TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

Multi-layer injection moulding of oil-filled rubber dampers

Citation for published version (APA):
Swartjes, F. H. M. (1995). *Multi-layer injection moulding of oil-filled rubber dampers*. (DCT rapporten; Vol. 1995.096). Technische Universiteit Eindhoven.

Document status and date:
Published: 01/01/1995

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Download date: 08. Feb. 2024

Multi-layer injection moulding of oil-filled rubber dampers

F.H.M. Swartjes

WFW report 95.096

Supervised by: dr.ir. G.W.M. Peters and ir. L. Kleintjes
Faculty of Mechanical Engineering
Eindhoven University of Technology
July 1995

Contents

Summary

This study is on multicomponent injection moulding. The main goal of this work is the use of the multi-layer injection moulding process for producing oil-filled dampers. A relatively simple tool is created to determine the feasibility of the process itself. This tool has not been designed for a real production process, but only has to produce few dampers.
Numerical simulations of the filling of the mould have been carried out with the finite element package SEPRAN. At the inflow the rubber and oil are labelled. During filling of the mould, the labels are followed by solving a convective equation and the air is modelled at the boundaries with a slip condition. The air slips with the 'mean' velocity of the front. It is difficult and time-consuming to predict the right slipvelocities. The results are satisfactory when the velocities are chosen in the right range. The sequential injection gives better results than the combined injection in terms of the oildistribution in the product.
The injection of oil with the experiments is not controllable and therefore not reproduceble. Some of the problems are that the filling of the mould is asymmetrically and that, when the mould is filled completely, rubber is pressed into the oil feeding.

1 Introduction

1.1 Multicomponent injection moulding

The injection moulding process is a flexible production method to fabricate plastic parts in series, characterized by the ability to realize complex shaped, highly integrated products in small cycle times. The high flexibility of the injection moulding technique can be extended with the possibility of combining different materials within one produkt, each with its own specific properties. With a multicomponent (or multilayer) injection moulding technique a layered structure of two or more components can be realized within a thin-walled product. The geometry of the layers in the product depends mainly on the position of the gate, the geometry of the nozzle and the sequence of injection (simultaneously and/or sequential).
An injection moulding machine mainly consists of two parts: an injection and a clamping unit. Granulated polymer is supplied by a hopper to the feeding part of the rotating extruder screw. The rotating of the extruder screw causes the granulate to move through the heated extruder barrel. Heat, generated by the heating elements and by the screw rotation, causes the granulate to plasticize. When sufficient material has been plasticized, the screw acts as a piston and pushes the melt through a nozzle and runner system into the mould cavity. The clamping unit supports the two mould halves and prevents the mould from opening despite of the high pressure that occurs during the process.
The moulding cycle can be divided into three stages: the injection, the packing and holding, and the cooling stage. First the molten polymer is injected in the mould (injection stage). After complete filling of the cavity, extra material will be added to compensate for shrinkage (packing and holding stage). At the moment the gate is sealed, compensation for shrinkage is no longer possible and the cooling stage starts. When the temperature has decreased below the ejection temperature, the mould is opened and the produkt can be ejected.
In this study the multicomponent injection moulding process will be used to make oil-filled rubber dampers.

1.2 Oil-filled rubber dampers

Oil-filled rubber dampers are used for the suspension of the laser module inside an outdoor CD-player. The CD-mechanism is damped with four oil-filled rubber dampers. Such a damper has excellent performance in terms of damping and reduction characteristics at high frequencies (de Geus [1]).
The current used oil-filled rubber damper in the outdoor CD-players, has a high cost-prize due to the used production method. The process includes multiple production cycles. The first production cycle is the injection moulding of the rubber cup. In a second cycle the cup is filled with the silicon oil, and in the last cycle a rubber disc is glued on the cup to close the damper. Miniaturization of outdoor CD-applications demands that the oil-filled rubber damper will be reduced in volume substantially in near future.
The multilayer injection moulding process will be used to produce oil-filled rubber dampers. It is vital that the complete process is reproducible. Geometry variations, like the thickness of the oil layer, have large effects on the dynamic characteristics of the damper. To enhance flow visualization and to keep the mould within reasonable filling measures (not to small) the dimensions of the damper are made larger with respect to the current used oil-filled rubber dampers. The dynamic characteristics are of minor importance in this study. Further on, the multi-layer structure may not exist in the injection channels after complete filling of the mould.

The oil is a fluid at room temperature, so the damper would not be closed.
The ultimate goal of simulation of the injection moulding process, is to predict the process conditions given the required product properties. With a numerical simulation in the finite element package SEPRAN (Segal [3]) the time and duration of injection can be determined. By simulating the injection moulding process for different times and different durations of the injection of the oil, the process conditions for the experiments can be determined.
In chapter 2 of this report the experimental and numerical setup will be discussed. The materials used are described in chapter 3. The numerical results of two different kinds of injection are discussed in chapter 4. Also the results of the experiments are discussed in this chapter. Finally, chapter 5 summarizes the main results and gives suggestions for future research.

2 Experimental and numerical methods

2.1 Experimental setup

2.1.1 Dimensions of the product

The oil-filled rubber damper for the experimental setup is made somewhat larger as the conventional oil-filled rubber damper, as stated before (see Figure 1). The experiments are primarily

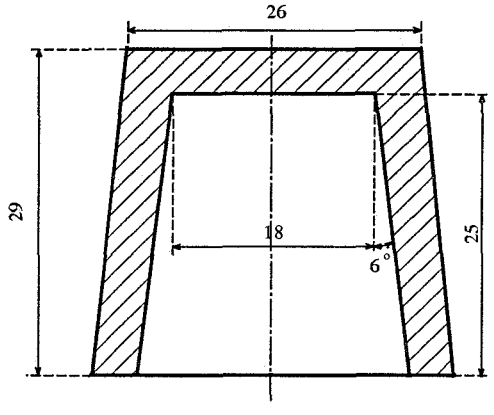

Figure 1: Dimensions of the product

focused on the feasibility of the process itself. Down sizing the product can be done when the process has been proven feasible. Figure 1 shows that the walls of the product are not perpendicular to the top of the product. This angle of 6.0^o is necessary to ease the remove of the product. The exact volume of the product is equal to 10.889 $[mm^3]$ (see Appendix A). The maximum injectionvolume of oil is about 5659 $[mm^3]$. With the experiments and the numerical calculations these values have been taken into account.

2.1.2 Equipment

The mould was not designed for a real production process, but for experimental convenience. A cross section of the mould is given in Figure 2. The needle stays during the process in the

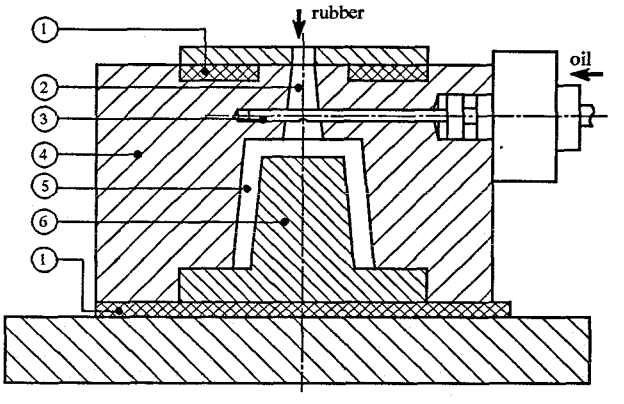

Figure 2: Cross section of the mould: 1. isolation; 2. injection channel; 3. needle; 4. upper part mould; 5. product; 6. lower part mould

position given in the figure. It has a small diameter in order to reduce the influence on the flow of the rubber. The outer diameter of the needle is 2.0 [mm]. The needle can rotate around his axis and can be fastened with a screw. The diameter of the hole in the needle is 0.5 [mm].
The complete setup is given in Figure 3. A heated buffer (3) is used to hold the material that

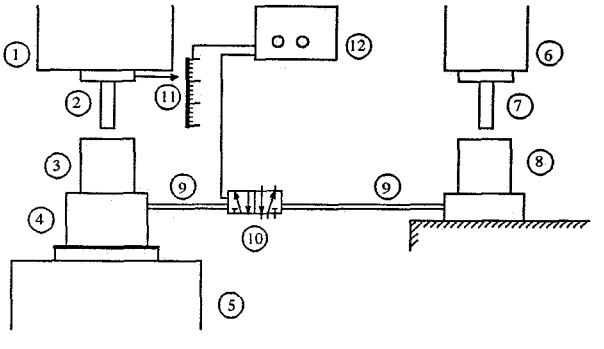

Figure 3: Complete setup: 1. upper part press; 2. piston; 3. buffer for rubber; 4. mould; 5. lower part press; 6. press; 7. piston; 8. buffer for oil; 9. pipe; 10. 5/2 valve; 11. position detector; 12. control unit

has to be injected. A piston (2) is used for the injection of the rubber. The piston-buffer like injection unit, is only capable of producing one damper, and has to be refilled after that. The buffer is heated with six heating elements, each heating element has a power of 200 [W]. The elements are, together with a thermocouple, connected with a supply box.
A combination of simultaneous and sequential injection (rubber - rubber and oil - rubber) was chosen at the experiments. The oil, which is also injected with a piston (7) - buffer (8) injection unit, is under constant pressure during the experiment. The oilbuffer is connected to the needle

with a pipe (9) and a valve (10). The valve and the position detector (11) of the rubber injection unit, are connected with a control unit (12) which is used to adjust the injection of oil. Two knobs with a scale [0-10], corresponding to the (maximum) stroke of the piston (50 [mm]), are used for that.

2.1.3 Working method

The rubber, Hytrel, is available in grains. The grains are pressed in plates at a temperature of $230°C$. First the grains are melted and after that the pressure is raised after each five minutes (the adjusted compressional forces are: 1, 2, 5, 10, and 20 [kN]). The plates are cut in small blocks from which the slices are made at a lathe. These slices fit in the buffer.
The different actions during the experiment are:

- clean the mould with a brass brush and acetone
- fill the buffer with rubber and (if necessary) fill the oilbuffer
- press the mould against the buffer
- position the needle. The needle has to be fastened and in the right direction before material is injected.
- press the piston against the rubber
- heat the buffer
- choose the begin and end of the injection of oil
- when the buffer is on temperature wait for fifteen minutes
- inject the rubber and oil
- cool down the mould with air or wait for a time and take out the product after the pressure has been taken of the mould

2.2 Numerical methods

2.2.1 Particle tracking

In order to realize the desired product geometry in the mold, knowledge of the time and duration of injection of the different components is required. Using the conservation of identity, the position of material injected at arbitrary moments can be determined. In this particle tracking technique material particles are defined by their unique identity (e.g. material, colour, place and time of injection). The particles are abstract, distinct points in the flow that have to be followed in time and space. By following the particles through the flow domain the material distribution is known.
Zoetelief [2] has shown that the conservation of identity method can be applied succesfully to track the material interfaces. Those interfaces that occur in the mould filling simulations, can be modelled with a jump of the material properties (e.g. η or ρ) at the inferfaces.
In this study, two different interfaces can be distinguished. First, there exists an interface between the rubber melt and the air during the filling. The second type of interface is the one

between the rubber and the oil. The filling of the mould is assumed to be isothermal, so there is no interface between a solid and liquid layer of rubber. Both interfaces are modelled with a discontinuity in the viscosity. They are approximated by continuous functions with a steep gradient. The maximum steepness is controlled by the local mesh size and the order of the element.
A major influence on the final particle distribution throughout the product is the so-called fountain flow. In flows with one or more free boundaries and a no-slip condition at the walls, fluid elements adjacent to the moving front experience this phenomenon. The fluid near the center moves at a higher speed than the local average speed across the channel. When the fluid reaches the front, it spreads towards the walls. Material injected later in the injection period may breakthrough previously injected material. So breakthrough of the second injected material through the first injected material is completely governed by the fountain effect at the flow front. Breakthrough may not occur with the production of the oil filled rubber dampers. The oil must be completely surrounded by the rubber.

2.2.2 Balance equations

From continuum mechanics the balance equations can be derived. The problem is modelled isothermal, so the energy equation is not taken into account. The transport of the identity during the flow can be described by a convection equation. This convection equation is solved in an eulerian way.
The simulation of the filling stage can now be accomplished by adding the extra law of the conservation of identity of material particles, to the set of equations. This law is given by:

$$\dot{\underset{\sim}{\xi}} = \frac{D\underset{\sim}{\xi}}{Dt} = \frac{\partial \underset{\sim}{\xi}}{\partial t} + \vec{v} \cdot \vec{\nabla} \underset{\sim}{\xi} = \underset{\sim}{0} \tag{1}$$

where ξ denotes the labels.
The flow is modelled via the instationary Navier-Stokes equations for incompressible fluids. These equations can be derived from the equations for conservation of mass and balance of momentum, using Newton's constitutive equation. This equation is given by:

$$\sigma = -p\mathbf{I} + 2\eta\mathbf{D} \tag{2}$$

The dimensionless form of the Navier-Stokes equation is:

$$\begin{cases} Sr\frac{\partial \vec{v}}{\partial t} + (\vec{v} \cdot \vec{\nabla})\vec{v} - \frac{1}{Re}(\vec{\nabla} \cdot \eta\vec{\nabla})\vec{v} + \vec{\nabla}p = \vec{f} \\ \nabla \cdot v = 0 \end{cases} \tag{3}$$

with: Re the Reynolds number ($Re = \frac{\rho V L}{\eta_0}$)
Sr the Strouhal number ($Sr = \frac{\omega L}{V}$)

Since the Reynolds number is small, the inertia forces are negligible with respect to the viscous forces. Then the instationary Stokes equations are obtained:

$$\begin{cases} Sr\frac{\partial \vec{v}}{\partial t} - \frac{1}{Re}(\vec{\nabla} \cdot \eta\vec{\nabla})\vec{v} + \vec{\nabla}p = \vec{f} \\ \nabla \cdot v = 0 \end{cases} \tag{4}$$

With the numerical simulation of the filling of the mould, two equations are solved: the Stokes equation and the label equation.

2.2.3 Boundary conditions

The attachment point of the front moves in time. Because of this, it is difficult to model the front. This problem is solved by modelling the air at the walls with a slip condition. The air slips with the mean velocity of the front. At the front the slip condition for the air changes into a stick condition for the rubber. This point can be found by looking at the labels. The slipvelocity of the air is difficult to define at this complex geometry.
The geometry is given in Figure 4. Only one half is considered because of symmetry. At the

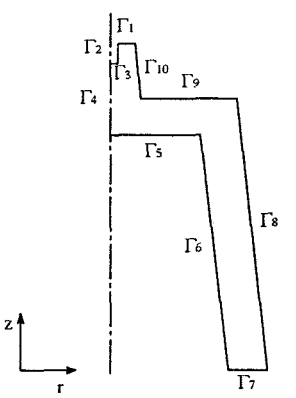

Figure 4: Boundaries of the problem

entrance boundary Γ_1 there is inflow of rubber and at the entrance boundary Γ_3 there is inflow of oil. Both, at the inflow of rubber and at the inflow of oil a Poisseuille profile is assumed. Boundary Γ_4 defines the symmetric axis. On the exit boundary Γ_7 the flow leaves the domain. At the other boundaries a slip or stick condition, depending on the material present, is defined. At the wall the air slips and the rubber sticks. The walls are not porous except at the corner between the boundaries Γ_8 and Γ_9. By this, no air will be left in this corner during filling of the mould. The boundary conditions for the problem are given by:

$$\left.\begin{array}{l} \vec{v}\cdot\vec{t}=0 \\ \vec{v}\cdot\vec{n}=v_{inflow}(r) \end{array}\right\} \; on\ \Gamma_1\ and\ \Gamma_3 \tag{5}$$

$$\left.\begin{array}{l} \sigma_t=0 \\ \vec{v}\cdot\vec{n}=0 \end{array}\right\} \; on\ \Gamma_4 \tag{6}$$

$$\left.\begin{array}{l} \sigma_t=0 \\ \sigma_n=0 \end{array}\right\} \; on\ \Gamma_7 \tag{7}$$

$$\left.\begin{array}{l} \vec{v}\cdot\vec{t}=v_{slip}(r,z) \\ \vec{v}\cdot\vec{n}=0 \end{array}\right\} \; on\ \Gamma_2,\Gamma_5,\Gamma_6,\Gamma_8,\Gamma_9\ and\ \Gamma_{10} \tag{8}$$

Both, the boundaries of the label equation and the Stokes equation are time-dependent. The boundary conditions of the Stokes equation depend on the position of the front. The two materials can be injected sequential (rubber - oil - rubber) or a combination of simultaneous and sequential (rubber - rubber and oil - rubber). The label of boundary Γ_3 has only the label of oil when oil is injected.

2.2.4 Numerical calculation

The calculation of the filling of the mould is executed in the following steps:

- solve the label equation with the velocities of the previous time step
- determine the time-dependent boundary conditions for the Stokes equation based on the labels of the boundary nodes
- solve the Stokes equation
- determine the viscosity based on the labels
- generate output for postprocessing

The program and the input for the preprocessor, solver and postprocessor can be found in Appendix B.

3 Materials

The damper consists of two materials: a rubber and an oil. The rubber component of the conventional oil-filled rubber damper is butyl rubber. Butyl rubber is a vulcanized rubber, and therefore not thermo-reversible. To simplify the experiments, Hytrel 6356, is used. Hytrel 6356 is a thermoplastic polymer. So Hytrel does not vulcanize, but it behaves like a vulcanized rubber. Hytrel is a hygroscopic material, so the time during which Hytrel is exposed to atmospheric moisture must be kept minimum. The resin should be dried for 2-3 hours at $105°C - 120°C$, if exposure to ambient air exceeds one hour. The viscosity of Hytrel was measured using small amplitude oscillatory shear experiments on a Rheometrics Dynamics Spectrometer RDS-II. The parallel plate geometry was used. The sample geometry was: ø25 x 1.374 [mm]. The data are measured as a function of the frequency at different temperatures between $215°C$ and $275°C$. The melting temperature of Hytrel is equal to $211°C$. The dynamic viscosity of Hytrel is about 50 [Pas] at a temperature of $250°C$. The experiments are done at this temperature.
The oil in the damper is a silicon or polybutene oil. Silicon oils are build up of non-cross linked polydimethylsiloxane elastomeric chains. They mainly behave viscous and are water free fluids. The silicon oils used are produced by Wacker Chemie. The silicon oils AK are available in a wide viscosity range. For damping applications, silicon oils with a viscosity of $\eta > 10$ [Pas] are of real importance (de Geus [1]).
The valve, which regulate the injection of oil in the mould, is not able to resist high pressures, which is necessary to press a higly viscous oil through the pipes. The valve opens when the pressure is too high. For this reason the viscosity of the oil may not be taken too high.
A prediction of the viscosity of the oil can be determined. The maximum pressure the valve can stand is about $27.1 \cdot 10^5$ [Pas]. The pressure necessary to press the oil through the pipe has

to be lower than this value. It is assumed that the volumeflow in the straight pipes satisfy the relation of Hagen-Poiseuille:

$$\phi = \pi R^2 u_m = -\frac{\pi}{8\eta}\frac{dp}{dx}R^4 \tag{9}$$

where u_m is the mean velocity, η is the viscosity, $\frac{dp}{dx}$ is the pressuregradient and R the radius of the pipe. The pressure difference over a pipe with length L is equal to

$$\Delta p = L \cdot \phi \cdot \frac{8\eta}{\pi R^4} \tag{10}$$

The supply pipe has a length of 0.977 [m] and an inner radius of $2.0 \cdot 10^{-3}$ [m]. The needle has a length of $5.5 \cdot 10^{-2}$ [m] and an inner radius of $5.0 \cdot 10^{-4}$ [m]. The prescribed flow of oil is equal to $3.5 \cdot 10^{-7}[m^3 s^{-1}]$. The pressure drop is calculated for two different viscosities. The results are given in Table 1. The value of the pressure difference is mainly influenced by the needle.

	Δp [Pa]		
η [Pas]	supply pipe	needle	total
0.1	$5.4 \cdot 10^3$	$7.84 \cdot 10^4$	$8.38 \cdot 10^4$
1.0	$5.4 \cdot 10^4$	$7.84 \cdot 10^5$	$8.38 \cdot 10^5$

Table 1: Pressure drop

Both Wacker AK100 and Wacker AK1000 satisfy the condition above. The dampers which have been made, are filled with AK100 ($\eta = 0.1$ [Pas]).

4 Experimental and numerical results

4.1 Numerical results

The viscosity for the air, oil and rubber are modelled as constants. In Table 2 these viscosities are given. The viscosity of air cannot be chosen lower than the given value, because in that case the Reynolds number becomes too high. The exact viscosity of air is equal to $17.1 \cdot 10^{-6}$ [Pas]. In that case, instabel vortices are developed in the air, after which the calculations after a few steps stopped. In a flow through a straight pipe the viscosity of air could be taken lower. The viscosity of the rubber is chosen a factor 10^5 higher than the viscosity of air and a factor 10^2 higher than the viscosity of oil. In reality the viscosity of rubber is also a factor 10^2 higher than the viscosity of oil. The viscosity is interpolated linearly per element to avoid the occurrence of

material	η [Pas]
air	$1.0 \cdot 10^{-1}$
rubber	$1.0 \cdot 10^4$
oil	$1.0 \cdot 10^2$

Table 2: Chosen viscosities

unrealistic values in the integration points. This may occur at the two material interfaces due to the quadratic shape functions of the elements used.
The numerical simulations are performed using a finite element mesh consisting of 1472 quadratic triangular elements as is depicted in Figure 5. The fine mesh is necessary to visualize the fountainflow and to determine the front of the rubber. The applied boundary conditions are already given in Figure 4. Because of symmetry only one half of the geometry is modelled. The slipvelocities of the air on the boundaries are described in Appendix C.

The Stokes equation with the incompressibility constraint is solved with a penalty method. The calculation of the velocity and the pressure are uncoupled and the incompressibility constraint is taken into account in the Stokes equation with a penalty parameter after discretization of equation (4) and applying the Galerkin formulation. The velocity field is calculated by a Picard iteration method (successive substitution) every time step. The rate of convergence of Picard is linear (see also van Steenhoven [4]).
For the solution of the particle tracking problem, the Streamline Upwind Petrov-Galerkin finite element method is applied using the same mesh as in the Stokes problem. The SUPG method provides stable solutions in case of convection dominated flows with discontinuities in the solution as may occur in the particle tracking problem. The classical upwind scheme is used ($\zeta = 1$ see Appendix D). From all the types of upwinding within the SUPG method this scheme gives the least accurate, but smoothest results. The time integration is carried out with an Euler implicit scheme ($\theta = 1$). This scheme applied to equation (1) leads to

$$\frac{\underset{\sim}{\xi}_{n+1} - \underset{\sim}{\xi}_n}{\Delta t_{n+1}} + \vec{v}_{n+1} \cdot \vec{\nabla} \underset{\sim}{\xi}_{n+1} = \underset{\sim}{0} \tag{11}$$

The total time-span of 16.0 [s] is divided into 1600 timesteps. The oil is injected after 6.0 [s] till 12.0 [s]. Calculations are done for both, the combination of simultaneous and sequential and sequential injection. The chosen flows can be found in Table 3. The materials are defined by the following labels: air with label 0.0, rubber with label 1.0 and oil with label 2.0.

material	flow $[mm^3/s]$	
	combined injection	sequential injection
rubber	$4.25 \cdot 10^{-7}$	$8.5 \cdot 10^{-7}$
oil	$3.5 \cdot 10^{-7}$	$3.5 \cdot 10^{-7}$

Table 3: Chosen flows

The label plots for the combination of simultaneous and sequential injection are shown in Figure 6 - Figure 9. The slipvelocities on boundaries Γ_6 and Γ_8 are defined a little bit too high. The fountain flow effect is seen in Figure 8 and in Figure 9: the oil spreads towards the wall. The labelplot at t=14 [s] shows a small, long distribution of oil. The oil can also be found at the boundaries Γ_5 and Γ_6. In practical applications this is not admissible.
The label plots for the sequential injection are shown in Figure 10 - Figure 13. The first seconds of the simulation give the same label distribution as the combined injection. The slipvelocities are defined in the same way for the sequential injection as for the combined injection. For this reason the slipvelocities at boundary Γ_8 are defined much too high. The labelplot at t=14 [s] shows a very good distribution of the oil. So the distribution of oil for the simulation of the sequential injection is better than the one for the combined injection.

4.2 Experimental results

Before the experiments could be done first the total injection volume had to be known. The input channel has a volume of 651 $[mm^3]$. So the total volume that has to be injected is equal to 11540 $[mm^3]$. The inside diameter for the buffers is equal to 16 [mm], so the total displacement of the two pistons has to be 57.4 [mm].
The plates of rubber have been made at a temperature of $230°C$. The input of rubber is twenty slices with a thickness of 3.0 [mm] each. The slices have been dried for at least three hours at a temperature of about $120°C$. The drying is necessary to reduce the expanding of the rubber during the heating caused by water evaporation. After drying the experiment has to be done within an hour.
The buffer temperature is chosen equal to $250°C$, so $39°C$ above the melt temperature of Hytrel. The mould is not heated, but due to the large contact surface between the buffer and the mould, the mould becomes warm.
Two experiments have been done. At the first experiment the injection times of the oil have been chosen according to the numerical injection times of the combined injection: 6.0 and 12.0 [s] (4.0 and 8.0 on the scale of the control unit).
At the first try the oil has broken through the rubberfront. Some oil can be seen at the end of the product. At the second try the oil has been injected later, the next values on the control unit have been chosen: 4.5 and 8.0. Unfortunately the oil has also broken through the rubberfront. The displacements of the pistons for the two experiments can be found in Table 4. At both the experiments all the rubber has been injected. Much more material has been injected than the theoretical 57.4 [mm]. After the oil has been broken through the rubberfront it leaks out of the mould. In the congealed injection channels of both products no oil was present.

	displacement [mm]	
piston	first experiment	second experiment
rubber	58	59
oil	5	21

Table 4: Displacement of the pistons

A few problems have been occured. It is difficult to adjust the pressure of the oil. The maximum pressure the valve can stand is about 0.5 tons (27.1 bar on the piston). It is chosen to adjust the pressure at about 0.4 tons (21.7 bar on the piston) on a scale of 23 tons. This cannot be done accurate enough, and that is why at the first experiment less oil has been injected than at the second experiment. Further on, the valve has not been closed completely when only the rubber is injected, so the pressure of the oilpress has been chosen too high.
Second, during the heating some rubber comes out of the buffer. So the exact amount of rubber injected is unknown. This problem could be solved when the volume of the mould should be smaller.
Third, it is very difficult to control the direction of injection of the oil. A few experiments without the injection of the oil show that the filling of the mould occurs symmetrically. When also oil has been injected, the filling of the mould is asymmetrically.
The fourth and last problem occurs when the mould has been filled completely. The rubber has been pressed into the needle, because the oil can leak through the coupling between the pipe

and the valve.

5 Conclusions and recommendations

5.1 Conclusions

The numerical calculations show that the distribution of oil for the simulation of the sequential injection is better than the one for the combined injection. The combined injection gives a wider distribution of oil, but oil can be found near the innerwall and the oil approaches the rubberfront at the end of the simulation. The real viscosity of air could not be implemented in the program, because in that case the Reynolds number becomes too high.
The method with the slip of air at the walls give only good results when the right slipvelocities are chosen. For a complex geometry this is quite difficult and time-consuming. In the future new elements will be available which generate the right slipvelocity.
The fine mesh is necessary to determine the front of the rubber and to visualize the fountain flow. The fountain flow effect is visualized at the combined injection.
The injection of oil at the experiments is not controllable and reproduceble. The valve can not stand high pressures, so an oil with a low viscosity has to be used. The filling of the mould is asymmetrically when also oil is injected. The theoretical, vertical injection of the oil can not be adjusted accurately. This could be the reason for the asymmetrical injection.
When the mould is completely filled, rubber is pressed into the needle. The vicosity of the oil is too low by which the oil is pressed through leaks at the couplings.
Drying of the rubber slices is necessary to reduce the expanding of the rubber during the heating.
During heating of the buffer some rubber leaks out of the buffer. The buffer has to be filled almost completely to ensure that the mould will be filled completely.

5.2 Recommendations

To appoximate the reality more, the program has to be extended with the energy equation and models for the viscosity. The premature congelation of rubber could have a great influence on the profiles of the flow. In the future the new element could be implemented in the program.
The volume of the mould should be made smaller. In that case less slices are needed for an experiment and the leaking of the rubber out of the buffer will be reduced.
The injection of oil has to be changed to inject also oils with a higher viscosity than the oil used ($\eta = 0.1$ [pas]). The 5/2 valve has to be replaced by a valve which resists the high pressures necessary to press the oil with the higher viscosity through the pipe and needle. Those oils will leak less at the couplings between the pipes and the valve. In that case less rubber will be pressed into the needle.
Also experiments should be done with the sequential injection, because the numerical results give a well defined labeldistribution.
The injection of the oil has to be vertical. Provisions have to be made to achieve this accurately.

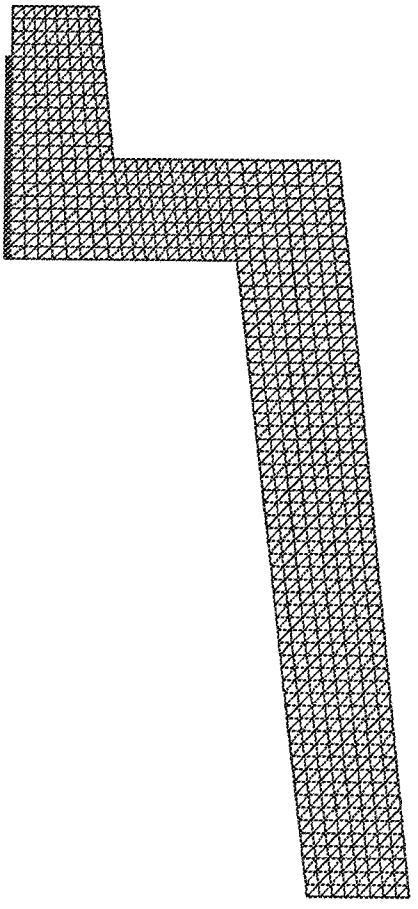

Figure 5: Finite element mesh of the problem

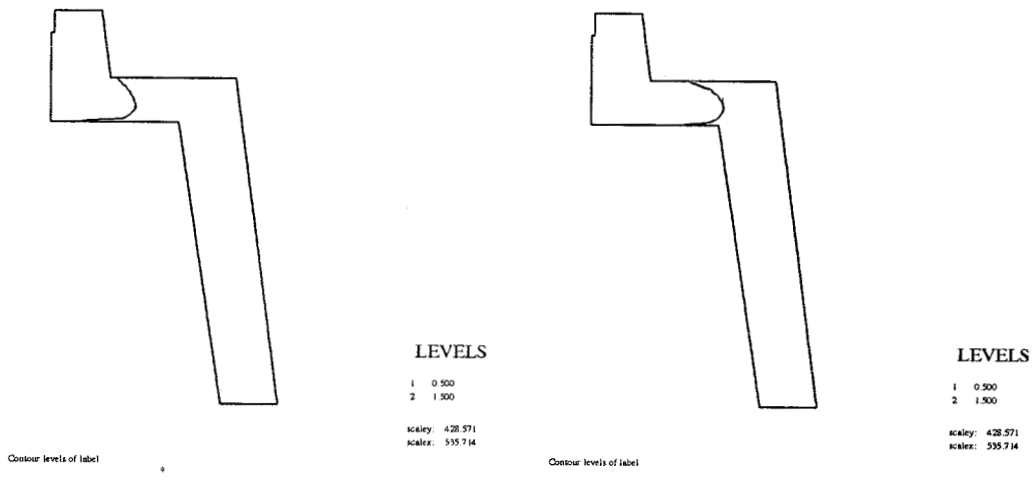

Figure 6: Combined injection (t=1 [s] and t=2 [s])

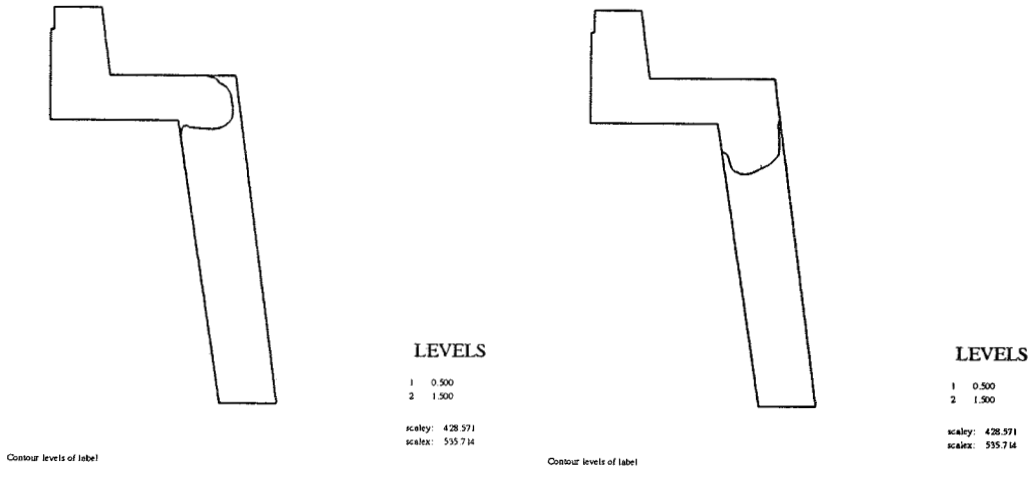

Figure 7: Combined injection (t=4 [s] and t=6 [s])

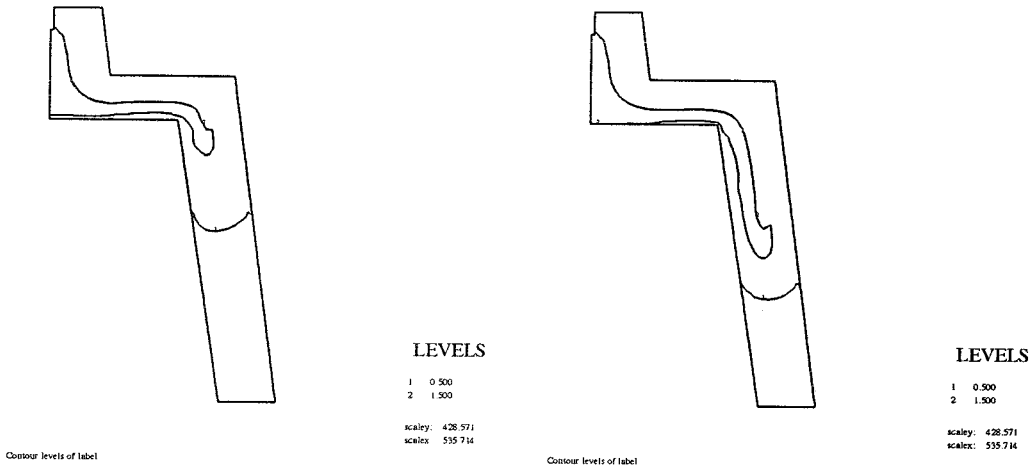

Figure 8: Combined injection (t=8 [s] and t=10 [s])

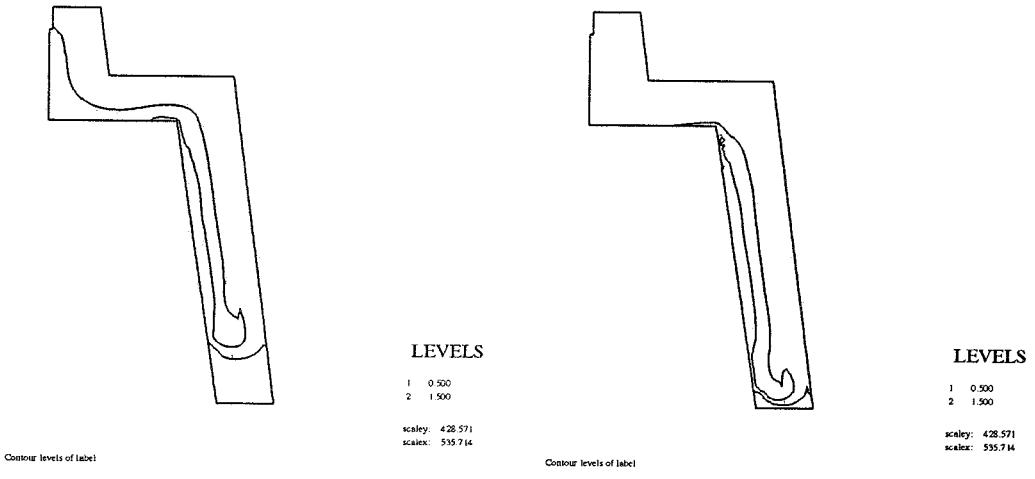

Figure 9: Combined injection (t=12 [s] and t=14 [s])

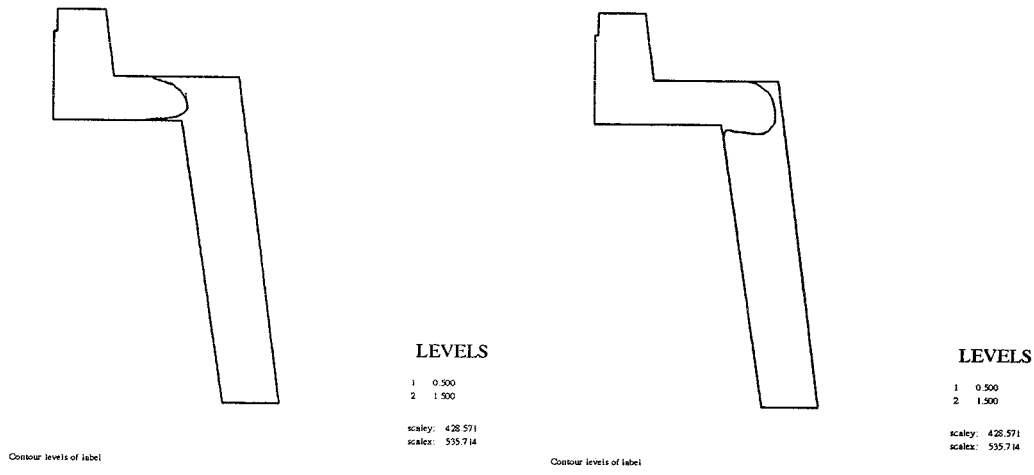

Figure 10: Sequential injection (t=1 [s] and t=2 [s])

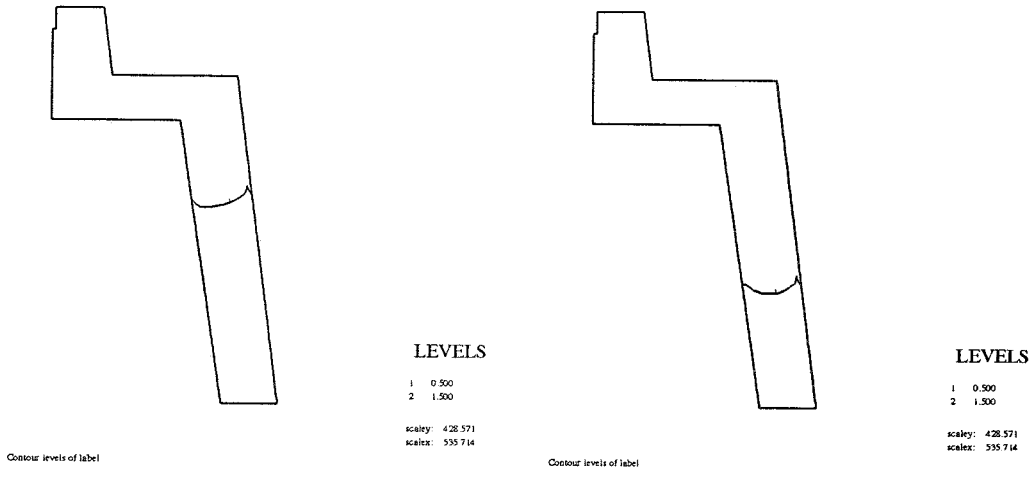

Figure 11: Sequential injection (t=4 [s] and t=6 [s])

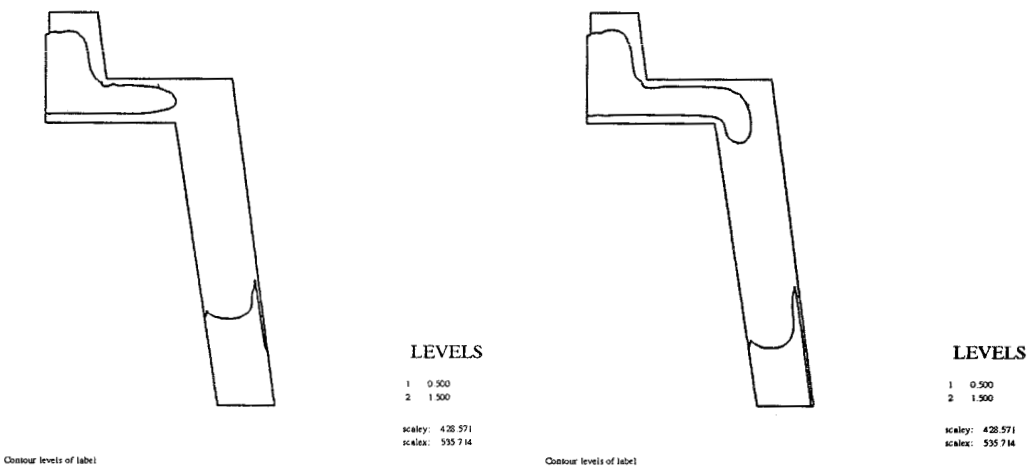

Figure 12: Sequential injection (t=8 [s] and t=10 [s])

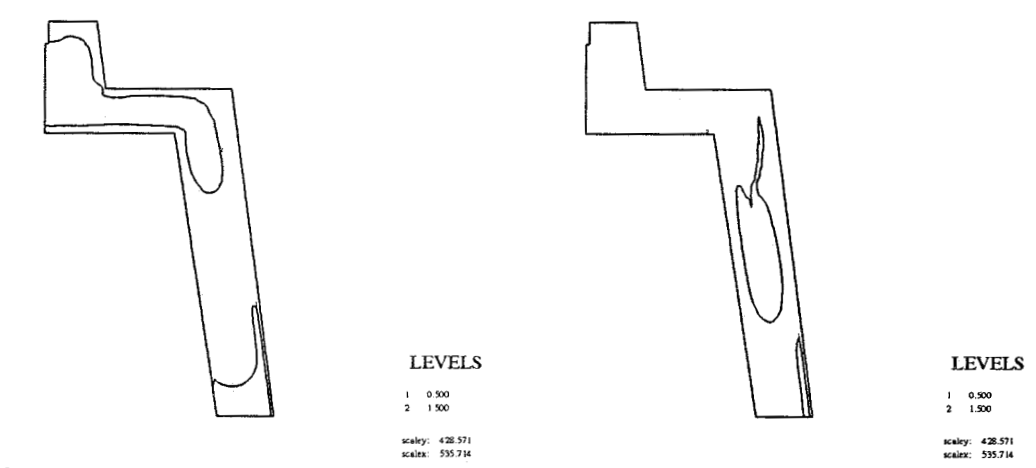

Figure 13: Sequential injection (t=12 [s] and t=14 [s])

Appendix A Volume of the product

The volume of the product can be calculated with the formula for the volume of a cone.

$$V = \frac{1}{3}A_g h = \frac{1}{3}\pi r^2 h \tag{12}$$

with: A_g the groundsurface of the cone
h the height of the cone
r the radius of the groundsurface of the cone

With the formula for the angle α of the cone

$$\tan\alpha = \frac{r}{h} \tag{13}$$

equation (12) leads to

$$V = \frac{\pi}{3\tan\alpha}r^3 \tag{14}$$

The different radii can be taken from Figure 1. The angle of the mould is 6.0 degrees. The volumes of the cones for the different radii can be seen in Table 5. The exact volume of one product is about $V = (41179 - 21890) - (15663 - 7263) = 10889[mm^3]$.

radius [mm]	$volume[mm^3]$
13.00	21890
16.05	41179
9.00	7263
11.63	15663

Table 5: Volumes of the different cones (total volume)

Suppose the oil is situated 1.0 [mm] from the outside of the product, the volume of the oil injected can be calculated in the same way as the calculation of the total volume of the product. The volumes of the cones for the different radii are given in Table 6. The exact injectionvolume

radius [mm]	$volume[mm^3]$
12.11	17673
14.84	33244
9.89	9653
11.63	15565

Table 6: Volumes of the different cones (oilvolume)

of oil is about $V = (33244 - 17673) - (19565 - 9653) = 5659[mm^3]$. The position of the oil is in this case ideal. This value of the injectionvolume of oil has been taken as a maximum.

Appendix B Listings

B.1 Main program

```
      program damper
c=======================================================================
c     In this program the momentum and label equation are solved. The
c     The modified theta-method is used to approximate the time deri-
c     vates. The problem specific parameters are listed in a separate
c     input file which is read by the main program.
c     Used files :
c      meshoutput       Output of program sepmesh
c      damper.dat      Standard input file
c      damper.out      Standard output file
c      damper.pos      Postscript input file
c
c      In array isol is stored:
c           column(1)  velocity solution
c           column(2)  label
c           column(3)  viscosity
c
c      In array ivcold is stored:
c           column(1)  velocity solution at preceding time level
c           column(2)  label at preceding time level
c           column(3)  viscosity
c=======================================================================
      implicit none
      double precision difmax1, difmax2
      double precision p, q
      integer kmesh(100), kprob(500), iuser(100,2)
      integer matrs(5,2), irhsd(5,2), matrm(5,2)
      integer iessb(5,1), matrms(5,1)
      integer ielhlp, xjdim, jpoint, ipoint
      integer isol(5,3), ivcold(5,3), iu1(10)
c
c=======================================================================
c      Arrays containing information concerning the structure of
c      the large matrix:
c      intmatNS : for Navier Stokes equation,
c      intmatL  : for label equation.
c=======================================================================
c
      integer intmatNS(5), intmatL(5,2)
      integer iinvec(11), invec1(5,3), iinbld(10)
      integer invec2(5,3), ptype
      integer ichois, ix, jdegfd, ivec
```

```
      integer istep
      integer i, iresul(5,1), ihelp(5,3), ihelpp(5,3)
      integer ihlp(5,3), itest(5,3), ibound(5,3)
      integer shearv(5), shearn(5), iinder(2)
      integer ieta(5), idum(5)
      integer itemp(5)
      integer jbuffr, jbfree
      integer iinput(500)
      integer imovestep, istep_pw
      integer ichcrv, istart, irotat, ioutp, itime

      double precision user(100,2), u1(10), alpha2, beta2
      double precision rinvec(11)
      double precision rdum

      common ibuffr(20 000 000)
      integer ibuffr
      common /cbuffr/ nbuffr, kbuffr, intlen, ibfree
      integer nbuffr, kbuffr, intlen, ibfree
      common /cmcdpi/ irefwr, irefre, irefer
      integer irefwr, irefre, irefer
      common /ctima/ theta, deltat, t, ratime(7),
     .               nstep, irtime(9)
      integer ictime, nstep, irtime
      double precision theta, deltat, t, ratime, tstep
      common /carray/ iinfor, infor(3,1500)
      integer iinfor, infor
      common /dprotim/ trb, tre, tob, toe, injec
      double precision trb, tre, tob, toe, injec
c
c=======================================================================
c      Common blok for machine dependent variables:
c            iref10: reference number for file meshoutput
c            iref73: reference number for file sepcomp.inf
c            iref74: reference number for file sepcomp.out
c=======================================================================
c
      integer  machin, lenwor, numchr, nrec4, iref10, iref73, iref74,
     .         irefpl, idumma(21)
      common /cmachn/  machin, lenwor, nrec4, numchr, iref10, iref73,
     .                 iref74, irefpl, idumma
c
c=======================================================================
c     I N I T I A L I Z E     A R R A Y S
c=======================================================================
c
      kmesh(1)   = 100
```

```
c
c=======================================================================
c     Note: element 800 needs 121 entries in kprob !!!!!
c=======================================================================
c
      kprob(1)   = 500
      iinput(1)  = 500
      do 101 i = 2,5
         iinput(i) = 0
101   continue
c
c=======================================================================
c      Arrays for coefficients:
c=======================================================================
c
c*****   Momentum equation:
      iuser(1,1) = 100
      user(1,1)  = 100.0d0
c
c*****   Label equation:
      iuser(1,2) = 100
      user(1,2)  = 100.0d0
c
c=======================================================================
c     S T A R T   T H E   P R O G R A M                        PG 2.2
c=======================================================================
c
      istart = 10
      irotat = 0
      ioutp = -1
      itime = 0
      CALL START(istart, irotat, ioutp, itime)
      nbuffr = 20 000 000
c
c=======================================================================
c     R E A D   T H E   M E S H                                PG 3.2
c=======================================================================
c*****   Read information from the file "meshoutput"
c*****   ichois = 5: Renumber Sloan band
c*****   iref10    : Reference number for file "meshoutput"
c                    This file is opened in START.

      ichois = 5
      CALL MESHRD(ichois, iref10, kmesh )
c
c=======================================================================
c     P R O B L E M   D E F I N I T I O N                      PG 4.1
```

```
c=======================================================================
c*****   Information for PROBDF is read from the standard input file
c*****   and stored in kprob and iinput.
c
      ichois = 2
      CALL PROBDF( ichois, kprob, kmesh, iinput )
c
c*****   Write information of problems to file sepcomp.inf. This
c*****   information is necessary for the postprocessing program SEPPOST
c
       write(iref73,999) (iinput(i),i=1,iinput(3))
999    format(5i6/(10i6))
c
c=======================================================================
c     I N F O R M A T I O N   O F   L A R G E   M A T R I X      PG 4.4
c=======================================================================
c
c*****   Impulse equation:
c
      CALL COMMAT( 2 , kmesh, kprob, intmatNS )
c
c*****   Label equation:
c*****   This "double" initialization is necessary for element 800.
c
      CALL COMMAT( 1002, kmesh, kprob, intmatL(1,1) )
      CALL COMMAT( 1002, kmesh, kprob, intmatL(1,2) )
c
c=======================================================================
c     C R E A T E   S T A R T   V A L U E S                  PG 5.3
c=======================================================================
      CALL CREATE( 0, kmesh, kprob, ivcold )
      CALL CREATE( 0, kmesh, kprob, isol )
c=======================================================================
c     Read parameters:
c          trb    :   start of the inflow of rubber
c          tre    :   end of the inflow of rubber
c          tob    :   start of the inflow of oil
c          toe    :   end of the inflow of oil
c          nstep  :   number of time steps in interval trb - tre
c          problem type: 1. Oil filled rubber damper
c          sequence of injection: 1. rubber; rubber and oil; rubber
c                                 2. rubber; oil; rubber
c=======================================================================
      write(*,10)
10    format(' double precision tre=', f10.2 )
      read(*,*) tre
      write(*,20)
```

```
20    format(' double precision tob=', f10.2 )
      read(*,*) tob
      write(*,30)
30    format(' double precision toe=', f10.2 )
      read(*,*) toe
      write(*,40)
40    format(' integer injec=', i5 )
      read(*,*) injec
      write(*,50)
50    format(' integer nstep=', i5 )
      read(*,*) nstep
      write(*,60)
60    format(' steps/plot=', i5 )
      read(*,*) istep_pw
c
      trb=0d0
      ptype=1
c
c***** Program can only deal with theta=1d0 !
c
      theta=1d0
c
c***** Write current values to output file:
c
      write(irefwr,*)'theta:',theta
      write(irefwr,*)'trb:',trb
      write(irefwr,*)'tre:',tre
      write(irefwr,*)'tob:',tob
      write(irefwr,*)'toe:',toe
      write(irefwr,*)'injec:',injec
      write(irefwr,*)'nstep:',nstep
c
c***** Time steps before and after trb:
c
      deltat = (tre-trb)/nstep
c
c***** Counter for time steps after trb:
      imovestep = 0
c
c***** Start with time step before trb:
c
      t = 0d0
c
c=======================================================================
c     S T A R T   I T E R A T I O N
c=======================================================================
      do 500 istep = 1,nstep
```

```
c
      tstep = theta*deltat
      t = t + tstep
c
c***** Count number of steps after trb (for plots etc.):
c
      if (t.gt.trb) then
         imovestep=imovestep+1
      endif
c
c***** Write some values to output file:
c
      write(irefwr,*)'istep:',istep
      write(irefwr,*)'imovestep:',imovestep
      write(irefwr,*)'deltat:',deltat
      write(irefwr,*)'t:',t
c
c=======================================================================
c     P R E S C R I B E   B O U N D A R Y   C O N D I T I O N S  PG 5.5
c=======================================================================
c
      if (istep.eq.1) then
          ictime = 0
          CALL PRESTM( ictime, kmesh, kprob, isol )
      else
          ictime = 1
          CALL PRESTM( ictime, kmesh, kprob, isol )
      endif
c
c=======================================================================
c     F I L L   C O E F F I C I E N T S                    PG 5.11
c=======================================================================

      if ( istep.eq.1 ) then
          CALL FILCOF( iuser(1,1), user(1,1), kprob, kmesh, 1 )
      endif
      CALL COPYVC( isol(1,1), ivcold(1,1) )
      CALL COPYVC( isol(1,2), ivcold(1,2) )
c
c=======================================================================
c     L A B E L   E Q U A T I O N
c=======================================================================
c     C O M P U T E   M A T R I X  &  V E C T O R          PG 5.1
c     Compute stiffness matrix and right-hand side with essential
c     boundary conditions.
c     Determine stiffness matrix (matrs(1,2)) and mass matrix
c     (matrm(1,2)) in one call of BUILD.
```

```
c=======================================================================
      if ( istep.eq.1 ) then
          CALL FILCOF( iuser(1,2), user(1,2), kprob, kmesh, 2 )
      endif
c
      iinbld(1) = 10
      iinbld(2) = 1
      iinbld(3) = 0
      iinbld(4) = 2
      iinbld(5) = 1
      iinbld(6) = 0
      iinbld(7) = 0
      iinbld(8) = 0
      iinbld(9) = 2
      iinbld(10) = 3

      CALL BUILD(iinbld, matrs(1,2), intmatL, kmesh, kprob,
     &           irhsd(1,2), matrm(1,2), isol(1,2), ivcold,
     &           iuser(1,2), user(1,2))
c
c=======================================================================
c    C R E A T E   N E W   L A R G E   M A T R I X            PG 5.10
c    For label equation.
c=======================================================================
c
      CALL COPYMT( matrs(1,2), matrms(1,1), kprob )
      CALL ADDMAT( kprob, matrms(1,1), matrm(1,2), intmatL(1,1),
     &             tstep, alpha2, 1d0, beta2 )
c
c=======================================================================
c     S O L V E   L A B E L   E Q U A T I O N
c=======================================================================
c
c=== compute mass matrix * old y-label ======================== PG 6.9 ==
c
      CALL MAVER( matrm(1,2), ivcold(1,2), iresul(1,1),
     &            intmatL(1,1), kprob, 5 )
c
c=== y-label boundaries at new time level ===========================
c
      CALL MAVER( matrms(1,1), isol(1,2), iessb(1,1),
     &            intmatL(1,1), kprob, 6 )
      CALL ALGEBR( 3, 0, iresul(1,1), iessb(1,1), ihelp(1,1),
     &             kmesh, kprob, 1d0, -1d0, p, q, ipoint )
c
c=== solve y-label equation ========================================= PG 6.8 ==
c
```

```
      CALL SOLVE( 0, matrms(1,1), isol(1,2), ihelp(1,1),
     &            intmatL, kprob )
c
c=======================================================================
c     M O M E N T U M   E Q U A T I O N
c=======================================================================
c     C O M P U T E  M A T R I X  &  V E C T O R              PG 5.1
c=======================================================================
c
      iinbld(1) = 10
      iinbld(2) = 1
      iinbld(3) = 1
      iinbld(4) = 1
      iinbld(5) = 1
      iinbld(6) = 0
      iinbld(7) = 0
      iinbld(8) = 0
      iinbld(9) = 1
      iinbld(10) = 3

      CALL BUILD(iinbld, matrs(1,1), intmatNS, kmesh, kprob,
     &           irhsd(1,1), matrm(1,1), isol(1,1), ivcold,
     &           iuser(1,1), user(1,1))
c
c=======================================================================
c     I N C O R P O R A T E  B O U N D A R Y  C O N D I T I O N S
c     including partial slip on boundary
c=======================================================================
c
      CALL COPYVC(isol(1,1), ibound(1,1))
      CALL COPYVC(isol(1,2), itest(1,1))

      iinvec(1) = 5
      iinvec(2) = 32
      iinvec(3) = 0
      iinvec(4) = 0
      iinvec(5) = 1
      rinvec(1) = 3d0
      CALL MANVEC( iinvec, rinvec, itest(1,1), idum, itest(1,1),
     &            kmesh, kprob)

c      CALL PRINOV(ibound(1,1), kmesh, kprob, 2, 'bound', rdum, idum)
c      CALL PRINOV(itest(1,1), kmesh, kprob, 2, 'test', rdum, idum)
c
c     array kmeshm to workspace
c
      CALL INI070(kmesh(27))
```

```
c
c     create start vector
c
      ichois = 0
      ichcrv = 1001
      iu1(1) = 0
      u1(1) = 0d0
      CALL CREAVC(ichois, ichcrv, idum, ihlp(1,1), kmesh, kprob,
     &            iu1, u1, idum, rdum)
c
c     define pointer in ibuffr
c
      CALL BOUNCD(kprob, itest, ibound, isol,
     &            ibuffr(infor(1,kmesh(27))), ihlp)

c      CALL PRINOV(ihlp, kmesh, kprob, 2, 'ihlp', rdum, idum)

      iinder(1) = 2
      iinder(2) = 4
      CALL DERIV(iinder, ihlp(1,1), kmesh, kprob, ihlp(1,1),
     &           iuser(1,1), user(1,1))

c      CALL PRINOV(ihlp, kmesh, kprob, 2, 'ihlp', rdum, idum)

      CALL BOUNCD1(kprob, itest, ibound, isol,
     &             ibuffr(infor(1,kmesh(27))), ihlp)

c      CALL PRINOV(isol(1,1), kmesh, kprob, 2, 'bound1', rdum, idum)

      CALL MAVER(matrs(1,1), isol, ihelpp(1,1), intmatNS, kprob, 6)

      iinvec(1) = 2
      iinvec(2) = 27
      rinvec(1) = 1d0
      rinvec(2) = -1d0
      CALL MANVEC(iinvec, rinvec, irhsd(1,1), ihelpp(1,1),
     &            irhsd(1,1), kmesh, kprob)
c
c=======================================================================
c     S O L V E   E Q U A T I O N                              PG 6.8
c=======================================================================
c
      CALL SOLVE( 1, matrs(1,1), isol(1,1), irhsd(1,1), intmatNS,
     &            kprob )
c
c=======================================================================
c     C O M P U T E   M A X I M U M   D I F F E R E N C E S
```

```
c=======================================================================
c
      CALL DIFFVC( 0, isol(1,1), ivcold(1,1), kprob, difmax1 )
      write(6,2000) istep,difmax1
c
      CALL DIFFVC( 0, isol(1,2), ivcold(1,2), kprob, difmax2 )
      write(6,2100) istep,difmax2
c
c=======================================================================
c     C O M P U T E   E T A                                    PG 6.5
c=======================================================================
      iinvec(1) = 5
      iinvec(2) = 32
      iinvec(3) = 0
      iinvec(4) = 1
      iinvec(5) = 1

      if ( ptype.eq.1) then
         rinvec(1) = 1d0
      endif

      CALL MANVEC( iinvec, rinvec, isol(1,2), invec2,
     &             ieta, kmesh, kprob )

c=======================================================================
c     Due to large viscosity gradients, it is possible that the
c     viscosity in the centroid of the element obtains a negative
c     value. To avoid this, the viscosity values in the vertices of the
c     elements is interpolated lineairly to the nodal points.
c=======================================================================
      iinder(1) = 2
      iinder(2) = 4

      CALL INI056(ivcold(1,3), 'Main')

      CALL DERIV( iinder, ivcold(1,3), kmesh, kprob, ieta,
     &            iuser, user )

c=======================================================================
c     Prevent negative label to increase.
c=======================================================================

      iinvec(1) = 11
      iinvec(2) = 32
      iinvec(3) = 0
      iinvec(4) = 1
      iinvec(5) = 1
```

```
       iinvec(6) = 3
       iinvec(7) = 0
       iinvec(8) = 1
       iinvec(9) = 2
       iinvec(10) = 3
       iinvec(11) = 2

       rinvec(1) = 2d0
       CALL MANVEC( iinvec, rinvec, isol(1,2), invec2, isol(1,2),
      &             kmesh, kprob)
c
c=======================================================================
c Print arrays after last iteration step.                        PG 8.
c=======================================================================
c
       if (istep.eq.nstep) then
        CALL PRINOV( isol(1,1), kmesh, kprob, 2,
      &              'isol(1,1) = velocity', xjdim, jpoint )
        CALL PRINOV( isol(1,2), kmesh, kprob, 2,
      &              'isol(1,2) = label', xjdim, jpoint )
        CALL PRINOV( ivcold(1,3), kmesh, kprob, 2,
      &              'ivcold(1,3) = viscosity', xjdim, jpoint )
       endif
c
       CALL COPYVC( ivcold(1,3), isol(1,3) )
c
c=======================================================================
c     W R I T E   O U T P U T  PG 8.7
c=======================================================================
c
c*****   Save some results after trb:
c
       if (t.ge.trb.and.mod(imovestep,istep_pw).eq.0.or.
      &                                        istep.eq.nstep) then
           CALL OUTTIM( t, kmesh, kprob, isol )
       endif
c
c==== continue computation ============================================
c
       write(6,2500) t
c
500    continue
c
       write(*,*) 501
c
c=======================================================================
c     S T O P   T H E   P R O G R A M PG 4.3
```

```
c=======================================================================
      CALL FINISH( 0 )

2000  format( ' istep: ',i5,' difmax1 = ',e10.3 )
2100  format( ' istep: ',i5,' difmax2 = ',e10.3 )
2500  format( ' time= ',e10.3 )
2600  format( ' i= ',i3,' iuser: ',i10,' user: ',e10.3 )
      end
```

clearpage

```
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c                          S U B R O U T I N E S
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
      SUBROUTINE FUNVEC( rinvec, reavec, nvec, coor, outvec )
      implicit none
      common /cmcdpi/ irefwr, irefre, irefer
      integer irefwr, irefre, irefer
      integer nvec
      double precision rinvec(*), reavec(nvec), coor(*), outvec

c=======================================================================
c               D E T E R M I N E   O U T V E C
c=======================================================================
      if (rinvec(1).eq.1d0) then
c**** Oil filled rubber damper
c************************************************************************
c     reavec(1)=label
c************************************************************************

        if (reavec(1).le.0.5d0) then
            outvec = 1.0d-1
        else
            if (reavec(1).le.1.5d0) then
               outvec = 1.0d4
            else
               outvec = 1.0d2
            endif
        endif

      else if (rinvec(1).eq.2d0) then
c************************************************************************
c     reavec(1)=isol(1,2) (solution of the label)
c************************************************************************

        if (reavec(1).lt.0d0) then
            outvec = 0d0
```

```
        else
            if (reavec(1).gt.2d0) then
                outvec = 2d0
            else
                outvec = reavec(1)
            endif
        endif

      else if (rinvec(1).eq.3d0) then
c*********************************************************************
c     reavec(1)=itest(1,1) (=isol(1,2))
c*********************************************************************

        if (reavec(1).lt.0.5d0) then
            outvec = 1d0
        else
            outvec = 0d0
        endif

      endif
      end
c
c
c
      FUNCTION FUNCBC( ichois, x, y, z)

      implicit none

      double precision theta, deltat, t, ratime
      integer nstep, irtime
      common /ctima/ theta, deltat, t, ratime(7),
     .               nstep, irtime(9)
      double precision trb, tre, tob, toe, injec
      common /dprotim/ trb, tre, tob, toe, injec
      double precision x, y, z
      double precision pi, ugemr, ugemo, funcbc, label
      double precision floo, flor, alpha
      integer ichois

      pi = 4d0*atan(1d0)
      floo = 3.5d-7
      alpha = (6.0 * pi)/180
c
c**** definition of the inflow ***************************************
c
      if (injec.eq.1) then
         flor = 4.25d-7
```

```
            if (t.lt.tob.or.t.gt.toe) then
               ugemo=0d0
               ugemr=flor/((pi*(3.551d-3)**2)-(pi*(0.25d-3)**2))
            else
               ugemo=floo/(pi*(0.25d-3)**2)
               ugemr=flor/((pi*(3.551d-3)**2)-(pi*(0.25d-3)**2))
            endif
         endif

         if (injec.eq.2) then
            flor = 8.5d-7
            if (t.lt.tob.or.t.gt.toe) then
               ugemo=0d0
               ugemr=flor/((pi*(3.551d-3)**2)-(pi*(0.25d-3)**2))
            else
               ugemo=floo/(pi*(0.25d-3)**2)
               ugemr=0d0
            endif
         endif

c**** definition of the time-dependent label boundary condition ********
c**** of curve 10

         if (t.ge.tob.and.t.le.toe) then
            label=2.0d0
         else
            label=1.0d0
         endif
c
c**** ichois = 1 and 2, boundary conditions of curves 2 and 3 **********
c
         if (ichois.eq.1) then
            funcbc=flor/(pi*((1.3d-2 + (2.9d-2-y)*dtan(alpha))**2 -
     &      (0.858d-2+(2.9d-2-y)*dtan(alpha))**2))*dsin(alpha)*
     &      4.0d0
          endif

          if (ichois.eq.2) then
            funcbc=-flor/(pi*((1.3d-2 + (2.9d-2-y)*dtan(alpha))**2 -
     &      (0.858d-2+(2.9d-2-y)*dtan(alpha))**2))*dcos(alpha)*
     &      4.0d0
          endif
c
c**** ichois = 3, boundary condition of curve 4 and 5 ******************
c
         if (ichois.eq.3) then
            funcbc=flor/(2 * pi * x * 4.0d-3) *
```

```
     &          2/pi * atan(1.8d3 * (x - 4.182d-3))
      endif
c
c**** ichois = 4 and 5, boundary conditions of curve 6 *****************
c
      if (ichois.eq.4) then
         funcbc=flor/(pi*x**2) * dsin(alpha)
      endif

      if (ichois.eq.5) then
         funcbc=-flor/(pi*x**2) * dcos(alpha)
      endif
c
c**** ichois = 6 and 7, boundary conditions of curve 7 *****************
c
      if (ichois.eq.6) then
         funcbc=flor/(pi*(2.5d-3+(4.5d-2 - y)*dtan(alpha))**2 -
     &            (0.25d-3)**2) * dsin(alpha)
      endif

      if (ichois.eq.7) then
         funcbc=-flor/(pi*(2.5d-3+(4.5d-2 - y)*dtan(alpha))**2 -
     &            (0.25d-3)**2) * dcos(alpha)
      endif
c
c**** ichois = 8, boundary condition of curve 8 ************************
c
      if (ichois.eq.8) then
         funcbc=-2d0*ugemr*(1.0d0-((x-1.901d-3)/(1.651d-3))**2)
      endif
c
c**** ichois = 9, boundary condition of curve 9 ************************
c
      if (ichois.eq.9) then
         funcbc=-flor/(pi*(2.5d-3+(4.5d-2 - y)*dtan(alpha))**2 -
     &            (0.25d-3)**2)
      endif
c
c**** ichois = 10, boundary condition of curve 10 **********************
c
      if (ichois.eq.10) then
         funcbc=-2d0*ugemo*(1.0d0-(x/(0.25d-3))**2)
      endif
c
c**** ichois = 11, boundary conditions of curves 13, 14 and 15 *********
c
      if (ichois.eq.11) then
```

```
         funcbc=flor/(2*pi*4.0d-3*(x+1.0d-5)) *
     &            2/pi * atan(1.8d3 * x)
      endif
c
c**** ichois = 12 and 13, boundary condition of curve 16 ***************
c
      if (ichois.eq.12) then
         funcbc=flor/(pi*((1.3d-2 + (2.9d-2-y)*dtan(alpha))**2 -
     &     (0.858d-2+(2.9d-2-y)*dtan(alpha))**2))*dsin(alpha)*
     &     2.0d0
      endif

      if (ichois.eq.13) then
         funcbc=-flor/(pi*((1.3d-2 + (2.9d-2-y)*dtan(alpha))**2 -
     &     (0.858d-2+(2.9d-2-y)*dtan(alpha))**2))*dcos(alpha)*
     &     2.0d0
      endif
c
c**** ichois = 14, label of curve 10 ***********************************
c
      if (ichois.eq.14) then
         funcbc=label
      endif
      end
c
c
c
      SUBROUTINE BOUNCD( kprob, itest, ibound,
     &             isol, kmeshm, ihlp)
c
c input/output parameters
c
      implicit none

      integer kmeshm(*), kprob(*), itest(5,*), isol(5,*), ibound(5,*),
     &        ihlp(5,*)
c
c common blocks
c
      integer irefwr, irefre, irefer
      common /cmcdpi/ irefwr, irefre, irefer
      save /cmcdpi/
c
      integer ibuffr
      common ibuffr(20 000 000)
c
      integer nbuffr, kbuffr, intlen, ibfree
```

```
      common /cbuffr/ nbuffr, kbuffr, intlen, ibfree
      save /cbuffr/
c
      integer iinfor, infor
      common /carray/ iinfor, infor(3,1500)
      save /carray/

      double precision theta, dt, t, ratime
      integer irtime, nstep
      common /ctima/ theta, dt, t, ratime(7),
     &               nstep, irtime(9)
      save /ctima/
c
      double precision trb, tre, tob, toe, injec
      common /dprotim/ trb, tre, tob, toe, injec
c
c local variables
c
      integer ino, ndegfd, jno, n, ncrv, nnocrv(100), i, idum

      double precision bound(5), test(5), test2(5), test3(5),
     &                 dif


      call eropen('bouncd')

      do 25 i = 1, 5
        test3(i) = 0d0
        test2(i) = 0d0
  25  continue

      ncrv = kmeshm(2) - 6 - kmeshm(1)
c        write(irefwr,*) 'ncrv =', ncrv
      do 21 n = 1, ncrv
        nnocrv(n) = kmeshm(5 + n) - kmeshm(4 + n)
c        write(irefwr,*) 'nnocrv =', nnocrv(n)
  21  continue
      do 31 n = 1, ncrv
        if (n.eq.8.or.n.eq.10.or.n.eq.11.or.n.eq.12) then
           dif = 0d0
        else
           if (n.eq.1) then
              dif = 0d0
           else
              dif = 1d0
           endif
        endif
```

```
c         write(irefwr,*) ' dif0 =', dif
        if (dif .ne. 0d0) then
          do 22 ino = 1 , nnocrv(n)
          jno = kmeshm(kmeshm(2) + kmeshm(4+n) - 1 + ino - 1)
c          write(irefwr,*) 'jno0 =', jno
           ndegfd = 2
           call getvec(kprob, ibound(1,1), jno, bound, ndegfd)
           ndegfd = 1
           call getvec(kprob, itest(1,1), jno, test, ndegfd)
             test2(1) = bound(1) * test(1)
             test2(2) = bound(2) * test(1)
c           write(irefwr,*) ' test2(1)0 = ', test2(1)
c           write(irefwr,*) ' test2(2)0 = ', test2(2)
           call putvec(kprob, isol(1,1), jno, test2, idum)
           call putvec(kprob, ihlp(1,1), jno, test, idum)
   22    continue
        endif
   31 continue

      call erclos ('bouncd')
      end
c
c
c
      SUBROUTINE BOUNCD1( kprob, itest, ibound,
     &             isol, kmeshm, ihlp)
c
c input/output parameters
c
      implicit none

      integer kmeshm(*), kprob(*), itest(5,*), isol(5,*), ibound(5,*),
     &        ihlp(5,*)
c
c common blocks
c

      integer irefwr, irefre, irefer
      common /cmcdpi/ irefwr, irefre, irefer
      save /cmcdpi/
c
      integer ibuffr
      common ibuffr(20 000 000)
c
      integer nbuffr, kbuffr, intlen, ibfree
      common /cbuffr/ nbuffr, kbuffr, intlen, ibfree
      save /cbuffr/
```

```
c
      integer iinfor, infor
      common /carray/ iinfor, infor(3,1500)
      save /carray/
c
      double precision theta, dt, t, ratime
      integer irtime, nstep
      common /ctima/ theta, dt, t, ratime(7),
     &               nstep, irtime(9)
      save /ctima/
c
      double precision trb, tre, tob, toe, injec
      common /dprotim/ trb, tre, tob, toe, injec
c
c local variables
c
      integer ino, ndegfd, jno, n, ncrv, nnocrv(100), i, idum

      double precision bound(5), test(5), test2(5),
     &                 dif

      call eropen('bouncd1')

      do 25 i = 1, 5
        test2(i) = 0d0
  25  continue

      ncrv = kmeshm(2) - 6 - kmeshm(1)
c        write(irefwr,*) 'ncrv =', ncrv
      do 21 n = 1, ncrv
        nnocrv(n) = kmeshm(5 + n) - kmeshm(4 + n)
c        write(irefwr,*) 'nnocrv =', nnocrv(n)
  21  continue
      do 31 n = 1, ncrv
        if (n.eq.8.or.n.eq.10.or.n.eq.11.or.n.eq.12) then
           dif = 0d0
        else
           if (n.eq.1) then
              dif = 0d0
           else
              dif = 1d0
           endif
        endif
c       write(irefwr,*) ' dif1 =', dif
      if (dif .ne. 0d0) then
        do 22 ino = 1 , nnocrv(n)
        jno = kmeshm(kmeshm(2) + kmeshm(4+n) - 1 + ino - 1)
```

```
c         write(irefwr,*) 'jno1 =', jno
          ndegfd = 2
          call getvec(kprob, ibound(1,1), jno, bound, ndegfd)
          ndegfd = 1
          call getvec(kprob, ihlp(1,1), jno, test, ndegfd)
            test2(1) = bound(1) * test(1)
            test2(2) = bound(2) * test(1)
c            write(irefwr,*) 'test2(1)1 = ', test2(1)
c            write(irefwr,*) 'test2(2)1 = ', test2(2)
          call putvec(kprob, isol(1,1), jno, test2, idum)
   22   continue
        endif
   31 continue

        call erclos ('bouncd1')
        end
c
c
c ***********************************************************************
cdccgetvec
      subroutine getvec( kprob, ivec, ino, vec, ndegfd )
c
c ***********************************************************************
c
c        programmer   Leo Caspers
c        version   1.1    date   03-08-94  (LC: complete revision)
c        version   1.0    date   20-11-91
c
c   copyright (c) 1991  "VIp"
c   permission to copy or distribute this software or documentation
c   in hard copy or soft copy granted only by written license
c   obtained from "VIp".
c   all rights reserved. no part of this publication may be reproduced,
c   stored in a retrieval system ( e.g., in memory, disk, or core)
c   or be transmitted by any means, electronic, mechanical, photocopy,
c   recording, or otherwise, without written permission from the
c   publisher.
c
c ***********************************************************************
c
c     Store all dofs of array isol for nodal point ino into vec.
c     Store # dofs into ndegfd
c     Only allow vector types 110, 115
c     All dofs must be doubles
c     Length of vec on input is stored in ndegfd
c
c ***********************************************************************
```

```
c
c               INPUT / OUTPUT PARAMETERS

      implicit none

      integer ino, ndegfd

      integer kprob(*), ivec(5)
      double precision vec(*)

c
c     kprob     i      standard sepran array
c     ivec      i      standard sepran array
c     ino       i      node number of which value is desired
c     ndegfd    i/o    input  : maximum allowed
c                      output : actual length
c     vec       o      values of ivec in ino
c
c ***************************************************************************
c
c               PARAMETERS IN COMMON BLOCKS
c
      integer ibuffr
      common ibuffr(1)
c
c                /ibuffr/
c     blank common
c
c     ibuffr(1)      array for storage of large int. and dp. arrays
c
c  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      integer iinfor,infor
      common /carray/ iinfor,infor(3,1500)
c
c                /carray/
c     information about ibuffr
c
c     iinfor         gives number of types stored in infor
c     infor(..)      information of arrays stored in ibuffr
c
c  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      integer nbuffr, kbuffr, intlen, ibfree
      common /cbuffr/ nbuffr, kbuffr, intlen, ibfree
      save /cbuffr/

c               /cbuffr/
c    Several variables related to the common buffer ibuffr
```

```
c
c    nbuffr     Declared length of array ibuffr
c    kbuffr     Last position used in ibuffr
c    intlen     Length of a real variable divided by the length of
c               an integer variable
c    ibfree     Next free position in array ibuffr
c
c  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      integer irefwr,irefre,irefer
      common /cmcdpi/ irefwr,irefre,irefer
c
c                 /cmcdpi/
c     unit numbers of i/o files
c
c     irefwr         unit number standard output
c     irefre         unit number standard input
c     irefer         unit number standard error file
c
c  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
c ***********************************************************************
c
c               LOCAL PARAMETERS

      integer indprf, indprh, indprp, nphys, ipvec, ipkprf, ndgfdm,
     &        ipkprh

c
c     #name    #i/o  #explanation
c
c ***********************************************************************
c
c               SUBROUTINES CALLED
c
c      EROPEN    #explanation
c      INI055    #explanation
c      WRVEC     #explanation
c      ERCLOS    #explanation
c
c ***********************************************************************
c
      call eropen ( 'getvec' )

c Save max of ndegfd

      ndgfdm = ndegfd

c Get pointers of kprobf, kprobh, kprobp, isol
```

```
      call ini075( ivec, kprob, indprf, indprh, indprp, nphys)

      ipkprf = max( 1, infor(1,indprf) )
      if( indprf .gt. 0 )then
         ndegfd=ibuffr(ipkprf+ino)-ibuffr(ipkprf+ino-1)
      else
         ndegfd=nphys
      endif

c Check max ndegfd

      if ( ndegfd .lt. 0 .or. ndegfd .gt. ndgfdm ) then
         write(irefwr,*)'getvec: number of degrees of freedom is',
     &                  ndegfd
         write(irefwr,*)'        which is greater than the maximum'
         write(irefwr,*)'        allowed ', ndgfdm, ' or less than zero'
         call instop
      endif

      if( ndegfd .eq. 0 )goto 1000

      ipkprh = max( 1, infor(1,indprh) )
      ipvec=infor(1,ivec(1))

c Do the copying

      call vput01( ibuffr(ipvec), ibuffr(ipkprh), ibuffr(ipkprf),
     &             indprh, indprf, ndegfd, ino, vec, 1 )

1000  call erclos ( 'getvec' )

      end
cdc*eor

cdccputvec
      subroutine putvec( kprob, ivec, ino, vec, idum )
c
c ***********************************************************************
c
c        programmer   Leo Caspers
c        version   1.2    date   02-10-94  (LC: add dummy argument)
c        version   1.1    date   03-08-94  (LC: complete revision)
c        version   1.0    date   20-11-91
c
c   copyright (c) 1991  "VIp"
c   permission to copy or distribute this software or documentation
```

```
c   in hard copy or soft copy granted only by written license
c   obtained from "VIp".
c   all rights reserved. no part of this publication may be reproduced,
c   stored in a retrieval system ( e.g., in memory, disk, or core)
c   or be transmitted by any means, electronic, mechanical, photocopy,
c   recording, or otherwise, without written permission from the
c   publisher.
c
c ***********************************************************************
c
c     Store all dofs of array isol for nodal point ino into vec.
c     Only allow vector types 110, 115
c     All dofs must be doubles
c
c ***********************************************************************
c
c               INPUT / OUTPUT PARAMETERS

      implicit none

      integer ino, idum

      integer kprob(*), ivec(5)
      double precision vec(*)

c
c     kprob     i     standard sepran array
c     ivec      i     standard sepran array
c     ino       i     node number of which value is desired
c     vec       o     values of ivec in ino
c     idum      i     dummy for historical reasons
c
c ***********************************************************************
c
c               PARAMETERS IN COMMON BLOCKS
c
      integer ibuffr
      common ibuffr(1)
c
c                 /ibuffr/
c     blank common
c
c     ibuffr(1)       array for storage of large int. and dp. arrays
c
c  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      integer iinfor,infor
      common /carray/ iinfor,infor(3,1500)
```

```
c
c                /carray/
c     information about ibuffr
c
c     iinfor         gives number of types stored in infor
c     infor(..)      information of arrays stored in ibuffr
c
c  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      integer nbuffr, kbuffr, intlen, ibfree
      common /cbuffr/ nbuffr, kbuffr, intlen, ibfree
      save /cbuffr/

c                /cbuffr/
c    Several variables related to the common buffer ibuffr
c
c    nbuffr     Declared length of array ibuffr
c    kbuffr     Last position used in ibuffr
c    intlen     Length of a real variable divided by the length of
c               an integer variable
c    ibfree     Next free position in array ibuffr
c
c  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      integer irefwr,irefre,irefer
      common /cmcdpi/ irefwr,irefre,irefer
c
c                /cmcdpi/
c     unit numbers of i/o files
c
c     irefwr         unit number standard output
c     irefre         unit number standard input
c     irefer         unit number standard error file
c
c  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
c ***********************************************************************
c
c               LOCAL PARAMETERS

      integer indprf, indprh, indprp, nphys, ipvec, ipkprf,
     &        ipkprh, ndegfd

c
c     #name    #i/o  #explanation
c
c ***********************************************************************
c
c               SUBROUTINES CALLED
c
```

```
c      EROPEN    #explanation
c      INI075    #explanation
c      ERCLOS    #explanation
c
c ************************************************************************
c
      call eropen ( 'putvec' )

c Get pointers of kprobf, kprobh, kprobp, isol

      call ini075( ivec, kprob, indprf, indprh, indprp, nphys)

      ipkprf = max( 1, infor(1,indprf) )
      if( indprf .gt. 0 )then
         ndegfd=ibuffr(ipkprf+ino)-ibuffr(ipkprf+ino-1)
      else
         ndegfd=nphys
      endif

      if( ndegfd .eq. 0 )goto 1000

      ipkprh = max( 1, infor(1,indprh) )
      ipvec=infor(1,ivec(1))

c Do the copying

      call vput01( ibuffr(ipvec), ibuffr(ipkprh), ibuffr(ipkprf),
     &             indprh, indprf, ndegfd, ino, vec, 2 )

1000  call erclos ( 'putvec' )

      end
cdc*eor

cdccvput01
      subroutine vput01( vector, kprobh, kprobf, indprh, indprf,
     &                   ndegfd, ino, vecusr, ichois )
c ======================================================================
c
c        programmer     Leo Caspers
c        version 1.0    date   23-06-94  (First draft)
c
c   copyright (c) 1993  "VIp"
c   permission to copy or distribute this software or documentation
c   in hard copy or soft copy granted only by written license
c   obtained from "VIp".
c   all rights reserved. no part of this publication may be reproduced,
```

```
c   stored in a retrieval system ( e.g., in memory, disk, or core)
c   or be transmitted by any means, electronic, mechanical, photocopy,
c   recording, or otherwise, without written permission from the
c   publisher.
c
c ***********************************************************************
c
c                          DESCRIPTION
c
c        # explain subroutine
c
c ***********************************************************************
c
c                       KEYWORDS
c
c ***********************************************************************
c
c                       INPUT / OUTPUT    PARAMETERS
c
      implicit none
      integer ndegfd, ino, indprh, indprf, ichois
      integer kprobf(*), kprobh(*)
      double precision vector(*), vecusr(*)

c
c     ichois    i   1: called by getvec: copy vector to vecusr
c                   2: called by putvec: copy vecusr to vector
c
c ***********************************************************************
c
c                       COMMON BLOCKS
c    -
c ***********************************************************************
c
c                     LOCAL PARAMETERS
c
      integer iunder, i
c
c     iunder  i   relative pointer to 1st dof of node ino
c ***********************************************************************
c
c              SUBROUTINES CALLED
c
c     ERCLOS:  Resets old name of previous subroutine of higher level
c     EROPEN:  Produces concatenated name of local subroutine
c     ERRSUB:  Error messages
c
```

```
c ***********************************************************************
c
c                    I/O
c    none
c ***********************************************************************
c
c                    ERROR MESSAGES
c   -
c ***********************************************************************
c
c                    PSEUDO CODE
c
c   trivial
c =======================================================================
c
      call eropen ( 'vput01' )

      if( indprf .gt. 0 )then
         iunder = kprobf(ino)+1
      else
         iunder = (ino-1)*ndegfd+1
      endif

      if( ichois .eq. 1 )then

c Called by getvec

         if( indprh .eq. 0 )then
            do 10 i = 1, ndegfd
               vecusr(i) = vector(iunder+i-1)
10          continue
         else
            do 20 i = 1, ndegfd
               vecusr(i) = vector(kprobh(iunder+i-1))
20          continue
         endif
      else

c Called by putvec

         if( indprh .eq. 0 )then
            do 30 i = 1, ndegfd
               vector(iunder+i-1) = vecusr(i)
30          continue
         else
            do 40 i = 1, ndegfd
               vector(kprobh(iunder+i-1)) = vecusr(i)
```

```
40          continue
         endif
      endif

      call erclos ( 'vput01' )
      end
cdc*eor
```

B.2 Input for meshgeneration

```
mesh2d
points
p1 = (1.205d-2 , 0)
p2 = (1.605d-2 , 0)
p3 = (1.342d-2 , 2.5d-2)
p4 = (1.3d-2 , 2.9d-2)
p5 = (8.58d-3 , 2.9d-2)
p6 = (4.182d-3 , 2.9d-2)
p7 = (3.761d-3 , 3.3d-2)
p8 = (3.551d-3 , 3.5d-2)
p9 = (0.25d-3 , 3.5d-2)
p10 = (0.25d-3 , 3.3d-2)
p11 = (0 , 3.3d-2)
p12 = (0 , 2.9d-2)
p13 = (0 , 2.5d-2)
p14 = (0.25d-3 , 2.5d-2)
p15 = (4.602d-3 , 2.5d-2)
p16 = (9.0d-3 , 2.5d-2)
p17 = (0.25d-3 , 2.9d-2)
curves
c1 = line2(p1, p2, nelm=8)
c2 = line2(p2, p3, nelm=50)
c3 = line2(p3, p4, nelm=8)
c4 = line2(p4, p5, nelm=8)
c5 = line2(p5, p6, nelm=10)
c6 = line2(p6, p7, nelm=8)
c7 = line2(p7, p8, nelm=4)
c8 = line2(p8, p9, nelm=8)
c9 = line2(p9, p10, nelm=4)
c10 = line2(p10, p11, nelm=2)
c11 = line2(p11, p12, nelm=8)
c12 = line2(p12, p13, nelm=8)
c13 = line2(p13, p14, nelm=2)
c14 = line2(p14, p15, nelm=8)
c15 = line2(p15, p16, nelm=10)
c16 = line2(p16, p1, nelm=50)
c17 = line2(p16, p3, nelm=8)
c18 = line2(p5, p16, nelm=8)
c19 = line2(p6, p17, nelm=8)
c20 = line2(p17, p12, nelm=2)
c21 = line2(p7, p10, nelm=8)
c22 = line2(p10, p17, nelm=8)
c23 = line2(p17, p14, nelm=8)
c24 = line2(p6, p15, nelm=8)
```

```
surfaces
s1 = rectangle4(8,50,c1,c2,-c17,c16)
s2 = rectangle4(8,8,c17,c3,c4,c18)
s3 = rectangle4(8,10,-c18,c5,c24,c15)
s4 = rectangle4(8,8,-c24,c19,c23,c14)
s5 = rectangle4(8,2,-c23,c20,c12,c13)
s6 = rectangle4(8,8,-c19,c6,c21,c22)
s7 = rectangle4(2,8,-c20,-c22,c10,c11)
s8 = rectangle4(8,4,-c21,c7,c8,c9)
meshsurf
selm1=(s1,s8)
plot(yfact=0.8)
renumber
end
```

B.3 Input for the solver

```
problem 1
   types
      elgrp1=(type=404)                    # Type number for Navier Stokes
   essbouncond
      degfd1=curves0(c2,c16)
      degfd2=curves0(c2,c10)
      degfd2=curves0(c13,c16)
problem 2
   types
      elgrp1=(type=800)                    # Type number for label
   essbouncond
      degfd1=curves0(c8)
      degfd1=curves0(c10)
end
create vector 3, problem 1               # Fill array ivcold
   type = vector of special structure 2
   value = 1.0d-1
end
create vector 1, problem 1               # Fill array isol
   type = solution vector
   value = 0d0                           # initial velocities
create vector 2, problem 2
   type = solution vector
   value = 0d0                           # initial label
create vector 3, problem 1
   type = vector of special structure 2
   value = 1.0d-1                        # viscosity
end
essential boundary conditions 1
   curves0(c2),degfd1=(func=1)
   curves0(c2),degfd2=(func=2)
   curves0(c3),degfd1=(func=1)
   curves0(c3),degfd2=(func=2)
   curves0(c4),degfd1=(func=3)
   curves0(c5),degfd1=(func=3)
   curves0(c6),degfd1=(func=4)
   curves0(c6),degfd2=(func=5)
   curves0(c7),degfd1=(func=6)
   curves0(c7),degfd2=(func=7)
   curves0(c8),degfd2=(func=8)
   curves0(c9),degfd2=(func=9)
   curves0(c10),degfd2=(func=10)
   curves0(c13),degfd1=(func=11)
   curves0(c14),degfd1=(func=11)
   curves0(c15),degfd1=(func=11)
```

```
   curves0(c16),degfd1=(func=12)
   curves0(c16),degfd2=(func=13)
essential boundary conditions 2
   curves0(c8),degfd1=(value=1.0d0)
   curves0(c10),degfd1=(func=14)
end
*  Definition of coefficients for impuls balance:
coefficients 1
   elgrp1 ( nparm=8 )          # The coefficients are defined by 8 parameters
      coef1  =(value=1d-15)    #  1:  Penalty function parameter eps
      coef2  =(value=1.0d3)    #  2:  Density
      icoef3 =11               #  3:  Type of linearization (10=Stokes flow)
                               #                            (11=Picard)
                               #                            (12=Newton)
                               #  4:  angular velocity = 0
                               #  5:  body force in x-direction = 0
                               #  6:  body force in y-direction = 0
      icoef7 =1                #  7:  type of constitutive equation(1=Newtonian)
                               #                                   (2=powerlaw )
      coef8  =(old solution 3)
                               #  8:  eta
end
* Definition of coefficients for label:
* Coefficients according to element 800.
coefficients 2
   elgrp1 ( nparm=20 )         # The coefficients are defined by 20 parameters
      icoef1=0                 # Not yet used (must be 0)
      icoef2=1                 # Type of upwinding (1 = classical scheme)
      icoef3=0                 # Type of numerical integration (default
                               #                                  value)
      icoef4=1                 # Type of coordinates (axi-symmetric)
      icoef5=0                 # Not yet used (must be 0)
      coef6=(value=1d-12)      # Alpha11 = diffusion
      coef7=(value=0d0)        # Alpha12
      coef8=(value=0d0)        # Alpha13
      coef9=(value=1d-12)      # Alpha22 = diffusion
      coef10=(value=0d0)       # Alpha23
      coef11=(value=0d0)       # Alpha33
      coef12=(old solution 1, degree of freedom 1)
                               # u1 (radial velocity)
      coef13=(old solution 1, degree of freedom 2)
                               # u2 (axial velocity)
      coef14=(value=0d0)       # u3 (not used in this problem)
      coef15=(value=0d0)       # beta
      coef16=(value=0d0)       # f (right hand side)
      coef17=(value=1d0)       # coefficient in front of mass matrix
      icoef18=0                # not used, must be zero
```

```
      icoef19=0              # not used, must be zero
      icoef20=0              # not used, must be zero
end
output
write 3 solutions
end
```

B.4 Input for postprocessing

```
set warn on                              # display warnings (on/off)
set time on                              # display CPU time (off/on)
set output on                            # display all information (off/on/out)
start
   database = not                        # use file2 (not/new/old)
   norotate                              # rotate plots (norotate/rotate)
   renumber sloan band                   # renumber
   sepcomp = formatted                   # file format for sepcomp.out
end
postprocessing
name v0=velocity
name v1=label
name v2=dynamic viscosity
time=(0.0d0,1.6d1,1)
#plot vector v0, factor=0.01, yfact=0.8d0
plot contour v1, yfact=0.8d0, levels=(0.5, 1.5)
#plot coloured levels v1, yfact=0.8d0, levels=(0.0, 0.1, //
#             0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0)
#plot coloured levels v1, yfact=0.8d0, levels=(0.0, 0.5, 1.5, 2.0)
end
```

Appendix C Slipvelocities

The slipvelocities are difficult to define for the complex geometry used. In general the mean velocity u_m of the front can be calculated as follows:

$$u_m = \frac{\phi}{A} \tag{15}$$

with: ϕ the volumeflow through the pipe
A the surface perpendicular to the direction of the flow

There are two simple examples at which the mean velocity can be calculated simply.

1. Flow through a pipe. The mean velocity is equal to

$$u_m = \frac{\phi}{\pi R^2} \tag{16}$$

with: R the radius of the pipe

2. Flow between two plates out of a point-source. The mean velocity is equal to

$$u_m = \frac{\phi}{2\pi r d} \tag{17}$$

with: r the radius from a certain point to the point-source
d the distance between the two plates

These two simple examples have been taken into account at the choice for the slipvelocities at the walls. At the boundaries Γ_2 and Γ_{10} the slipvelocities decrease with decreasing z, because the surface increases with decreasing z (see Figure 14). For the same reason the slipvelocities of boundaries Γ_6 and Γ_8 decrease with decreasing z (see Figure 16). The slipvelocities of boundaries Γ_5 and Γ_9 decrease with increasing r, because the surface increases with increasing r. For a better simulation both functions have been multiplied with a steep arctangens (see Figure 15). On boundary Γ_3 no slipvelocity has to be defined, because on this boundary there is no air for simplicity reasons. In the labelplot can be seen whether the slipvelocities are defined right or wrong. When the slipvelocities are defined wrong, the rubberfront has a strange shape.

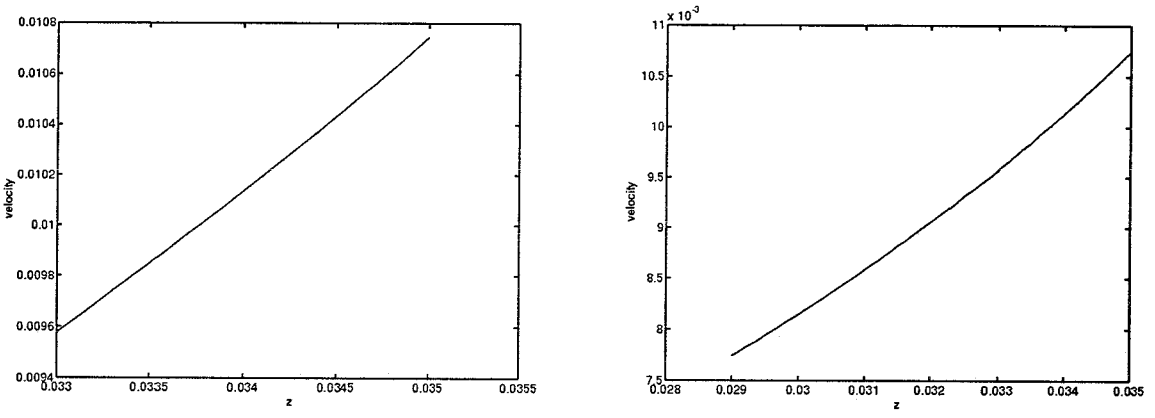

Figure 14: Slipvelocity near boundaries Γ_2 (left plot) and Γ_{10} (right plot)

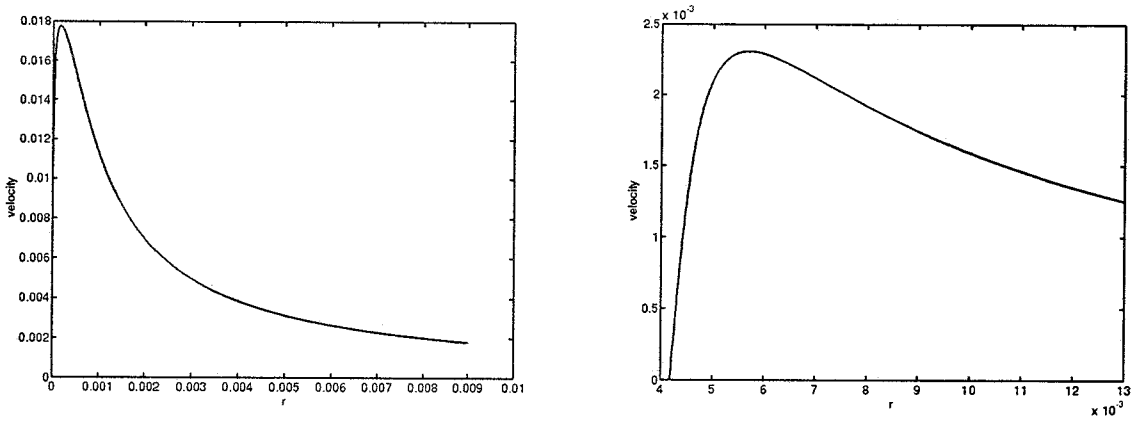

Figure 15: Slipvelocity near boundaries Γ_5 (left plot) and Γ_9 (right plot)

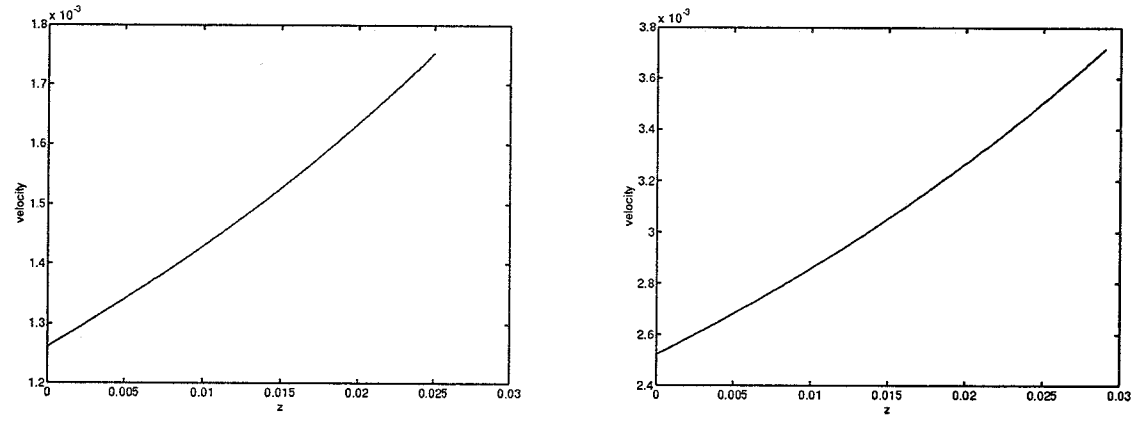

Figure 16: Slipvelocity near boundaries Γ_6 (left plot) and Γ_8 (right plot)

Appendix D Upwinding

An improvement of the accuracy may be possible by applying a so-called upwind technique. Upwinding improves not always the accuracy. The building of matrices in case of upwinding is more expensive than in the standard case. The upwinding in SEPRAN is realised by the streamline upwind Petrov-Galerkin method (SUPG), see Brooks and Hughes (1982). Essential in this method is that next to the standard Galerkin equation an extra term of the following type is added:

$$\int_e (Dc - f)pd\Omega \tag{18}$$

where e is the element. Dc represents the differential equation applied to c and f is the right-hand side. The upwind parameter p is defined by

$$p_i = \frac{h\xi}{2} \frac{\mathbf{u} \cdot \nabla \phi_\mathrm{i}}{\|\mathrm{u}\|} \tag{19}$$

with: h the width of the element in the direction of the flow,
ϕ_i the i^{th} basis function,
$\mathbf{u}$ the velocity, and
ζ a choice parameter defining the type of upwinding.

References

[1] Geus, A.F.R. de
On the design and manufacturing of oil-filled rubber dampers.
Technical report WFW 94.076, Eindhoven University of Technology, 1994

[2] Zoetelief, W.F.
Multi-component Injection Moulding.
Ph.D. thesis, ISBN 90-386-0016-X, Eindhoven University of Technology, 1995

[3] Segal, A.
SEPRAN, user manual, standard problems and programming guide.
Ingenieursbureau SEPRA, Leidschendam, The Netherlands, 1984

[4] Steenhoven, A.A. van
De toepassing van de eindige elementenmethode bij stromingsproblemen.
lecture notes, Eindhoven University of Technology, The Netherlands, 1984