)
(*              destroywidth (in ABSORB)                      *)
(*              merging parameters, dependent of the region   *)
(*     ABSORB MAG SLECHTS TWEE MAAL PER N-CYCLE WORDEN AANGE- *)
(*     ROEPEN INDIEN X(.I0.) GOED WORDT MEEGENOMEN            *)
(*                                                            *)
(*     VARIABLES :                                            *)
(*                                                            *)
(*     UINF     Uniform velocity at infinity                  *)
(*              UINF is complex                               *)
(*     ABSUIN   Magnitude of UINF                             *)
(*     ALPHA    Incidence in degrees                          *)
(*     NVORT    Number of vortices                            *)
(*     Z        Positions of vortices                         *)
(*              Z is a complex array (max. 2000)              *)
(*     VM       Velocities of vortices                        *)
(*                  VM = VX + i VY                            *)
(*              VM is a complex array (max. 2000)             *)
(*     G, H     are used only to facilitate the complex calcu-*)
(*              lations. These complex variables have no other*)
(*              use and replace HULP,HULP1,... if desired     *)
(*     B, Y,                                                  *)
(*     U, V,                                                  *)
(*     XS, XX,                                                *)
(*     MERTST   zijn alle locaal gebruikte arrays die i.v.m.  *)
(*              de ruimte globaal gedefinieerd zijn           *)
(*     SIGMA    Core-radius                                   *)
(*     SIGMA2   SQR(SIGMA)                                    *)
(*     N2       een doorgeefveld (file3), dat nergens voor    *)
(*              wordt gebruikt                                *)
(*     XPLOT,                                                 *)
(*     YPLOT,                                                 *)
(*     IAR, NJ  globaal gedeclareerde arrays t.b.v. de        *)
```

```
(*              plotroutines                                        *)
(*                                                                  *)
(*     The program computes unsteady flows, superposing and        *)
(*     starting from a vortex-free potential flow.                  *)
(*     The solid shape is arbitrary, given by routines SOLID        *)
(*     and SOLID1. It can be made of several separate bodies.      *)
(*                                                                  *)
(*******************************************************************)
CONST BOUT = 1100;
TYPE COMPLEX = RECORD
     RE,IM : REAL
END;
TYPE   IAR2 = ARRAY(.1..2.) OF INTEGER;
      IA100 = ARRAY(.1..100.) OF INTEGER;
     IA1100 = ARRAY(.1..BOUT.) OF INTEGER;
     IA0600 = ARRAY(.0..600.) OF INTEGER;
    IA01100 = ARRAY(.0..BOUT.) OF INTEGER;
     IAR152 = ARRAY(.1..15,1..2.) OF INTEGER;
     IAR500 = ARRAY(.1..500.) OF INTEGER;
   IAR01100 = ARRAY(.0..BOUT.) OF INTEGER;
       RAR2 = ARRAY(.1..2.) OF REAL;
     RA0500 = ARRAY(.0..500.) OF REAL;
    RA01100 = ARRAY(.0..BOUT.) OF REAL;
    SRA0500 = ARRAY(.0..500.) OF SHORTREAL;
   SRA01100 = ARRAY(.0..BOUT.) OF SHORTREAL;
      RA100 = ARRAY(.1..100.) OF REAL;
      RA500 = ARRAY(.1..500.) OF REAL;
     RA1100 = ARRAY(.1..BOUT.) OF REAL;
     RA5001 = ARRAY(.1..500,1..1.) OF REAL;
   RA500500 = ARRAY(.1..500,1..500.) OF REAL;
       CAR2 = ARRAY(.1..2.) OF COMPLEX;
      CA500 = ARRAY(.1..500.) OF COMPLEX;
     CA1100 = ARRAY(.1..1100.) OF COMPLEX;
     CA5001 = ARRAY(.1..500,1..1.) OF COMPLEX;
    CA05001 = ARRAY(.0..500,1..1.) OF COMPLEX;
   CA500500 = ARRAY(.1..500,1..500.) OF COMPLEX;
    STRINGN = PACKED ARRAY (.1..90.) OF CHAR;
     STRTEK = ARRAY (.1..80.) OF CHAR;
(*******************************************************************)
(*  Example : ca2000 = complex array, max 2000                      *)
(*******************************************************************)

VAR  STARTCODE, N, I,
     NBDIES, NDES, NDIM, NEND, NOLD,
     NPTS, NSTART, NSTEP, NVORT,
     N2, INDEX, ITIME                 : INTEGER;

     AA, ABSUIN, ALPHA, ARCL, CHARD,
     DELT, D0, GAMMA0, TUBERADIUS,
     PI, SIGMA2, T, V0, YYENAMAX,
     HULP1, R0                        : REAL;
     DELTAS                           : SHORTREAL;
     AVFO, DELZ, UINF, ZET            : COMPLEX;
     NINC, NWALL                      : IAR2;
     INC                              : IAR152;
```

```
     ISUB                             : IA1100;
     MOM, IKSMAX                      : RAR2;
     GAMMA, MERTST, PS, B             : RA1100;
     DPDS, PSI, THETA                 : RA5001;
     VM, Z                            : CA1100;
     WALL, ZCR                        : CA05001;
     ZZ                               : CA5001;
     FORCE, HUB, Z0                   : CAR2;
     FILE2                            : TEXT;
     FILE3                            : TEXT;
     NONCLOSED, EXTRA, MATRIX         : BOOLEAN;
     A                                : RA500500;
     XX                               : RA500;
     U,V                              : RA01100;
     X,XS                             : RA01100;
     Y                                : RA01100;
     XPLOT, YPLOT                     : SRA01100;
     IAR, NJ                          : IA01100;

(*%PRINT OFF;*)
%PAGE;
```

```
%PAGE;
(*==============================================================*)
(*                                                              *)
(*                       INIT                                   *)
(*                                                              *)
(*==============================================================*)
PROCEDURE INIT (VAR NSTART,NVORT     : INTEGER;
                VAR T,V0             : REAL;
                VAR GAMMA            : RA1100;
                VAR VM,Z             : CA1100);
VAR I : INTEGER;
(**************************************************************)
(*  Initialize time dependent variables.                      *)
(**************************************************************)
BEGIN
  IF STARTCODE <> 1 THEN
(**************************************************************)
(*  Case of a start from a previous run.                      *)
(**************************************************************)
    BEGIN
    READ (FILE2,NSTART,T,NVORT);
    READ (FILE2,V0);
    FOR I := 1 TO NVORT DO      READ (FILE2, Z(.I.).RE, Z(.I.).IM);
    FOR I := 1 TO NVORT DO      READ (FILE2, GAMMA(.I.) );
    FOR I := 1 TO NVORT DO      READ (FILE2, VM(.I.).RE, VM(.I.).IM);
    NSTART := NSTART + 1;
     WRITELN('FILE2 IS INGELEZEN IN INIT');
     WRITELN('gammas 1 en NVORT zijn ',GAMMA(.1.),GAMMA(.NVORT.));
     WRITELN('VM(1) = ',VM(.1.).RE,NVORT);
     WRITELN('V0 from INIT = ',V0);
    END;  (*  end of startcode <> 1 situation  *)
(**************************************************************)
(*  Case of a start from vortex-free potential flow.          *)
(*  Start at step 1 with time T=0 and no vortices.            *)
(**************************************************************)
  IF STARTCODE = 1 THEN
    BEGIN
    NSTART := 1;
    T := 0;
    NVORT := 0;
(**************************************************************)
(*  Give phony values to the vortex positions and circulations.  *)
(**************************************************************)
    FOR I := 1 TO 300 DO
      BEGIN
      Z(.I.) := CMPLX(10.0,0.0);
      VM(.I.) := CMPLX(0.0,0.0);
```

```
      GAMMA(.I.):= 0.0;
      END;
(*****************************************************************)
(*  A tentative value for V0, which will be adjusted later.      *)
(*****************************************************************)
    V0 := 1.0E-5 * ABSUIN*EXP (3*LN(CHARD/D0));
    V0 := 0.02;
    WRITELN('V0 (tentative value) =', V0:7:3 );
    END;   (* end of ISTART<>0 situation  *)
END;  (*  end of routine INIT  *)
%PAGE;

(*===============================================================*)
(*                                                               *)
(*                          READPRINT                            *)
(*                                                               *)
(*===============================================================*)
PROCEDURE READPRINT(VAR NDES, N2                    : INTEGER;
                    VAR ABSUIN, ALPHA, D0, GAMMA0 : REAL);
(*****************************************************************)
(*   ABSUIN        Modulus of UINF                               *)
(*   ALPHA         Incidence in degrees                          *)
(*   D0            Parameter in merging device.                  *)
(*   D0 smaller puts more vortices near the solid and less far   *)
(*   from it.                                                    *)
(*   DO moet ongeveer 5% van de maximale dimensie van een lichaam*)
(*   zijn, of 50% van de afstand tussen lichamen.                *)
(*   GAMMA0        allows the user to disturb the flow to make   *)
(*                 it reach the shedding regime faster.          *)
(*   GAMMA0 = 0    leaves it undisturbed.                        *)
(*   GAMMA0 <> 0   artificially adds a circulation GAMMA0 at     *)
(*                 the beginning of the run.(GAMMA0 is ignored   *)
(*                 if ISTART = 0).                               *)
(*****************************************************************)
BEGIN
  IF STARTCODE <> 1 THEN
      BEGIN
      READ (FILE2, NDES, N2);
    (*  one time adjustment of NDES is allowed                     *)
    (*  a statement 'NDES :=' should be discarded at other times  *)
      READ (FILE2, ABSUIN, ALPHA);
      READ (FILE2, D0, GAMMA0);
      END;
  IF STARTCODE = 1 THEN
      BEGIN
      NDES := 900;
      N2 := 200;
      ABSUIN := 1;
      ALPHA := 0;
      D0 := 0.05;
      GAMMA0 := 0.0;
      END;
  UITLN(2);
  WRITELN('VORTEX : SIMULATION OF 2-DIMENSIONAL FLOW');
  WRITELN;
```

```
  IF STARTCODE = 1 THEN
    WRITELN(' This run started with circulation: ',
             GAMMA0 : 8:2);
  WRITELN;
  WRITELN(' Approximate number of vortices: ', NDES : 6);
  WRITELN;
  WRITELN(' vortex free stream velocity magnitude ',
            ABSUIN : 7:4,' with alpha ', ALPHA : 7:4);
  WRITELN;
  WRITELN(' time step ', DELT : 7:4);
  WRITELN;
  WRITELN(' characteristic dimension in merging device ',
            D0 : 7:4);
END;

%PAGE;
(*=============================================================*)
(*                                                             *)
(*                          PARAMS                            *)
(*                                                             *)
(*=============================================================*)
(*                                                             *)
(*  PARAMETERS:                                                *)
(*                                                             *)
(*  STARTCODE = 1   if run is from time 0                      *)
(*  STARTCODE = 0   if it is a follow-up                       *)
(*  NSTEP           number of steps                            *)
(*  DELT            time step; DELT must be smaller than       *)
(*                  DELTAS / ABS(U)                            *)
(*                                                             *)
(***************************************************************)
PROCEDURE PARAMS (VAR STARTCODE, NSTEP          : INTEGER;
                  VAR PI, DELT                  : REAL;
                  VAR EXTRA, MATRIX             : BOOLEAN );

BEGIN
PI := 4 * ARCTAN(1.0) ;
EXTRA := TRUE ;
STARTCODE := 0 ;
NSTEP := 19;
DELT := 0.010 ;
MATRIX := TRUE;
END;  (*  end of routine PARAMS  *)

%PAGE;
(***************************************************************)
(*               Einde van de procedures                       *)
(***************************************************************)
%PAGE;
(*=============================================================*)
(*                                                             *)
(*                                                             *)
(*               MAIN BODY OF THIS PROGRAMME                   *)
(*                                                             *)
(*                                                             *)
```

```
(*==============================================================*)
(*                                                              *)
(*  PARAMETERS:                                                 *)
(*                                                              *)
(*  STARTCODE = 1    if run is from time 0                      *)
(*  STARTCODE = 0    if it is a follow-up                       *)
(*  NSTEP            number of steps                            *)
(*  DELT             time step; DELT must be smaller than       *)
(*                   DELTAS / ABS(U)                            *)
(*  DELTAS           the smallest distance between              *)
(*                   DELTAS = 1/50 for  SOLID1                  *)
(****************************************************************)
BEGIN
  VSCOM;
  FSPIE;
(*  dit waren FORTRAN-routines voor plotten en timing resp.  *)

 PARAMS (STARTCODE, NSTEP, PI, DELT, EXTRA, MATRIX) ;

  RESET (FILE2);
 READPRINT(NDES,N2,ABSUIN,ALPHA,D0,GAMMA0);

  WRITELN('einde READPRINT');
(****************************************************************)
(*                    SET UP THE GEOMETRY                       *)
(*     Define the solid and the creation points on it's surface. *)
(*     Compute and GAUSS eliminate matrix of influence          *)
(*     coefficients between wall points, and do other things    *)
(*     that depend only on the solid.                           *)
(****************************************************************)

 GEOMETRY(ISUB,WALL,NWALL,SIGMA2,CHARD,NBDIES,DELTAS,XX,TUBERADIUS,
        NONCLOSED,THETA,NDIM,HUB,IKSMAX,A,NINC,INC,ZCR,Z0,B,R0,ZZ);

  WRITELN('einde GEOMETRY');
(****************************************************************)
(*                         INITIALIZE                           *)
(*              the time dependent variables                    *)
(*                                                              *)
(*    Read and print the parameters out of the file behind the  *)
(*    source. RESET closes the file and, if necessary, places   *)
(*    the pointer on first field to allow for reading.          *)
(****************************************************************)

 INIT(NSTART, NVORT, T, V0, GAMMA, VM, Z);

  WRITELN('einde INIT');
(****************************************************************)
(* ALPHA is de hoek in graden tussen de horizontale x-as en de  *)
(* uniforme snelheid van de vortex vrije initiele oplossing.    *)
(****************************************************************)
  CEXP( CMPLX(0.0, ALPHA*ARCTAN(1.0) / 45.0 ), UINF);
  UINF := CMPLX (UINF.RE*ABSUIN, UINF.IM*ABSUIN);
  AVFO := CMPLX (0.0, 0.0);
  IF (DELT > (DELTAS / ABSUIN)) THEN
```

```
    BEGIN
    WRITELN('TIME STEP TOO LARGE; DELT = ',DELT,' DELTAS =',DELTAS);
    HALT;
    END;
(*****************************************************************)
(*     Main loop; Advance flow time step by time step.            *)
(*****************************************************************)
  SETTIM;
  UITLN(2);
  WRITELN(' Step by step evolution of the flow :');
  UITLN(2);
  NEND := NSTART + NSTEP - 1;

  FOR N := NSTART TO NEND DO
      BEGIN
      WRITELN('**********************************************');
      WRITELN('N =',N);
(*****************************************************************)
(* The body absorbs vortices and emits new vortices to account   *)
(* for it, plus some new vorticity which allow the velocity      *)
(* field to satisfy the boundary condition :                     *)
(*                        U = V = 0                              *)
(*                                                               *)
(* Detect and absorb vortices that crashed into the wall.        *)
(* Start computing pressure and force.                           *)
(*****************************************************************)

  IF N <> 1 THEN ABSORB(X, MOM, FORCE, DPDS, NVORT, GAMMA, VM, Z);

   WRITELN('EINDE ABSORB');
  IF N <> NSTART THEN
     BEGIN
     IF (N MOD(2)=0) OR (N=NEND)
            THEN  HULPPLOT(NVORT, TUBERADIUS, XS, Y, U, V, EXTRA);
     END;  (*  end of IF N <> condition  *)
(*****************************************************************)
(*     Merge vortices to keep their number reasonable.           *)
(*****************************************************************)

     MERGE(MERTST, B, NVORT, V0, GAMMA, VM, Z);

     WRITELN ('einde MERGE');

(*   IF (N > 1) AND (N < NSTART + 1)
              THEN ABSORB(X, MOM, FORCE, DPDS, NVORT, GAMMA, VM, Z);
     WRITELN ('einde tweede ABSORB');       *)
(*****************************************************************)
(*   DE TWEEDE ABSORB MAG ALLEEN ALS er rekening wordt gehouden  *)
(*   met de X(I0) die opnieuw op nul gesteld wordt in ABSORB     *)
(*****************************************************************)
IF (N < NSTART + 2) AND (N <> 1) THEN BEGIN
            WRITELN ('Volgt extra HULPPLOT achter MERGE en 2de ',
                          'ABSORB');
            HULPPLOT(NVORT, TUBERADIUS, XS, Y, U, V, EXTRA);
            END;
```

```
(*******************************************************************)
(* Emit new vortices to satisfy boundary condition, and finish   *)
(* computing pressure, force, etc.                                *)
(* Treat boundary condition at the body by an exchange of        *)
(* vorticity                                                      *)
(*******************************************************************)

 EMIT(XX, NVORT, T, MOM, DPDS, PSI, GAMMA, PS, X, Z, FORCE, B, NOLD);

 BOUNDCHECK;
  WRITELN('einde EMIT');
 IF (N=NSTART + 2) THEN BEGIN
   ZET := CMPLX(0.0, 0.0);
   VELOCALC(ZET);
   ZET := CMPLX(0.0, 0.1*DELTAS);
   VELOCALC(ZET);
   ZET := CMPLX(0.0, 0.2*DELTAS);
   VELOCALC(ZET);
   FOR I:= 3 TO 13 DO BEGIN
     ZET := CMPLX(0.0, 0.1 * I *DELTAS);
     VELOCALC(ZET);
     END;
   END;  (* end of IF N= situation *)

 IF N MOD(2) = 0 THEN BEGIN
     FOR I:= 1 TO 21 DO BEGIN
        ZET := CMPLX(0.0, -1 + (I-1)*0.1);
        VELOCALC(ZET);
        END;
     END;

 IF (N MOD(2)=0) OR (N=NSTART) OR (N=NEND)
          THEN  HULPPLOT(NVORT, TUBERADIUS, XS, Y, U, V, EXTRA);

 ASKTIM(ITIME);
 IF (ITIME*100 MOD(60) = 0) AND (N>NSTART+ 52) THEN
       BEGIN;
       REWRITE (FILE3);
       WRITELN(FILE3,NDES,'    ',N2,'    ');
       WRITELN(FILE3,ABSUIN,'    ',ALPHA,'    ');
       WRITELN(FILE3,D0,'    ',GAMMA0,'    ');
       WRITELN(FILE3,N,'    ',T,'    ',NVORT,'    ');
       WRITELN(FILE3,V0,'    ');
       WRITECANTOT(Z,NVORT);
       WRITERANTOT(GAMMA,NVORT);
       WRITECANTOT(VM,NVORT);
       WRITELN('GAMMAs 1 en NVORT zijn ',GAMMA(.1.),GAMMA(.NVORT.));
       WRITELN('************  ************');
       END;

  CADD(AVFO, FORCE(.1.), AVFO);
(*******************************************************************)
(*    Move vortices.                                               *)
(*******************************************************************)
```

```
   IF N <> NEND THEN MOVE(MERTST, B, VM, Z);

      WRITELN('einde MOVE');
(*    IF (N MOD(7) = 0) OR (N=1) OR (N=NEND)
          THEN HULPPLOT(NVORT, TUBERADIUS, XS, Y, U, V, EXTRA); *)
      WRITELN('einde ',N:2,' de hulpplot achter MOVE');
      UITLN(2);
    END;          (*  end of main loop  *)
(******************************************************************)
(*                    END OF MAIN LOOP                          *)
(*                                                              *)
(*     Store results in case we want a follow up to this run.   *)
(******************************************************************)
IF N > 52 THEN BEGIN      (*   N>1  NORMAAL  *)
  WRITELN(N, T, NVORT);
  WRITELN (V0);
  REWRITE (FILE3);
  WRITELN(FILE3, NDES,'   ',N2,'   ');
  WRITELN(FILE3, ABSUIN,'   ',ALPHA,'   ');
  WRITELN(FILE3, D0,'   ',GAMMA0,'   ');
  WRITELN(FILE3, N,'   ',T,'   ',NVORT,'   ');
  WRITELN(FILE3, V0,'   ');
  WRITECANTOT(Z, NVORT);
  WRITERANTOT(GAMMA, NVORT);
  WRITECANTOT(VM, NVORT);
  WRITELN('GAMMAS 1 en NVORT zijn',GAMMA(.1.), GAMMA(.NVORT.) );
(******************************************************************)
(*     Output average loads                                     *)
(******************************************************************)
  AVFO := CMPLX (AVFO.RE/NSTEP, AVFO.IM/NSTEP);
  WRITELN;
  WRITELN(' AVERAGE DRAG AND LIFT: ',
           AVFO.RE : 8:4, AVFO.IM : 8:4);
  WRITELN('Files zijn weggeschreven');
  END;  (*  end of storing results  *)
END.
```

APPENDIX 5

Source listing of Pascal routines for plotting on IBM.
Second version (no dedicated plotting utilities needed)

```
%PAGE;
(******************************************************************)
(*                                                                *)
(*  PROCEDURE  FSPIE    maakt het mogelijk een CPU-tijd teller    *)
(*                      mee te laten lopen                        *)
(*             SETTIM   zet de CPU-tijd teller op nul             *)
(*             ASKTIM   vraagt de CPU-tijd                        *)
(*                                                                *)
(*  De CPU-tijd in centiseconden komt uit deze FORTRAN routines   *)
(*                                                                *)
(******************************************************************)
PROCEDURE FSPIE;
FORTRAN;

PROCEDURE SETTIM;
FORTRAN;

PROCEDURE ASKTIM (VAR  ITIME   :   INTEGER);
FORTRAN;

%PAGE;
(***********************************************************************)
(*                                                                     *)
(*                PROCEDURES FOR PLOTTING, CURRENTLY                   *)
(*                  AVAILABLE ON THE IBM MAINFRAME                     *)
(*                                                                     *)
(*                                                                     *)
(*             VSCOM    Opent mogelijkheid tot het aanroepen van       *)
(*                      FORTRAN procedures. Al de binnen dit kader     *)
(*                      verklaarde standaard tekenprocedures zijn      *)
(*                      FORTRAN geschreven.                            *)
(*             PLOTS    Reserveert papier voor volgende tekening       *)
(*             J06ABF   Tekent assen met marks                         *)
(*             J06ADF   Tekent grid                                    *)
(*             J06AFF   Tekent grens met getallen                      *)
(*             J06AJF   Tekent titel langs assen                       *)
```

```
(*              J06WAF   Initialiseert plotroutines                              *)
(*              J06WBF   definieert viewpoort                                    *)
(*              J06WCF   past grootte viewpoort aan                              *)
(*              J06WZF   Sluit plotsysteem af                                    *)
(*              J06YAF   verplaatst pen zonder te tekenen                        *)
(*              J06YCF   verplaatst pen met tekenen                              *)
(*              J06YHF   Tekent reeks van letters                                *)
(*              J06YKF   Zet karaktergrootte                                     *)
(*              J06YLF   Zet karakter spatiering                                 *)
(*              J06YMF   Verandert kleur pen                                     *)
(*                                                                               *)
(*  Voor de nadere verklaring van de variabelen die in de boven-                 *)
(*  staande, zogenaamde NAG-procedures gebruikt worden, zult U de                *)
(*  handleiding, die hiervoor bestaat, moeten raadplegen.                        *)
(*                                                                               *)
(*********************************************************************************)

PROCEDURE VSCOM;
FORTRAN;

PROCEDURE PLOTS(CONST IS,IL:INTEGER);
FORTRAN;

PROCEDURE J06ABF(CONST DX,DY:REAL);
FORTRAN;

PROCEDURE J06ADF(CONST DX,DY:REAL);
FORTRAN;

PROCEDURE J06AFF(CONST DX,DY:REAL);
FORTRAN;

PROCEDURE J06AJF(CONST IAXIS :INTEGER;
                 CONST ITITLE:STRTEK;
                 CONST NCHAR :INTEGER);
FORTRAN;

PROCEDURE J06WAF;
FORTRAN;

PROCEDURE J06WBF(CONST  XMIN,XMAX,YMIN,YMAX:REAL;
                 CONST  MARGIN:INTEGER);
FORTRAN;

PROCEDURE J06WCF (CONST P1, P2, Q1, Q2 : REAL);
FORTRAN;

PROCEDURE J06WZF;
FORTRAN;

PROCEDURE J06YAF(CONST X,Y:REAL);
FORTRAN;

PROCEDURE J06YCF(CONST X,Y:REAL);
FORTRAN;
```

```
PROCEDURE J06YHF(CONST ICHAR:STRTEK;
                 CONST N    :INTEGER);
FORTRAN;

PROCEDURE J06YKF(CONST WIDTH,HEIGHT:REAL);
FORTRAN;

PROCEDURE J06YLF(CONST DX,DY:REAL);
FORTRAN;

PROCEDURE J06YMF(CONST IPEN:INTEGER);
FORTRAN;

%PAGE;
(*==================================================================*)
(*                                                                  *)
(*                      READTEKST                                   *)
(*                                                                  *)
(*==================================================================*)
(*                                                                  *)
(*  Procedure READTEKST leest variabele TEKST in en schrijft deze  *)
(*  weg onder de variabele NAME.                                    *)
(*                                                                  *)
(*  NCHAR = aantal karakters waaruit TEKST bestaat                  *)
(*  TEKST = stringvariabele waarin TEKST opgeslagen is              *)
(*                                                                  *)
(********************************************************************)

PROCEDURE READTEKST (    NCHAR : INTEGER;
                         TEKST : STRING(30);
                      VAR NAME : STRTEK);
VAR I : INTEGER;

BEGIN
   I := 1;
   REPEAT
      NAME(.I.) := TEKST(.I.);
      I := I + 1;
   UNTIL I > NCHAR;
END;

%PAGE;
(*==================================================================*)
(*                                                                  *)
(*                         DRAWTEKST                                *)
(*                                                                  *)
(********************************************************************)
(*                                                                  *)
(*  Procedure DRAWTEKST schrijft tekst langs assen van assenkruis,  *)
(*  maar wel aan de buitenkant van de viewpoort.                    *)
(*  De aanroep van de procedure ziet er als volgt uit:              *)
(*                                                                  *)
(*             DRAWTEKST('TITELX','TITELY')                         *)
(*                                                                  *)
```

```
(*  NCHAR  = aantal charakters waaruit de tekst bestaat              *)
(*  TITELX = de tekst die langs de X-as komt te staan                *)
(*  TITELY = de tekst die langs de Y-as komt te staan                *)
(*                                                                   *)
(*********************************************************************)

PROCEDURE DRAWTEKST (    TITELX, TITELY     : STRING(30) );

VAR NCHAR : INTEGER;
    NAME  : STRTEK;

BEGIN
   J06YMF(1);
   NCHAR := LENGTH(TITELX);
   READTEKST(NCHAR, TITELX, NAME);
   J06AJF(1, NAME, NCHAR);
   NCHAR := LENGTH(TITELY);
   READTEKST(NCHAR, TITELY, NAME);
   J06AJF(2, NAME, NCHAR);
END;

%PAGE;
(*=================================================================*)
(*                                                                 *)
(*                         PLTEKST                                 *)
(*                                                                 *)
(*=================================================================*)
(*                                                                 *)
(*  Procedure PLTEKST schrijft een bepaalde tekst bij de plot, te  *)
(*  beginnen bij het punt (XST, YST).                              *)
(*  Deze punten kunt U zelf aanpassen bij de aanroep van de pro-   *)
(*  cedure. De aanroep van de procedure ziet er als volgt uit:     *)
(*                                                                 *)
(*                    PLTEKST('TEKST', XST, YST)                   *)
(*                                                                 *)
(*                           TEKST = de te plotten tekst           *)
(*                           XST   = startpunt in X-richting       *)
(*                           YST   = startpunt in Y-richting       *)
(*                                                                 *)
(*  Ook is het mogelijk om de grootte en de spatiering van de      *)
(*  tekst in te stellen met de respectievelijk de subroutines      *)
(*  J06YKF en J06YLF, en de variabelen HEIGHT en WIDTH, die in     *)
(*  deze procedure staan.                                          *)
(*  Als en de grootte en de spatiering aangepast moeten worden,    *)
(*  dan kan dat het eenvoudigst met de variabelen HEIGHT en        *)
(*  WIDTH, als of de grootte of de spatieering aangepast moeten    *)
(*  worden, dan kan dat het gemakkelijkst met de procedures        *)
(*  J06YKF en J06YLF.                                              *)
(*                                                                 *)
(*  GROOTTE = teller waarmee onderzocht wordt of grote of kleine   *)
(*            letters afgedrukt moeten worden                      *)
(*  HULPSTR = array waarin tekst opgeslagen wordt                  *)
(*  I       = teller                                               *)
(*  NCHAR   = aantal karakters waaruit tekst bestaat               *)
(*                                                                 *)
```

```
(*  DX      = normale grootte en spatieering in X-richting          *)
(*            ter grootte van 0.01 * (XMAX - XMIN)                  *)
(*  DY      = normale grootte en spatieering in Y-richting          *)
(*            ter grootte van 0.01 * (YMAX - YMIN)                  *)
(*  HEIGHT  = grootte en spatieering in Y-richting                  *)
(*  WIDTH   = grootte en spatieering in X-richting                  *)
(*                                                                  *)
(********************************************************************)

PROCEDURE PLTEKST (TEKST    : STRING(80);
                   XST, YST : REAL;
                   DX, DY   : REAL );

VAR GROOTTE : INTEGER;
    HEIGHT  : REAL;
    HULPSTR : STRTEK;
    I       : INTEGER;
    NCHAR   : INTEGER;
    WIDTH   : REAL;

BEGIN
   WIDTH := 1.35 * DX;
   HEIGHT := 1.85 * DY;
   NCHAR := LENGTH(TEKST);
   J06YMF(4);
   J06YAF(XST,YST);
   FOR I := 1 TO NCHAR DO
   BEGIN
      HULPSTR(.1.) := TEKST(.I.);
      GROOTTE := ORD(HULPSTR(.1.));
      IF (GROOTTE>128) AND (GROOTTE<170) THEN
         BEGIN
         HULPSTR(.1.) := CHAR(GROOTTE + 64);
         J06YKF(0.30 * WIDTH, 0.30 * HEIGHT);
         J06YLF(0.6 * WIDTH, 0);
         END
      ELSE
         BEGIN
         J06YKF(0.45 * WIDTH, 0.45 * HEIGHT);
         J06YLF(0.8 * WIDTH, 0);
         END;
      J06YHF(HULPSTR, 1);
   END;   (*  end of I iteration  *)
END;   (*  end of subroutine PLTEKST  *)

%PAGE;
(*==============================================================*)
(*                                                              *)
(*                         ARROWHEAD                            *)
(*                                                              *)
(*==============================================================*)
(*                                                              *)
(*  Procedure ARROWHEAD tekent een pijltje vanaf punt           *)
(*  (XPLOT(.I.),YPLOT(.J.)) tot punt (UP,VP).                   *)
(*                                                              *)
```

```
(*  The point of the arrow will always be located on (UP, VP).   *)
(*  The tail (in red) will always be drawn from (XPLOT, YPLOT)   *)
(*  to the centre of the basis of the arrowhead. This implies    *)
(*  an unlogical 'inner' tail if the arrowhead is too big.       *)
(*                                                               *)
(*  De aanroep van de procedure ziet er als volgt uit:           *)
(*                                                               *)
(*               ARROWHEAD(UP, VP, HEIGHT, ALPHA)                *)
(*                                                               *)
(*  ALPHA           =  hoek, met positieve X-as (de horizontale  *)
(*                     as van de plot), waaronder de pijl moet   *)
(*                     te staan                                  *)
(*  DX, DY          =  schalingsfactoren, die voor de juiste     *)
(*                     vorm van de pijlpunt gebruikt moeten wor- *)
(*                     den, als de schaal langs de X- en Y-as    *)
(*                     verschillend is                           *)
(*  HEIGHT          =  te kiezen grootte van de pijlpunt         *)
(*  UP, VP          =  coordinaten van de punt van de pijlpunt   *)
(*  X1, X2, X3      =  X-coordinaten van drie punten op de basis *)
(*                     van de driehoek die de pijlpunt vormt     *)
(*  Y1, Y2, Y3      =  Y-coordinaten van drie punten op de basis *)
(*                     van de driehoek die de pijlpunt vormt     *)
(*                                                               *)
(*                          (UP,VP)                              *)
(*                             .                                 *)
(*                            /|\                                *)
(*                           / | \                               *)
(*                          /  |  \                              *)
(*                         /(X2,Y2)\                             *)
(*                  (X1,Y1).----.----.(X3,Y3)                    *)
(*                              |                                *)
(*                              |                                *)
(*                              .(XPLOT(.I.),YPLOT(.J.))         *)
(*                                                               *)
(*****************************************************************)

PROCEDURE ARROWHEAD (UP, VP, HEIGHT, ALPHA, DX, DY   : REAL);

VAR X1, X2, X3, Y1, Y2, Y3 : REAL;

BEGIN
   X2 := UP - HEIGHT*15*COS(ALPHA)*DX;
   Y2 := VP - HEIGHT*15*SIN(ALPHA)*DY;
   X1 := X2 + HEIGHT*5*SIN(ALPHA)*DX;
   Y1 := Y2 - HEIGHT*5*COS(ALPHA)*DY;
   X3 := X2 - HEIGHT*5*SIN(ALPHA)*DX;
   Y3 := Y2 + HEIGHT*5*COS(ALPHA)*DY;
J06YMF(2);
J06YCF(X2,Y2);
J06YMF(3);
   J06YCF(UP,VP);
   J06YCF(X1,Y1);
   J06YCF(X3,Y3);
   J06YCF(UP,VP);
   J06YAF(UP,VP);
```

```
END;   (*  end of subroutine ARROWHEAD  *)

%PAGE;
(*=================================================================*)
(*                                                                 *)
(*                            XYANAL                               *)
(*                                                                 *)
(*=================================================================*)
(*                                                                 *)
(*  XYANAL rangschikt de X en Y coordinaten van de vortexkernen   *)
(*  oplopend van klein naar groot. De coordinaten kunnen nu,       *)
(*  onafhankelijk van hun plaats in de inlees-file, worden in-     *)
(*  gelezen. Voorts bepaald het programma de maximale Y-waarde,    *)
(*  welke is ingelezen ivm dimensionering van de plot.             *)
(*                                                                 *)
(*****************************************************************)

PROCEDURE XYANAL (     EPS   : REAL;
                   VAR IMAX  : INTEGER;
                   VAR IMIN  : INTEGER;
                   VAR JJMAX : INTEGER;
                   VAR JJMIN : INTEGER;
                   VAR NJ    : IA01100;
                       NVORT : INTEGER;
                   VAR U     : RA01100;
                   VAR V     : RA01100;
                   VAR X     : RA01100;
                   VAR Y     : RA01100;
                   VAR YYMIN : REAL;
                   VAR YYMAX : REAL);

VAR Z1                     : RA01100;
    WISSEL                 : BOOLEAN;
    NHULP, NV, I, I3,
    J, JJENAMAX, NJMAX     : INTEGER;
    YYENAMAX               : REAL;
    JJ                     : IAR01100;

BEGIN
(*****************************************************************)
(*                                                                 *)
(*  Het rangschikken van de coordinaten van de                     *)
(*  vortexkernen.  Resultaat: laagste X-waarde voorop              *)
(*  oplopend tot maximum, gelijkertijd worden de                   *)
(*  snelheids componenten in de juiste volgorde geplaatst.         *)
(*                                                                 *)
(*****************************************************************)
  WISSEL := TRUE;
  FOR I := 0 TO NVORT DO JJ(.I.) := I;
  NV := NVORT -1;
  WHILE WISSEL DO
      BEGIN
      WISSEL := FALSE;
      FOR I := 0 TO NV DO
          BEGIN
```

```
        IF X(.JJ(.I.).) > X(.JJ(.I+1.).) THEN
            BEGIN
            NHULP := JJ(.I.);
            JJ(.I.) := JJ(.I+1.);
            JJ(.I+1.) := NHULP;
            WISSEL := TRUE;
            END;
        END;
    NV := NV-1;
    END;
FOR I := 0 TO NVORT DO    Z1(.I.) := X(.I.);
FOR I := 0 TO NVORT DO    X(.I.) := Z1(.JJ(.I.).);
FOR I := 0 TO NVORT DO    Z1(.I.) := Y(.I.);
FOR I := 0 TO NVORT DO    Y(.I.) := Z1(.JJ(.I.).);
FOR I := 0 TO NVORT DO    Z1(.I.) := V(.I.);
FOR I := 0 TO NVORT DO    V(.I.) := Z1(.JJ(.I.).);
FOR I := 0 TO NVORT DO    Z1(.I.) := U(.I.);
FOR I := 0 TO NVORT DO    U(.I.) := Z1(.JJ(.I.).);
(*******************************************************************)
(*                                                                 *)
(*  Samen rapen van 2 of meer X coordinaten van een vortexkern,   *)
(*  indien zij minder dan eps (MM) van elkaar verwijderd zijn.    *)
(*  De Y-assen zullen nu samen vallen.                            *)
(*                                                                 *)
(*  De Y coordinaten worden gerangschikt, indien zij dezelfde     *)
(*  X waarde hebben.                                              *)
(*                                                                 *)
(*******************************************************************)
NJMAX := NVORT;
FOR I := 0 TO NJMAX DO    NJ(.I.) :=1;
I := -1;
J := 0;
REPEAT
    BEGIN
    I := I + 1;
    J := J + 1;
    IF (ABS(X(.I.) - X(.I+1.)) < EPS) THEN
        BEGIN
        J := J - 1;
        NJ(.J.) := NJ(.J.) + 1;
        NJMAX := NJMAX - 1;
        IF (Y(.I+1.) < Y(.I.)) THEN
            BEGIN
            HULP1 := Y(.I.);
            Y(.I.) := Y(.I+1.);
            Y(.I+1.) := HULP1;
            HULP1 := U(.I.);
            U(.I.) := U(.I+1.);
            U(.I+1.) := HULP1;
            HULP1 := V(.I.);
            V(.I.) := V(.I+1.);
            V(.I+1.) := HULP1;
            END;  (*  end of if Y(I+1) < Y(I) situation   *)
        END;  (*  end of if .. < EPS  situation   *)
    END;  (*  end of repeat-blok  *)
```

```
    UNTIL I = (NVORT - 1);
(*******************************************************************)
(*                                                                 *)
(*    Nu worden de overtollige X-waarden geskipt.                  *)
(*                                                                 *)
(*******************************************************************)
  I := -1;
  FOR J := 0 TO NJMAX DO
      BEGIN
      I := I + 1;
      X(.J.) := X(.I.);
      IF (NJ(.J.) > 1)THEN    I := I + NJ(.J.) - 1;
      END;
(*******************************************************************)
(*                                                                 *)
(*    Nu worden de uiterste waarden in de Y-array bepaald.         *)
(*                                                                 *)
(*******************************************************************)
  IF (Y(.0.) < Y(.1.)) THEN
      BEGIN
      YYENAMAX := Y(.0.);
      YYMAX := Y(.1.);
      JJMAX := 1;
      JJENAMAX := 0;
      END;
  IF (Y(.0.) >= Y(.1.)) THEN
      BEGIN
      YYENAMAX := Y(.1.);
      YYMAX := Y(.0.);
      JJMAX := 0;
      JJENAMAX := 1;
      END;
  YYMIN := Y(.0.);
  JJMIN := 0;
  FOR I := 0 TO NVORT DO
      BEGIN
      IF (Y(.I.) > YYENAMAX) AND (Y(.I.) < YYMAX) THEN
          BEGIN
          YYENAMAX := Y(.I.);
          JJENAMAX := I;
          END;
      IF (Y(.I.) < YYMIN) THEN
          BEGIN
          YYMIN := Y(.I.);
          JJMIN := I;
          END;
      IF (Y(.I.) > YYMAX) THEN
          BEGIN
          YYENAMAX := YYMAX;
          JJENAMAX := JJMAX;
          YYMAX := Y(.I.);
          JJMAX := I;
          END;
      END;  (*  end of for I.. iteration  *)
  IMAX := NJMAX;
```

```
  IMIN := 0;
  WRITELN('jmax (=nvort) = ', NVORT:7,' imax = ', IMAX:7,' njmax = ',
           NJMAX:7 );
  WRITE('jjenamax = ', JJENAMAX:7, ' yyenamax = ', YYENAMAX:7);
  WRITE(' jjmax = ', JJMAX:7, ' yymax = ', YYMAX:7);
  WRITELN;
END;

%PAGE;
(*==================================================================*)
(*                                                                  *)
(*                           PLOTARROWS                             *)
(*                                                                  *)
(*==================================================================*)
(*                                                                  *)
(*  PROCEDURE PLOTARROWS PLOTS THE VECTORS (COMPONENTS U AND V)     *)
(*                                                                  *)
(*                                                                  *)
(*  VARIABLES :                                                     *)
(*                                                                  *)
(*  ANG       real angle of the arrow, with the positive X-axis     *)
(*            (horizontal axis of the plot), in radials             *)
(*  ANGLE     tangens ANG                                           *)
(*  IAR       contains the I values which are plotted               *)
(*  IMAX      upper level of number of gridpoints in X-direction    *)
(*  INCI      increment in X-direction                              *)
(*  JMAX      upper level of number of gridpoints in Y-direction    *)
(*  PHI       help-angle to get ANG in the 1st or 4th quarter,      *)
(*            dependent on the sign of SINUS or COSINUS;            *)
(*            this is necessary for the calculations with the       *)
(*            SINUS and COSINUS to get the shape of the arrow-      *)
(*            head                                                  *)
(*  U         U-velocity                                            *)
(*  V         V-velocity                                            *)
(*  UP, VP    coordinates of point in the middle of the baseline    *)
(*            of the triangle which the arrowhead is                *)
(*  XPLOT     plot coordinate in X-direction                        *)
(*  YPLOT     plot coordinate in Y-direction                        *)
(*                                                                  *)
(********************************************************************)

PROCEDURE PLOTARROWS(NXN      : INTEGER;
                     FAC      : REAL;
                     DX, DY   : REAL );

VAR I, II, INDEX, J, JJ, K,
    KLEUR,NYN                  : INTEGER;
    ANG, ANGLE, PHI, SHULPX,
    SHULPY, UP, VP, HEIGHT     : REAL;

BEGIN
  HEIGHT := 0.08 / 5;
  KLEUR := 2;
  J06YMF (KLEUR);
  INDEX := 0;
```

```
   FOR II := 0 TO NXN DO
       BEGIN
       I := IAR(.II.);
       IF (I <> II) AND (II >= 1) THEN
           BEGIN
           FOR K := (IAR(.II - 1.) + 1) TO (I - 1) DO
                               INDEX := INDEX + NJ(.K.);
           END;
       NYN := INDEX;
       FOR J := NYN TO (NYN + NJ(.I.) - 1) DO
           BEGIN
           INDEX := INDEX + 1;
           J06YAF(XPLOT(.I.), YPLOT(.J.));
           UP := U(.J.) / FAC + XPLOT(.I.);
           VP := - V(.J.) / FAC + YPLOT(.J.);
           SHULPX :=  UP - XPLOT(.I.) ;
           SHULPY :=  VP - YPLOT(.J.) ;
(******************************************************************)
(*                                                                *)
(*             Draw arrow                                         *)
(*                                                                *)
(******************************************************************)
           IF (SHULPX <> 0.0) THEN
             BEGIN
             ANGLE := (SHULPY*DX) / (SHULPX*DY) ;
             ANG := ARCTAN (ANGLE);
             IF SHULPX < 0 THEN       PHI := 1.0*PI
             ELSE
                 PHI := 0;
                 ARROWHEAD (UP, VP, HEIGHT, ANG+PHI, DX, DY );
                 END;
           IF SHULPX = 0.0 THEN
             BEGIN
             IF SHULPY<0 THEN ARROWHEAD (UP, VP, HEIGHT,-0.5*PI, DX, DY);
             IF SHULPY>0 THEN ARROWHEAD (UP, VP, HEIGHT, 0.5*PI, DX, DY);
(**********************************************************************)
(*                                                                    *)
(*   This is necessary because of ARCTAN(SHULPX/0) does not exist,    *)
(*   but we know that then the angle is 0.5*PI or -0.5*PI.            *)
(*                                                                    *)
(**********************************************************************)
             END;   (*  end of SHULPX = 0 situation  *)
          END;   (*  end of J iteration  *)
     END;   (*  end of FOR II := 0 TO NXN situation  *)
END;   (*  end of routine PLOTARROW  *)

%PAGE;
(*=============================================================*)
(*                                                             *)
(*                         PLOTSCALE                           *)
(*                                                             *)
(*=============================================================*)
(*                                                             *)
(*  Procedure PLOTSCALE scales a variable in                   *)
(*  X-direction to ensure that it will lie between two grid-   *)
```

```
(*  points which are plotted                                          *)
(*                                                                    *)
(*  VARIABLES :                                                       *)
(*                                                                    *)
(*  FAC     scale factor between variable and grid size               *)
(*  IAR     gives I points which are used in the plot                 *)
(*  IMAX    gives upper level of total number of X-points             *)
(*  JMAX    gives upper level of total number of Y-points             *)
(*  NJ      gives upper level of J as function of I                   *)
(*  NXN     upper level of number of X-points which are used in       *)
(*          the plot                                                  *)
(*  NYN     NJ(I) -1                                                  *)
(*                                                                    *)
(*********************************************************************)

PROCEDURE PLOTSCALE(NXN      : INTEGER;
               VAR   FAC      : REAL);

VAR I,II,I1,I2,INDEX,J,K,NYN : INTEGER;
    PHIP                     : REAL;
    SHULP                    : SHORTREAL;

BEGIN
  INDEX := 1;
  FOR II := 1 TO NXN-1 DO
      BEGIN
      I  := IAR(.II.);
      IF (I <> II) THEN
          BEGIN
          FOR K := (IAR(.II-1.) + 1) TO (I-1) DO
                                   INDEX := INDEX + NJ(.K.);
          END;
      I1 := IAR(.II+1.);
      I2 := IAR(.II-1.);
      NYN := INDEX;
      FOR J := NYN TO (NYN + NJ(.I.) - 1) DO
          BEGIN
          PHIP := U(.INDEX.);
          IF PHIP >= 0 THEN
              BEGIN
              SHULP := XPLOT(.I1.) - XPLOT(.I.);
              IF ( (PHIP/FAC) > SHULP) AND (SHULP <> 0.0) THEN
                                     FAC := 1.3 * PHIP / SHULP ;
              END;
          IF PHIP < 0 THEN
            BEGIN
            SHULP := XPLOT(.I.) - XPLOT(.I2.);
            IF (ABS(PHIP/FAC) > SHULP) AND (SHULP <> 0.0) THEN
                                    FAC := 1.3 * ABS(PHIP / SHULP) ;
            END;  (*  end of IF PHIP < 0 condition  *)
          INDEX := INDEX + 1;
        END;  (*  end of J iteration  *)
    END;  (*  end of II iteration  *)
END;  (*  end of subroutine PLOTSCALE  *)
```

```
%PAGE;
(*=================================================================*)
(*                                                                 *)
(*                          PLOTCHAN                               *)
(*                                                                 *)
(*=================================================================*)
(*                                                                 *)
(*  Procedure PLOTCHAN scales the X- and Y-coordinates and plots   *)
(*  the geometry of the channel and an outside border (optional).  *)
(*                                                                 *)
(*  The plotcoordinates XPLOT and YPLOT are also calculated.       *)
(*                                                                 *)
(*  Because of the definition of the viewport, the maximum and     *)
(*  minimum values in X- and Y-direction are calculated.           *)
(*                                                                 *)
(*  Factors DX and DY are calculated for the correct shape of the  *)
(*  arrows and the correct WIDTH and HEIGHT of the text which has  *)
(*  to be plotted in procedure PLTEKST.                            *)
(*                                                                 *)
(*  DX, DY        =  scaling factors; note that the scales of the  *)
(*                   X- and Y-axis are not the same                *)
(*  DATUM         =  stringvariable in which date and time are     *)
(*                   in                                            *)
(*  MAXX, MINX    =  maximum and minimum values in X-direction     *)
(*  MAXY, MINY    =  maximum and minimum values in Y-direction     *)
(*  XPLOT, YPLOT =  plotcoordinates                                *)
(*                                                                 *)
(*******************************************************************)

PROCEDURE PLOTCHAN (VAR DX         : REAL;
                    VAR DY         : REAL;
                        EPS        : REAL;
                        IMAX       : INTEGER;
                        IMIN       : INTEGER;
                        JJMAX      : INTEGER;
                        JJMIN      : INTEGER;
                    VAR LENGTH     : REAL;
                    VAR MAXX       : REAL;
                    VAR MAXY       : REAL;
                    VAR MINX       : REAL;
                    VAR MINY       : REAL;
                        NVORT      : INTEGER;
                    VAR RATIO      : REAL;
                        STEPRATIO  : REAL;
                        X          : RA01100;
                    VAR XPLOT      : SRA01100;
                    VAR YPLOT      : SRA01100);

VAR DATE, TIME                      : ALFA;
    DATUM                           : STRING(30);
    I, J,KLEUR                      : INTEGER;
    HELP,HULP                       : REAL;
    MIDLIN                          : SRA0500;
    SHULP                           : SHORTREAL;
```

```
BEGIN
(*******************************************************************)
(*                                                                 *)
(*  Calculate plot coordinates                                     *)
(*                                                                 *)
(*******************************************************************)
  FOR J := 0 TO NVORT DO
      BEGIN
      YPLOT(.J.) := ROUND(-Y(.J.) *1000/EPS)/(1000/EPS);
      END;
  FOR I := 0 TO IMAX DO
      BEGIN
      XPLOT(.I.) := ROUND( X(.I.) * 1000/EPS) /(1000/EPS);
      END;
(*******************************************************************)
(*                                                                 *)
(*  Draw geometry                                                  *)
(*                                                                 *)
(*******************************************************************)
  PLOTS (1, 15);
  LENGTH := ABS(X(.IMAX.) - 0);
  MAXX := XPLOT(.IMAX.) + 0.20*(XPLOT(.IMAX.)-XPLOT(.IMIN.));
  MINX := XPLOT(.IMIN.) - 0.01*(XPLOT(.IMAX.)-XPLOT(.IMIN.));
  MAXY := YPLOT(.JJMIN.) + 0.12*ABS(YPLOT(.JJMAX.)-YPLOT(.JJMIN.));
  MINY := YPLOT(.JJMAX.) - 0.01*ABS(YPLOT(.JJMAX.)-YPLOT(.JJMIN.));
  IF MAXX = MINX THEN WRITELN ('PLOTCHAN : FOUTE BOEL MET MAXX!!');
  DX := 0.01 * (MAXX - MINX);
  DY := 0.01 * (MAXY - MINY);
  RATIO := DX/DY;
  J06WBF(MINX, MAXX, MINY, MAXY, 1);
  J06WCF(0.0, 1.0, 0.0, 1.0);
  DATETIME(DATE,TIME);
  WRITESTR(DATUM,DATE,'    ',TIME);
  DRAWTEKST(DATUM,'    ');
  HULP := ABS(XPLOT(.IMAX.) - XPLOT(.IMIN.));
  KLEUR := 1;
(*******************************************************************)
(*                                                                 *)
(*      Draw channel geometry        (optional)                    *)
(*      ---------------------                                      *)
(*  If you want the geometry of the channel than change next       *)
(*  line's "KLEUR>?" in "KLEUR>0".                                 *)
(*  In this case in HULPPLOT the point (0.0, 0.0) should be        *)
(*  added !!                                                       *)
(*                                                                 *)
(*******************************************************************)
  IF (KLEUR>0) THEN
     BEGIN
     HELP := HULP/20;
     FOR I := 0 TO 20 DO    MIDLIN(.I.) := MAXX - I*HELP;
     FOR J := 1 TO 10 DO
         BEGIN
         I := 2*J-2;
         J06YAF(MIDLIN(.I.), 0.0);
         J06YCF(MIDLIN(.I+1.), 0.0);
```

```
          J06YAF((MIDLIN(.I+1.)-0.45*HELP), 0.0);
          J06YCF((MIDLIN(.I+1.)-0.55*HELP), 0.0);
          END;  (* end of J iteration  *)
      SHULP := -ABS(YPLOT(.JJMAX.)-0) * (1 - STEPRATIO);
      J06YCF(XPLOT(.0.), 0.0);
      J06YAF(XPLOT(.0.), SHULP);
      J06YCF(0.0, SHULP);
      J06YCF(0.0, YPLOT(.JJMAX.));
      J06YCF(MAXX, YPLOT(.JJMAX.));
   END;  (*  end of Draw channel geometry  *)
(*********************************************************************)
(*                                                                   *)
(*     Draw outside border    (optional)                             *)
(*     -------------------                                           *)
(*  If you want the outside border then change next line's           *)
(*  "KLEUR>?" in "KLEUR>0".                                          *)
(*  In this case in HULPPLOT the point (0.0, 0.0) should be added    *)
(*                                                                   *)
(*********************************************************************)
   IF (KLEUR>0) THEN
      BEGIN
(*  Starting in the lower left corner                                *)
      J06YAF( MINX - 15*DX, MINY - 15*DY );
      J06YCF( MAXX + 20*DX, MINY - 15*DY );
      J06YCF( MAXX + 20*DX, MAXY + 15*DY );
      J06YCF( MINX - 15*DX, MAXY + 15*DY );
      J06YCF( MINX - 15*DX, MINY - 15*DY );
      END;  (*  end of Draw outside border  *)
(*********************************************************************)
(*                                                                   *)
(*     Draw object SOLID1     (optional)                             *)
(*     ------------------                                            *)
(*  If you want this object drawn then change next line's            *)
(*  "KLEUR>?" into "KLEUR>0".                                        *)
(*                                                                   *)
(*********************************************************************)
   IF (KLEUR>0) THEN
      BEGIN
      MIDLIN(.1.) := ROUND(-4*1000/EPS) / (1000/EPS);
      MIDLIN(.2.) := ROUND(-2*1000/EPS) / (1000/EPS);
      J06YAF(1, MIDLIN(.1.) );
      J06YCF(1, MIDLIN(.2.) );
      FOR I := 1 TO 200 DO BEGIN
          HULP := -1 + (I-1)*2 / (200-1);
          MIDLIN(.1.) := - 0.2*(1 - HULP*HULP);
          MIDLIN(.1.) := ROUND( (MIDLIN(.1.)+1)*1000/EPS) / (1000/EPS);
          MIDLIN(.2.) := ROUND(-(HULP+3)*1000/EPS) / (1000/EPS);
          J06YCF(MIDLIN(.1.), MIDLIN(.2.) );
          END;  (* end of I iteration  *)
      END;  (*  end of Draw object SOLID1  *)
END;  (*  end of subroutine PLOTCHAN  *)

%PAGE;
(*================================================================*)
(*                                                                *)
```

```
(*                        PLOTGRID                              *)
(*                                                              *)
(*==============================================================*)
(*                                                              *)
(*  Procedure PLOTGRID draws a grid-system.                     *)
(*                                                              *)
(****************************************************************)

PROCEDURE PLOTGRID(IMAX      : INTEGER;
                   IMIN      : INTEGER;
                   JJMAX     : INTEGER;
                   JJMIN     : INTEGER);

VAR I,J,kleur        : INTEGER;
    H1,H2            : REAL;
    PUNT, I1         : SHORTREAL;

BEGIN
  KLEUR := 3;
  H1 := ABS(XPLOT(.IMAX.) - XPLOT(.IMIN.));
  H2 := ABS(YPLOT(.JJMAX.) - YPLOT(.JJMIN.));
  J06YMF (KLEUR);
(****************************************************************)
(*                                                              *)
(*     Plot the vertical grid lines                             *)
(*                                                              *)
(****************************************************************)
  FOR I := 1 TO 9 DO
      BEGIN
      I1 := XPLOT(.IMIN.) + I* H1/10;
      J06YAF(I1, YPLOT(.JJMIN.));
      J06YCF(I1, YPLOT(.JJMAX.));
      END;
(****************************************************************)
(*                                                              *)
(*     Plot the horizontal grid lines                           *)
(*                                                              *)
(****************************************************************)
  FOR J := 1 TO 9 DO
      BEGIN
      PUNT := XPLOT(.IMIN.) - (H2/10) * J;
      J06YAF(XPLOT(.IMIN.), PUNT);
      J06YCF(XPLOT(.IMAX.),PUNT);
      END;  (*  end of J iteration  *)
END;  (*  end of subroutine PLOTGRID  *)

%PAGE;
(*==============================================================*)
(*                                                              *)
(*                        FILLAR                                *)
(*                                                              *)
(*==============================================================*)
(*                                                              *)
(*  Procedure FILLAR fills the array which contains             *)
(*  the points in X-direction which are used when making        *)
```

```
(*  a plot.                                                    *)
(*                                                             *)
(*  VARIABLES :                                                *)
(*                                                             *)
(*  IAR      gives I points which are used in the plot         *)
(*           array IAR has index running from 0 to NXN + 1     *)
(*  IMAX     upper level of grid points in X-direction         *)
(*  INCI     increment for points in X-direction               *)
(*  NXN      maximum number of points in X-direction which are *)
(*           used                                              *)
(*                                                             *)
(***************************************************************)

PROCEDURE FILLAR (VAR IAR    : IA01100;
                      INCI   : INTEGER;
                      IMAX   : INTEGER;
                  VAR NXN    : INTEGER);

VAR I : INTEGER;

BEGIN
  NXN := ROUND(IMAX / INCI);
  IAR(.0.) := 0;
  FOR I := 1 TO NXN+1 DO  IAR(.I.) := 1+(I-1)*INCI;
END;

%PAGE;
(*==============================================================*)
(*                                                              *)
(*              PROCDIR         (procedure director)            *)
(*                                                              *)
(*==============================================================*)
(*                                                              *)
(*  Procedure PROCDIR is developed to plot the velocity-field   *)
(*                                                              *)
(*  Fields that should be filled before "PROCDIR" is run:       *)
(*                                                              *)
(*  IMAX       natural numbers larger than 2                    *)
(*  JMAX       natural numbers larger than 2                    *)
(*  INCI       1,2 or 3                                         *)
(*  I          this index runs: 0,1,..,IMAX                     *)
(*  J          this index runs: 0,1,..,JMAX                     *)
(*  X(I)       array of axial coordinates of points where the   *)
(*             velocity field U(I) and V(J) is given            *)
(*  Y(J)       array of radial coordinates of points where the  *)
(*             velocity field is given                          *)
(*  U(I)       axial component of velocity                      *)
(*  V(J)       vertical component of velocity                   *)
(*                                                              *)
(*                      0.0   + X                               *)
(*              __ . __ .------>__ . __ . __                    *)
(*                      .                                       *)
(*              ________|                                       *)
(*                      |                                       *)
(*                 + Y  V___________________                    *)
```

```
(*                                                                        *)
(*  NJ(I)    is an array of total number of "grid" points at              *)
(*           axial location I                                             *)
(*                                                                        *)
(*  VARIABLES THAT ARE CALCULATED :                                       *)
(*  -----------------------------                                         *)
(*                                                                        *)
(*  FAC        this variable gives the height of the tail of the          *)
(*             plotted arrow, when you make FAC smaller the tail          *)
(*             of the arrow will be bigger.                               *)
(*             this parameter can also be adapted ("PLOTSCALE")           *)
(*  IAR        ("FILLAR")                                                 *)
(*  NXN        ("FILLAR")                                                 *)
(*                                                                        *)
(*  VARIABLES :                                                           *)
(*  ---------                                                             *)
(*                                                                        *)
(*  HULPT      string variable, in which TEXT and a variable can          *)
(*             be read in, the use of this string is only deman-          *)
(*             ded if you want to plot a variable                         *)
(*  IAR        contains I values which are used in the plot               *)
(*  IMAX       number of grid points in X-direction                       *)
(*  JMAX       number of grid points in Y-direction                       *)
(*  INCI       increment of I, if INCI > 1 then not all of the            *)
(*             X-columns are used for plotting                            *)
(*  NXN        total number of I points in IAR                            *)
(*  STANSWER   startpoint, where the plotter starts with plotting         *)
(*             some of the variables                                      *)
(*  U          velocity in X-direction                                    *)
(*  V          velocity in Y-direction                                    *)
(*  XST        startpoint in X-direction, where the plotter               *)
(*             starts with plotting the text in the plot                  *)
(*  YST        startpoint in Y-direction                                  *)
(*                                                                        *)
(*  PROCEDURES USED :                                                     *)
(*  ---------------                                                       *)
(*                                                                        *)
(*  FILLAR         fills the array IAR, as function of the values         *)
(*                 of IMAX and INCI                                       *)
(*  PLOTARROWS      PLOTS UV-ARROW                                        *)
(*  PLOTCHAN       calculates and scales X- and Y-coordinates, and        *)
(*                 plots geometry of channel                              *)
(*  PLOTSCALE      scales the velocity between two neighbouring           *)
(*                 gridpoints                                             *)
(*  XYANAL         analyses X and Y                                       *)
(*                                                                        *)
(**************************************************************************)

PROCEDURE PROCDIR(VAR EPS          : REAL;
                      INCI         : INTEGER;
                      NVORT        : INTEGER;
                      STEPH        : REAL;
                      STEPRATIO    : REAL;
                      X            : RA01100);
```

```
VAR AA, BB, CC, IMAX, IMIN, JJMAX,
    JJMIN, KLEUR, NXN                   : INTEGER;

    FAC, STANSWER, XST, YST,
    YYMIN, YYMAX, DX, DY,
    MAXX, MAXY, MINX, MINY,
    LENGTH, RATIO, SHULP1, SHULP2  : REAL;
    NAME                                : STRTEK;
    HULPT                               : STRING(80);

BEGIN
(******************************************************************)
(*                                                                *)
(*  Draw large overall plot.                                      *)
(*                                                                *)
(******************************************************************)
  RATIO := 1;
  KLEUR := 2;
  XYANAL (EPS, IMAX, IMIN, JJMAX, JJMIN, NJ, NVORT, U, V, X, Y, YYMAX,
          YYMIN);
  PLOTCHAN (DX, DY, EPS, IMAX, IMIN, JJMAX, JJMIN, LENGTH, MAXX, MAXY,
            MINX, MINY, NVORT, RATIO, STEPRATIO, X, XPLOT, YPLOT);
(*PLOTGRID (IMAX,IMIN,JJMAX,JJMIN); *)
(*WRITELN ('einde PLOTGRID');       *)
  FILLAR (IAR, INCI, IMAX, NXN);
  FAC := 0.001;
  PLOTSCALE (NXN, FAC);
(*  The smaller FAC, the greater the arrows will be                     *)
  IF FAC = 0.0 THEN FAC := 1;
  IF (TUBERADIUS = 6) THEN FAC := FAC/100;
  PLOTARROWS (NXN, FAC, DX, DY );
(*WRITELN ('einde PLOTARROWS');     *)
  J06YMF (KLEUR);
(**********************************************************************)
(*                                                                    *)
(*  All the text is plotted on a specific place "XST","YST"           *)
(*  which is done to get the tekst on the same place by different     *)
(*  heights of different plots.                                       *)
(*                                                                    *)
(*  In some of the following statements the mark "=" and the given   *)
(*  answers are also plotted on a specified place named "STANSWER",   *)
(*  "YST". This is done to let the 'answers' begin on the same        *)
(*  vertical.                                                         *)
(*                                                                    *)
(**********************************************************************)
  XST := MINX + 0.1*(MAXX-MINX);
  STANSWER := MINX + 0.5*(MAXX-MINX);
  YST := MAXY + 8.9*DY;
  PLTEKST('PROGRAMME VORTEX', XST, YST, DX, DY );
  IF KLEUR>0 THEN
    BEGIN
     YST := MAXY + 5.7*DY;
     IF STEPH<1.0E6 THEN SHULP1:=ROUND(STEPH*1000)/1000
                    ELSE SHULP1:=0.0;
    PLTEKST('Stepheight ',XST, YST, DX, DY );
```

```
    WRITESTR(HULPT,'= ', SHULP1:8:2);
    PLTEKST(HULPT, STANSWER, YST, DX, DY );
     YST := MAXY + 4.0*DY;
     IF TUBERADIUS < 1.0E6 THEN SHULP1 := ROUND(TUBERADIUS * 100) / 100
                          ELSE SHULP1:= 0.0;
    PLTEKST('Channel radius',XST, YST, DX, DY );
    WRITESTR(HULPT,'= ', SHULP1:8:2);
    PLTEKST(HULPT, STANSWER, YST, DX, DY );
     IF LENGTH<1.0E6 THEN
       BEGIN
       YST := MAXY + 2.2*DY;
       PLTEKST('Max distance from inlet',XST, YST, DX, DY );
       WRITESTR(HULPT,'= ', LENGTH:8:2);
       PLTEKST(HULPT, STANSWER, YST, DX, DY );
       END;
     YST := MAXY + 0.5*DY;
     IF FAC < 1.0E6 THEN SHULP1:=ROUND(FAC*100) / 100  ELSE SHULP1:=0;
    PLTEKST('Scalefactor between velocity and X-lenght',
                      XST, YST, DX, DY );
    WRITESTR(HULPT,'= ', SHULP1:9:3);
    PLTEKST(HULPT, STANSWER, YST, DX, DY );
     YST := MAXY - 1.3*DY;
     SHULP1 := NVORT -1;
    PLTEKST('Number of vortices',XST, YST, DX, DY );
    WRITESTR(HULPT,'= ', SHULP1:8:2);
    PLTEKST(HULPT, STANSWER, YST, DX, DY );
     YST := MAXY - 3.1*DY;
    PLTEKST('Boundary coordinates',XST, YST, DX, DY );
    WRITESTR(HULPT,'= (',XPLOT(.IMIN.):6:2,',',YPLOT(.JJMIN.):6:2,
            ')      (',XPLOT(.IMAX.):6:2,',',YPLOT(.JJMIN.):6:2,')');
    PLTEKST(HULPT, STANSWER, YST, DX, DY );
     YST := MAXY - 4.9*DY;
    WRITESTR(HULPT,'  (',XPLOT(.IMIN.):6:2,',',YPLOT(.JJMAX.):6:2,
            ')      (',XPLOT(.IMAX.):6:2,',',YPLOT(.JJMAX.):6:2,')');
    PLTEKST(HULPT, STANSWER, YST, DX, DY );
     YST := MAXY - 6.6*DY;
    IF RATIO<1.0E6 THEN
       BEGIN
       WRITESTR(HULPT,'1 cm X corresponds to ',RATIO:4:2,' cm Y ');
       PLTEKST(HULPT, XST, YST, DX, DY );
       END;
     YST := MAXY - 8.4*DY;
     SHULP1:= N-1;
     IF DELT<1.0E6 THEN SHULP2 := ROUND(DELT * 1000 * 100) / 100
                                      ELSE SHULP2:=0.0;
    WRITESTR(HULPT,'Situation after ',SHULP1:4:2,' steps of ',
             SHULP2:5:2,' ms ');
    PLTEKST(HULPT, XST, YST, DX, DY );
  END;  (*  end of KLEUR condition   *)
(*WRITELN ('einde PROCDIR');          *)
  J06WZF;
END;  (* end of subroutine PROCDIR  *)

%PAGE;
(*=============================================================*)
```

```
(*                                                                 *)
(*                              HULPPLOT                           *)
(*                                                                 *)
(*=================================================================*)
(*                                                                 *)
(* VARIABLES THAT SHOULD BE FILLED:                                *)
(*                                                                 *)
(* X(I)  array of axial coordinates of points where the velocity *)
(*       field U,V is given    index I : 0 - NVORT                 *)
(* Y(I)  array of radial coordinates of points where the velocity*)
(* U(I)  array of axial components of velocity                     *)
(* V(I)  array of radial components of velocity                    *)
(* EXTRA boolean field to demand addition of points in HULPPLOT   *)
(* TUBERADIUS    inside radius of channel; must be > 1             *)
(*                                                                 *)
(* VARIABLES:                                                      *)
(*                                                                 *)
(* INCI  increment of I; if INCI > 1 then not all of the          *)
(*       X-columns are used for plotting                           *)
(* EPS   if the distance between two or more grid points is more *)
(*       than eps, the redundant X-coordinates will be skipped    *)
(* NVORT       number of vortices                                  *)
(* STEPRATIO   step from inlet to the wall                         *)
(*                                                                 *)
(*                                                                 *)
(*******************************************************************)
PROCEDURE HULPPLOT(        NVORT       : INTEGER;
                           TUBERADIUS  : REAL;
                       VAR XS          : RA01100;
                       VAR Y           : RA01100;
                       VAR U,V         : RA01100;
                           EXTRA       : BOOLEAN);

VAR INCI, I, NHULP          : INTEGER;
    SCP, EPS, STEPH         : REAL;
    STEPRATIO               : SHORTREAL;

BEGIN
NHULP := NVORT;

IF EXTRA THEN BEGIN
(*  we beginnen met een hulpblok speciaal voor VORTEX  *)
  FOR I := 1 TO NVORT DO
      BEGIN
      XS(.I.) := Z(.I.).RE + 1;
      Y(.I.) := Z(.I.).IM + 3;
      U(.I.) := VM(.I.).RE;
      V(.I.) := VM(.I.).IM;
      END;
  XS(.NVORT+1.) := WALL(.NWALL(.1.),1.).RE + 2;
  Y(.NVORT+1.) := WALL(.NWALL(.1.),1.).IM + 3;
  NHULP := ROUND(NWALL(.1.)/2);
  XS(.NVORT+2.) := WALL(.NHULP,1.).RE + 2;
  Y(.NVORT+2.) := WALL(.NHULP,1.).IM + 3;
  XS(.NVORT+3.) := WALL(.1,1.).RE + 2;
```

```
  Y(.NVORT+3.) := WALL(.1,1.).IM + 3;
  NHULP := ROUND(NWALL(.1.)/4);
  XS(.NVORT+4.) := WALL(.NHULP,1.).RE + 2;
  Y(.NVORT+4.) := WALL(.NHULP,1.).IM + 3;
  NHULP := ROUND(NWALL(.1.)/4);
  XS(.NVORT+5.) := 0;
  Y(.NVORT+5.) := WALL(.NHULP,1.).IM + 3;
  U(.NVORT+5.) := UINF.RE / 100;
  V(.NVORT+5.) := UINF.IM / 100;
  FOR I := 1 TO 4 DO
      BEGIN
      U(.NVORT+I.) := 0.00;
      V(.NVORT+I.) := 0.00;
      END;
  NHULP := NVORT + 5;
  END;  (*  einde van EXTRA en het hulpblok speciaal voor VORTEX  *)
(*******************************************************************)
  J06WAF;
  PI := 4 * ARCTAN (1.0);
  INCI := 1;
  EPS := 1E-04;
  STEPRATIO := 1 - 1/TUBERADIUS;
  STEPH := TUBERADIUS - 1;
  IF STEPH < 0 THEN WRITELN ('HULPPLOT : DEFINE TUBERADIUS!!!!');
  NHULP := NHULP + 1;
  XS(.NHULP.) := 0;
  Y(.NHULP.) := TUBERADIUS;
  U(.NHULP.) := 0;
  V(.NHULP.) := 0;
  NHULP := NHULP + 1;
  XS(.NHULP.) := 0.5 * TUBERADIUS;
  Y(.NHULP.) := TUBERADIUS;
  U(.NHULP.) := 0;
  V(.NHULP.) := 0;
(*  The point (0.0, 0.0) will be added to allow for drawing of a   *)
(*  centerline in PLOTCHAN                                          *)
  XS(.0.) := 0.0;
  Y(.0.) := 0.0;
  U(.0.) := 0.0;
  V(.0.) := 0.0;

  PROCDIR( EPS, INCI, NHULP, STEPH, STEPRATIO, XS );

  WRITELN('NVORT in PROCDIR is : ',NHULP);
END;  (*  end of subroutine HULPPLOT  *)

%PAGE;
(*=================================================================*)
(*                                                                 *)
(*                           POTFLOW                               *)
(*                                                                 *)
(*=================================================================*)
PROCEDURE POTFLOW;
(*******************************************************************)
(* Determination of flowlines at t=0 in the potential flow case  *)
```

```
(* of rectangular channel flow without vortices.                   *)
(*******************************************************************)
VAR HULP2,HULP3,P0,P,Q,Q1   : COMPLEX;
    MC,NS,PABS,PHI1,SCP     : REAL;
    IJ,I,J,NHULP            : INTEGER;
BEGIN
  PI := 4 * ARCTAN (1.0);
  FOR J := 1 TO 8 DO
      BEGIN
      WRITELN('J =',J);
      PHI1 := J * (15/360) * PI + PI/2;
      WRITELN('PHI1 =',PHI1);
      MC := COS(PHI1);
      NS := SIN(PHI1);
      WRITELN('MC =',MC,'   NS =',NS);
      P0 := CMPLX (MC,NS);
      WRITELN('P0.RE =',P0.RE,'    P0.IM =',P0.IM);
      FOR I := 1 TO 45 DO
          BEGIN
          IJ := I + (J-1)*45;
          PABS := 0.4 + (I / 45) * 5.5 * TUBERADIUS;
          CMULR(P0,PABS,P);
          CMUL(P,P,Q1);
          HULP2 := Q1;
          Q1.RE := Q1.RE - SQR(TUBERADIUS);
          HULP2.RE := HULP2.RE -1;
          CDIV(Q1,HULP2,Q1);
          CSQRT(Q1,Q);
          U(.IJ.) := Q.RE;
          V(.IJ.) := Q.IM;
          Q1 := Q;
          Q1.RE := Q1.RE + TUBERADIUS;
          CMULR(Q,-1,HULP2);
          HULP2.RE := HULP2.RE + TUBERADIUS;
          CDIV(Q1,HULP2,HULP2);
          CLOG(HULP2,Q1);
          CDIVR(Q1,TUBERADIUS,HULP3);
          Q1 := Q;
          Q1.RE := Q1.RE + 1;
          CMULR(Q,-1,HULP2);
          HULP2.RE := HULP2.RE + 1;
          CDIV(Q1,HULP2,HULP2);
          CLOG(HULP2,Q1);
          CSUB(Q1,HULP3,HULP2);
          CDIVR(HULP2,PI/TUBERADIUS,HULP2);
          SCP := P.RE;
          SCP := SCP/ABS(SCP);
          SCP := TUBERADIUS * SCP;
          XS(.IJ.) := HULP2.RE;
          Y(.IJ.) := HULP2.IM - SCP;
        END;
  NHULP := 361;
  XS(.0.) := 0;
  Y(.0.) := 0;
  U(.0.) := 0;
  V(.0.) := 0;
  END;
HULPPLOT(NHULP, TUBERADIUS, XS, Y, U, V, EXTRA);
END;
```