ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Introduction to Hyperbolic Equations and Fluid-Structure Interaction

Semester Project
of
Benjamin Stamm

Directed by:
Prof. Alfio Quarteroni
Simone Deparis

Institute of Analysis and Scientific Computing, EPFL, Lausanne

March 10, 2003

Contents

1 Introduction

In this semester project, we dealt with hyperbolic partial differential equations and Fluid-Structure Interaction.

Chapter 2 is a short introduction in the wave equation. It's the equation which describes the propagation of waves. In the last chapter, we use this equation to describe a one dimensional vessel in a simple fluid-structure interaction problem.

In Chapter 3 the Newmark method is studied in detail. The two cases of different boundary conditions, Dirichlet and Neuman, has to be treated differently, since the weak problem is not the same. In both cases, the weak problem of the wave equation is derived. Then we use the Galerkin method to discretise the space. This leads us to a second order ordinary differential equation, whose solution is approximated with the Newmark method. A Matlab code is developed from the theoretical description. The description of the code is more an illustration in some particular passages. You will find the code at the end of this report. In the fluid-structure interaction, we use the Newmark method to simulate the vessel.

In the following chapter, the Leap-Frog method is studied. It's also a numerical method to approximate the solution of the wave equation. You can also find the Matlab code at the end of this report. This method can be used in the fluid-structure problem, but we did not.

In chapter 5, a simple model of blood flow, based on the fluid-structure interaction is developed. An algorithm is presented, which couples the fluid and the vessel-part. Then, you can find the results of two different versions of the algorithm and a short analysis.

A short discussion of the project is given in the chapter 6.

In chapter 7, you find the Matlab codes of the Newmark- and Leap-Frog-method. You can not find the codes for the simulation of the fluid-structure interaction problem, because all the codes needed to run the program are too big to present it in this report.

2 The wave equation

2.1 The mathematical problem

This is a short introduction to the wave equation. It describes many wave phenomenons, like the string-, electro-magnetic- or acoustic-waves. The following mathematical problem (P) has to be studied: find $u : Q \to \mathbb{R}$ satisfying

$$\frac{\partial^2 u}{\partial t^2} - a \cdot \frac{\partial^2 u}{\partial x^2} + c \cdot u = f \tag{1}$$

with initial conditions

$$u(x,0) = u_0(x) \tag{2}$$

$$\frac{\partial}{\partial t}u(x,0) = u_1(x) \tag{3}$$

and boundary conditions. Note that Q is the space-time domain and that the space has dimension one. So we have $Q = (x_0, x_1) \times (a_t, b_t)$, (x_0, x_1) being the space-domain and (a_t, b_t) the time-domain. We also have $f : Q \to \mathbb{R}$, whereas a and c are two constants. We suppose that $\frac{\partial^2 u}{\partial t^2}$ and $\frac{\partial^2 u}{\partial x^2}$ exist.

2.2 Dirichlet boundary conditions

We say that the problem has Dirichlet boundary conditions when we know that

$$u(x_0, t) = bc_0(t) \qquad u(x_1, t) = bc_1(t) \tag{4}$$

where $bc_0, bc_1 : [a_t, b_t] \to \mathbb{R}$. Note that it is necessary that $u_0(x_0) = bc_0(t_0)$ and $u_0(x_1) = bc_1(t_0)$ to get a well posed problem.

2.3 Neumann boundary conditions

With Neumann boundary conditions, the spatial derivative on the boundary is given:

$$\frac{\partial u}{\partial x}(x_0, t) = bc_0(t) \qquad \frac{\partial u}{\partial x}(x_1, t) = bc_1(t) \tag{5}$$

where $bc_0, bc_1 : [a_t, b_t] \to \mathbb{R}$.

3 Numerical approximation by the Newmark method

3.1 Dirichlet boundary conditions

3.1.1 The weak problem

In this section, we would like to find a problem equivalent to (P), which is simpler (to treat and) to solve. We would like only first order derivatives in space instead of a second order one.
For this, we multiply the equation (1) by a test function $v : [x_0.x_1] \to \mathbb{R}$, $v \in V_0 := H_0^1(x_0, x_1)$ and integrate the obtained equation in space from x_0 to x_1. We obtain the following equation:

$$\begin{aligned}\int_{x_0}^{x_1} \frac{\partial^2}{\partial t^2}u(x,t)\cdot v(x)dx - a\cdot\int_{x_0}^{x_1}\frac{\partial^2}{\partial x^2}u(x,t)\cdot v(x)dx + c\cdot\int_{x_0}^{x_1}u(x,t)\cdot v(x)dx \\ = \int_{x_0}^{x_1} f(x,t)\cdot v(x)dx \end{aligned} \quad (6)$$

Integrating by parts gives

$$\int_{x_0}^{x_1}\frac{\partial^2}{\partial x^2}u(x,t)\cdot v(x)dx = \frac{\partial}{\partial x}u(x,t)\cdot v(x)|_{x_0}^{x_1} - \int_{x_0}^{x_1}\frac{\partial}{\partial x}u(x,t)\cdot\frac{\partial}{\partial x}v(x)dx \quad (7)$$

Since x_0 and x_1 do not depend on t, we can change the second derivative in time and the integral. We have $v(x_0) = 0 = v(x_1)$, because $v \in V_0$. This leads to an equivalent problem to (P): $\forall t > 0$ find $u(t) \in V := H^1(x_0, x_1)$ satisfying

$$\begin{aligned}\frac{d^2}{dt^2}\int_{x_0}^{x_1}u(x,t)\cdot v(x)dx + a\cdot\int_{x_0}^{x_1}\frac{\partial}{\partial x}u(x,t)\frac{\partial}{\partial x}v(x)dx + c\cdot\int_{x_0}^{x_1}u(x,t)\cdot v(x)dx \\ = \int_{x_0}^{x_1} f(x,t)\cdot v(x)dx \end{aligned} \quad (8)$$

for all $v \in V_0 := H_0^1(x_0, x_1)$. To simplify the notation we use the scalar product in $L^2(x_0, x_1)$

$$(f, g)_{L^2} = \int_{x_0}^{x_1} f(x)\cdot g(x)dx \quad (9)$$

We also can define the following bilinear forms:

$$a : H^1(x_0, x_1)\times H^1(x_0, x_1) \to \mathbb{R} \qquad c : L^2(x_0, x_1)\times L^2(x_0, x_1) \to \mathbb{R} \quad (10)$$

$$a(u, v) = a\cdot\int_{x_0}^{x_1}\frac{\partial}{\partial x}u(x)\frac{\partial}{\partial x}v(x)dx \quad (11)$$

$$c(u, v) = c\cdot(u, v)_{L^2} \quad (12)$$

Lemma 1 *The bilinear form $a(\cdot,\cdot)$ is continuous.*

proof 1 *We apply the Cauchy-Schwarz inequality:*

$$\mid a(u,v)\mid = \mid (\frac{\partial u}{\partial x}, \frac{\partial v}{\partial x})_{L^2}\mid \leq \mid u\mid_{H^1}\cdot\mid v\mid_{H^1} < \infty$$

because $u, v \in H^1(x_0, x_1)$. $\square$

Finally, we obtain with this notations the weak problem (WP) of (P):
find $u \in V$

$$\frac{d^2}{dt^2}(u, v)_{L^2} + a(u, v) + c(u, v) = (f, v)_{L^2} \quad (13)$$

$$\begin{aligned} u(x,0) &= u_0(x) \\ \frac{\partial}{\partial t}u(x,0) &= u_1(x) \\ u(x_0,t) &= bc_0(t) \\ u(x_1,t) &= bc_1(t) \end{aligned}$$

for all $v \in V_0$.

3.1.2 Space-discretization with the Galerkin method

In this section, we search a semi-discrete approximation of the weak problem (WP) using the Galerkin method. This leads us to a second order Cauchy-problem in time.

Let V_h be a $N_x + 1$ dimensional subspace of V and $V_{0,h} = V_h \cap V_0$. Then the following problem is an approximation of the weak problem (WP).
Find a function $u_h \in V_h$ that satisfies:

$$\frac{d^2}{dt^2}(u_h, v_h) + a(u_h, v_h) + c(u_h, v_h) = (f, v_h) \tag{14}$$

$$\begin{aligned} u_h(x,0) &= u_{0,h}(x) \\ \frac{\partial}{\partial t}u_h(x,t) &= u_{1,h}(x) \\ u_h(x_0,t) &= bc_0(t) \\ u_h(x_{01},t) &= bc_1(t) \end{aligned}$$

for all $v \in V_{0,h}$, where $u_{0,h}$ and $u_{1,h}$ are projections from V in V_h of u_0 resp. u_1.

The choice of V_h is completely arbitrary. So we can choose it the way, that for later treatment, it will be as easy as possible. For example, we subdivide the interval $[x_0, x_1]$ into partitions of equal distances h:
$x_0 = a_1 < a_2 < \cdots < a_{N_x} < a_{N_x+1} = x_1 \qquad a_i = x_0 + (i-1) \cdot h$

$$\begin{aligned} V_h &= \{v_h \in C^0([x_0, x_1]) : v_h \mid_{[a_i, a_{i+1}]} \in \mathbb{P}_1, \forall i = 1 \cdots N_x\} \\ V_{0,h} &= \{v_h \in V_h : v_h(x_0) = v_h(x_1) = 0\} \end{aligned}$$

Note, that the finite dimension allows us to build a finite base for the corresponding space. In the case of $V_{0,h}$ we have: $\{\varphi_i\}_{i=2}^{N_x}$ where

$$\forall i = 2 \cdots N_x \qquad \varphi_i(x) = \begin{cases} 0 & \text{if } x \in [a_1, a_{i-1}] \\ \frac{x - x_0}{h} - (i-2) & \text{if } x \in [a_{i-1}, a_i] \\ \frac{x_0 - x}{h} + i & \text{if } x \in [a_i, a_{i+1}] \\ 0 & \text{if } x \in [a_{i+1}, a_{N_x+1}] \end{cases}$$

while we add for V_h the two functions φ_1 and φ_{N_x+1} defined as:

$$\varphi_1(x) = \begin{cases} \frac{x_0 - x}{h} + 1 & \text{if } x \in [a_1, a_2] \\ 0 & \text{if } x \in [a_2, a_{N_x+1}] \end{cases}$$

$$\varphi_{N_x+1}(x) = \begin{cases} 0 & \text{if } x \in [a_1, a_{N_x}] \\ \frac{x - x_0}{h} - N_x + 1 & \text{if } x \in [a_{N_x}, a_{N_x+1}] \end{cases}$$

So that we can write u_h as a linear combination of the basic elements:

$$\begin{aligned} u_h(x,t) &= \sum_{j=1}^{N_x+1} \tilde{u}_j(t)\cdot\varphi_j(x) \\ u_{0,h}(x) &= \sum_{j=1}^{N_x+1} u_0(x_j)\cdot\varphi_j(x) \\ u_{1,h}(x) &= \sum_{j=1}^{N_x+1} u_1(x_j)\cdot\varphi_j(x) \end{aligned} \tag{15}$$

where $\tilde{u}_1(t) = bc_0(t)$ and $\tilde{u}_{N_x+1}(t) = bc_1(t)$ are given functions and $u_{0,h}$ resp. $u_{1,h}$ are approximations of u_0 resp. u_1. Using that $a(\cdot,\cdot)$ and $c(\cdot,\cdot)$ are bilinear forms and that equation (14) is valid for each element of the base $\{\varphi_i\}_{i=2}^{N_x}$, we obtain:

$$\sum_{j=1}^{N_x+1} \frac{d^2}{dt^2}\tilde{u}_j(t)\cdot(\varphi_j,\varphi_i)_{L^2} + \sum_{j=1}^{N_x+1} \tilde{u}_j(t)\cdot a(\varphi_j,\varphi_i) + \sum_{j=1}^{N_x+1} \tilde{u}_j(t)\cdot c(\varphi_j,\varphi_i) = (f(t),\varphi_i) \tag{16}$$

$\forall i = 2\cdots N_x$. This equation can be written in a vector form. For this we define the vectors $\vec{u}$, $\vec{u_0}$, $\vec{u_1}$ and $\vec{f}$ with components

$$f_i(t) := (f(t),\varphi_i)_{L^2} \qquad u_j(t) := \tilde{u}_j(t)$$
$$u_{0,j} := u_0(x_j) \qquad u_{1,j} := u_1(x_j)$$

and matrices $\mathbf{M}$, $\mathbf{A}$ and $\mathbf{C}$ as

$$m_{ij} := (\varphi_j,\varphi_i)_{L^2} \qquad a_{ij} := a(\varphi_j,\varphi_i) \qquad c_{ij} := c(\varphi_j,\varphi_i)$$

Note that $\mathbf{M},\mathbf{A},\mathbf{C} \in \mathbb{R}^{N_x-1\times N_x+1}$, $\vec{u} \in \mathbb{R}^{N_x+1}$ and $\vec{f} \in \mathbb{R}^{N_x-1}$. So that (16) is equal to the Cauchy problem

$$\begin{aligned} \mathbf{M}\cdot\frac{d^2}{dt^2}\vec{u}(t) + (\mathbf{A}+\mathbf{C})\cdot\vec{u}(t) &= \vec{f}(t) \\ \vec{u}(t_0) &= \vec{u_0} \\ \frac{d}{dt}\vec{u}(t_0) &= \vec{u_1} \end{aligned} \tag{17}$$

3.1.3 The Newmark method

In this section the Newmark method is explained and used afterwards in our specific problem.
For the theoretical discussion of the Newmark, let us consider the ordinary differential equation of second order in time

$$\begin{cases} \ddot{u}(t) = g(t,u(t),\dot{u}(t)) \qquad t\in[a_t,b_t] \\ u(0) = u_0 \\ \dot{u}(0) = u_1 \end{cases} \tag{18}$$

Note that the development of the Newmark method is also valid for a system of ordinary differential equations. To simplify, we use the scalar ODE. The next paragraph gives

you a motivation why the Newmark method is chosen in this way. The interval $[a_t, b_t]$ is subdivided in N_t subintervals of same length $\triangle t = \frac{b_t - a_t}{N_t}$. This defines the partition $a_t = t_1 < t_2 < \ldots < t_{N_t+1} = b_t$ where the t_n are defined as $t_n = a_t + \Delta t \cdot (n-1)$. We use the Taylor development of second order of $u(t)$. Finally we note g_{t_n} instead of $g(t_n, u(t_n), \dot{u}(t_n))$.

$$u(t_{n+1}) = u(t_n) + \dot{u}(t_n)\Delta t + \frac{1}{2}\ddot{u}(t_n)\Delta t^2 + \zeta\ddot{u}(t_n)\Delta t^2 - \zeta\ddot{u}(t_n)\Delta t^2 + O(\Delta t^3) \quad (19)$$

Note that $\zeta\ddot{u}(t_n)\Delta t^2$ is once added and once subtracted. We replace $\zeta\ddot{u}(t_n)\Delta t^2$ by the zero order Taylor approximation of $\ddot{u}(t_n) = \zeta\ddot{u}(t_{n+1})\Delta t^2 + O(\Delta t^3)$. Using that u is a solution of the differential equation (18), we can replace the second order derivative of u by $g_{t_n} := g(t_n, u(t_n), \dot{u}(t_n)) = \ddot{u}(t_n)$. This leads us to

$$u(t_{n+1}) = u(t_n) + \Delta t v(t_n) + \Delta t^2[\zeta g_{t_{n+1}} + (\frac{1}{2} - \zeta)g_{t_n}] \quad (20)$$

In addition, we develop $\dot{u}(t)$

$$\dot{u}(t_{n+1}) = \dot{u}(t_n) + \ddot{u}(t_n)\Delta t + \theta\ddot{u}(t_n)\Delta t - \theta\ddot{u}(t_n)\Delta t + O(\Delta t^2) \quad (21)$$

Approximating $\theta\ddot{u}(t_n)\Delta t$ with the help of the zero order Taylor development, we find

$$\dot{u}(t_{n+1}) = \dot{u}(t_n) + \Delta t[\theta\ddot{u}(t_{n+1}) + (1-\theta)\ddot{u}(t_n)] \quad (22)$$

Now, we use also that u is the solution of the differential equation and replace the second order derivative of u by g_{t_n}.

$$\dot{u}(t_{n+1}) = \dot{u}(t_n) + \Delta t[\theta g_{t_{n+1}} + (1-\theta)g_{t_n}] \quad (23)$$

This brings us to the following scheme:
u^n being an approximation of $u(t_n)$ and v^n one for $\dot{u}(t_n)$

$$\begin{cases} u^{n+1} = u^n + \Delta t v^n + \Delta t^2[\zeta g_{t_{n+1}} + (\frac{1}{2} - \zeta)g_{t_n}] \\ v^{n+1} = v^n + \Delta t[\theta g_{t_{n+1}} + (1-\theta)g_{t_n}] \end{cases} \quad (24)$$

Now we are ready to apply the method to the system (17). In theory, we have $\vec{g}(t, u(t), \dot{u}(t)) = M^{-1}[\vec{f} - (A+C)\vec{u}]$, but practically it coasts a lot of time to calculate M^{-1}. So we multiply both parts by $\mathbf{M}$

$$\begin{cases} \mathbf{M}\vec{u}^{n+1} = \mathbf{M}\vec{u}^n + \Delta t\mathbf{M}\vec{v}^n + \Delta t^2[\zeta\vec{f}_{t_{n+1}} + (\frac{1}{2} - \zeta)\vec{f}_{t_n}] \\ \qquad\qquad -\Delta t^2(\mathbf{A}+\mathbf{C})[\zeta\vec{u}^{n+1} + (\frac{1}{2} - \zeta)\vec{u}^n] \\ \mathbf{M}\vec{v}^{n+1} = \mathbf{M}\vec{v}^n + \Delta t[\theta\vec{f}_{t_{n+1}} + (1-\theta)\vec{f}_{t_n}] - \Delta t(\mathbf{A}+\mathbf{C})[\theta\vec{u}^{n+1} + (1-\theta)\vec{u}^n] \end{cases} \quad (25)$$

where $\vec{f}_{t_n} = \vec{f}(t_n)$. Taking all $\vec{u}^{n+1}$ resp. $\vec{v}^{n+1}$ on the left side gives you

$$\begin{cases} [\mathbf{M} + \Delta t^2\zeta(\mathbf{A}+\mathbf{C})]\vec{u}^{n+1} = \mathbf{M}\vec{u}^n + \Delta t\mathbf{M}\vec{v}^n + \Delta t^2[\zeta\vec{f}_{t_{n+1}} + (\frac{1}{2} - \zeta)\vec{f}_{t_n}] \\ \qquad\qquad -\Delta t^2(\mathbf{A}+\mathbf{C})(\frac{1}{2} - \zeta)\vec{u}^n \\ \mathbf{M}\vec{v}^{n+1} = \mathbf{M}\vec{v}^n + \Delta t[\theta\vec{f}_{t_{n+1}} + (1-\theta)\vec{f}_{t_n}] - \Delta t(\mathbf{A}+\mathbf{C})[\theta\vec{u}^{n+1} + (1-\theta)\vec{u}^n] \end{cases} \quad (26)$$

Now, we would like to develop an algorithm for solving the system (26). We regroup this system that we can recognize how to define new variables.

$$\begin{cases} [\mathbf{M}+\Delta t^2\zeta(\mathbf{A}+\mathbf{C})]\vec{u}^{n+1} = [\mathbf{M}+\Delta t^2\zeta(\mathbf{A}+\mathbf{C})]\vec{u}^{n} - \frac{\Delta t^2}{2}(\mathbf{A}+\mathbf{C})\vec{u}^n + \Delta t\mathbf{M}\vec{v}^n \\ \qquad\qquad +\Delta t^2[\zeta\vec{f}_{t_{n+1}} + (\frac{1}{2}-\zeta)\vec{f}_{t_n}] \\ \mathbf{M}\vec{v}^{n+1} = \mathbf{M}\vec{v}^n + \Delta t[\theta\vec{f}_{t_{n+1}} + (1-\theta)\vec{f}_{t_n}] - \Delta t\theta(\mathbf{A}+\mathbf{C})\vec{u}^{n+1} \\ \qquad\qquad -\Delta t(1-\theta)(\mathbf{A}+\mathbf{C})\vec{u}^n \end{cases} \tag{27}$$

We define some new variables

$$\begin{aligned} \mathbf{A_u} &:= [\mathbf{M}+\Delta t^2\zeta(\mathbf{A}+\mathbf{C})] && (28)\\ \vec{b_u}^n &:= \mathbf{A_u}\cdot\vec{u}^n && (29)\\ \vec{b_v}^n &:= \mathbf{M}\cdot\vec{v}^n \\ \vec{AC_u}^n &:= (\mathbf{A}+\mathbf{C})\cdot\vec{u}^n \end{aligned}$$

Note that $\vec{b_u}^n \in \mathbb{R}^{N_x-1}$, $\vec{b_v}^n \in \mathbb{R}^{N_x-1}$ and $\vec{AC_u}^n \in \mathbb{R}^{N_x-1}$. We replace the new variables in the above-mentioned system

$$\begin{cases} \vec{b_u}^{n+1} = \vec{b_u}^n - \frac{\Delta t^2}{2}\vec{AC_u}^n + \Delta t\vec{b_v}^n + \Delta t^2[\zeta\vec{f}_{t_{n+1}} + (\frac{1}{2}-\zeta)\vec{f}_{t_n}] \\ \vec{b_v}^{n+1} = \vec{b_v}^n - \Delta t\theta\vec{AC_u}^{n+1} - \Delta t(1-\theta)\vec{AC_u}^n + \Delta t[\theta\vec{f}_{t_{n+1}} + (1-\theta)\vec{f}_{t_n}] \end{cases} \tag{30}$$

This allows us to write an algorithm with only four variables: $\vec{u}, \vec{b_u}, \vec{b_v}$ and $\vec{AC_u}$

initial step:

- $\vec{u} = \vec{u_0}$
- $\vec{b_u} = [\mathbf{M}+\Delta t^2\zeta(\mathbf{A}+\mathbf{C})]\cdot\vec{u}$
- $\vec{b_v} = \mathbf{M}\cdot\vec{u_1}$
- $\vec{AC_u} = (\mathbf{A}+\mathbf{C})\cdot\vec{u}$

step n:

- $\vec{b_u} = \vec{b_u} - \frac{\Delta t^2}{2}\vec{AC_u} + \Delta t\vec{b_v} + \Delta t^2[\zeta\vec{f}_{t_{n+1}} + (\frac{1}{2}-\zeta)\vec{f}_{t_n}]$
- Solve the system $\mathbf{A_{u,red}}\cdot\vec{u}_{red} = \vec{b_u} - u_1\cdot\vec{A_{u_1}} - u_{N_x+1}\cdot\vec{A_{u_{N_x+1}}}$
- $\vec{b_v} = \vec{b_v} - \Delta t(1-\theta)\vec{AC_u} + \Delta t[\theta\vec{f_{n+a}} + (1-\theta)\vec{f_n}]$
- $\vec{AC_u} = (A+C)\cdot\vec{u}$
- $\vec{b_v} = \vec{b_v} - \Delta t\theta\vec{AC_u}$
- Solve the system $\mathbf{M_{red}}\cdot\vec{v}_{red} = \vec{b_v} - v_1\cdot\vec{M_1} - v_{N_x+1}\cdot\vec{M_{N_x+1}}$

where $\vec{A_{u_i}}$ resp. $\vec{M_i}$ is the i-th column of the matrix $\mathbf{A_u}$ resp. $\mathbf{M}$ and $\mathbf{A_{u,red}} = (\vec{A_{u_2}}\dots\vec{A_{u_{N_x}}})$ resp. $\mathbf{M_{red}} = (\vec{M_2}\dots\vec{M_{N_x}})$ and $\vec{u}_{red} = (u_2,\dots,u_{N_x})^\top$ resp.

$\vec{v}_{red} = (v_2, \ldots, v_{N_x})^\top$. The obtained vector $\vec{u}$ at each step n is an approximation of the exact solution, i.e. the element u_i is an approximation of $u(a_i, t_{n+1})$.

The last step is supplementary. If you are interested in an approximation of the first derivative in time of the exact solution, then it could be calculated at each step, like you calculate u. But there are some additional problems. You like to solve the system $\mathbf{M} \cdot \vec{v}^{n+1} = \vec{b_v}$ with $\mathbf{M} \in \mathbb{R}^{N_x - 1 \times N_x + 1}$, $\vec{v}^{n+1} \in \mathbb{R}^{N_x+1}$ and $\vec{b_v} \in \mathbb{R}^{N_x - 1}$. The difference to u is that you don't have the information on the boundary, so that you have $N_x + 1$ unknowns but only $N_x - 1$ equations. The solution is in a 2 dimensional space. Using the well known functions bc_0 and bc_0, which describe the function u on the bound, v can be approximated by centered differences (on the boundary), a second order method. Having the values of v on the boundary, you apply the same scheme like for u, i.e. taking the first and last column of M multiplied by v_1 and v_{N_x+1} on the other side of the equality. The element v_i of the obtained vector $\vec{v}$ at the step n is an approximation of $\frac{\partial}{\partial t} u(a_i, t_{n+1})$.

3.2 Neumann boundary conditions

The goal of this section is the same as in the preceding, the numerical approximation with the Newmark method, but this time with Neumann boundary conditions. It will be a short description, detailed only on the passages where the procedure deviates from the above-mentioned one. By uncertainty, see the same passage in the preceding section.

3.2.1 The weak problem

In this case, the wave equation is also multiplied by a test function $v \in V = H^1(x_0, x_1)$ and integrated in space. The first difference appears in integrating by parts. Knowing the derivatives in space on the bound, it isn't demanded to v being zero on the bound. We write:

$$\begin{aligned} -a \int_{x_0}^{x_1} \frac{\partial^2 u}{\partial x^2} \cdot v \, dx &= a \int_{x_0}^{x_1} \frac{\partial u}{\partial x} \cdot \frac{\partial v}{\partial x} dx + a \frac{\partial u}{\partial x}(x_0, t) \cdot v(x_0) - a \frac{\partial u}{\partial x}(x_1, t) \cdot v(x_1) \\ &= a \int_{x_0}^{x_1} \frac{\partial u}{\partial x} \cdot \frac{\partial v}{\partial x} dx + a \cdot bc_0(t) \cdot v(x_0) - a \cdot bc_1(t) \cdot v(x_1) \end{aligned} \quad (31)$$

With the same definitions of the scalar product and the forms $a(\cdot, \cdot)$ and $c(\cdot, \cdot)$ as above, you arrive to the weak problem (WP):
find $u \in V$

$$\frac{d^2}{dt^2}(u, v)_{L^2} + a(u, v) + c(u, v) = (f, v)_{L^2} + a[bc_1(t)v(x_1) - bc_0(t)v(x_0)] \quad (32)$$

$$\begin{aligned} u(x, 0) &= u_0(x) \\ \frac{\partial}{\partial t} u(x, 0) &= u_1(x) \\ \frac{\partial}{\partial x} u(x_0, t) &= bc_0(t) \\ \frac{\partial}{\partial x} u(x_1, t) &= bc_1(t) \end{aligned}$$

for all $v \in V$.

3.2.2 Space-discretization with the Galerkin method

Like in the case of Dirichlet boundary conditions, we replace V by a finite dimensional subspace.

$$V_h = \{v_h \in C^0([x_0, x_1]) : v_h \mid_{[a_i, a_{i+1}]} \in \mathbb{P}_1, \forall i = 1 \cdots N_x\} \tag{33}$$

The approximation $u_h \in V_h$ of $u \in V$ can be written in the $N_x + 1$-dimensional base $\{\varphi_j\}$, defined in the preceding section. We write

$$u_h(x,t) = \sum_{j=1}^{N_x+1} \tilde{u}_j(t) \cdot \varphi_j(x) \tag{34}$$

and let $u_{0,h}$, $u_{1,h}$ be the projections from V in V_h of u_0, u_1

$$\begin{aligned} u_{0,h}(x) &= \sum_{j=1}^{N_x+1} u_0(x_j) \cdot \varphi_j(x) \\ u_{1,h}(x) &= \sum_{j=1}^{N_x+1} u_1(x_j) \cdot \varphi_j(x) \end{aligned}$$

$u_{0,h}(x)$ and $u_{1,h}(x)$ being approximations of $u_0(x)$ resp. $u_1(x)$ and as a consequence, the discrete formulation of the weak problem is given.

$$\begin{aligned} \sum_{j=1}^{N_x+1} \frac{d^2}{dt^2}\tilde{u}_j(t) \cdot (\varphi_j, \varphi_i)_{L^2} + \sum_{j=1}^{N_x+1} \tilde{u}_j(t) \cdot a(\varphi_j, \varphi_i) + \sum_{j=1}^{N_x+1} \tilde{u}_j(t) \cdot c(\varphi_j, \varphi_i) \\ = (f(t), \varphi_i) + a[bc_1(t)\varphi_i(x_1) - bc_0(t)\varphi(x_0)] \end{aligned} \tag{35}$$

for all $i = 1 \ldots N_x + 1$. So that this system of equation can be written in a matrix-vector form. For this, the same definitions of the matrices $\mathbf{M}$, $\mathbf{A}$, $\mathbf{C}$ and the vectors $\vec{u}$, $\vec{f}$ as in the preceding section are used. We obtain

$$\begin{aligned} \mathbf{M} \cdot \ddot{\vec{u}}(t) + (\mathbf{A} + \mathbf{C}) \cdot \vec{u}(t) &= \vec{f}(t) + \vec{l}(t) \\ \vec{u}(t_0) &= \vec{u_0} \\ \dot{\vec{u}}(t_0) &= \vec{u_1} \end{aligned} \tag{36}$$

where $\vec{l}(t) = (-a \cdot bc_0(t) \quad 0 \quad \cdots \quad 0 \quad a \cdot bc_1(t))^T$, $\mathbf{M}, \mathbf{A}, \mathbf{C} \in \mathbb{R}^{N_x+1 \times N_x+1}$ and $\vec{u}, \vec{f} \in \mathbb{R}^{N_x+1}$.

3.2.3 The Newmark method

Like in the case of Neumann boundary conditions, the Cauchy problem (36) is resolved by the Newmark method. So that we get

$$\begin{cases} [\mathbf{M} + \Delta t^2 \zeta(\mathbf{A} + \mathbf{C})]\vec{u}^{n+1} = [\mathbf{M} + \Delta t^2 \zeta(\mathbf{A} + \mathbf{C})]\vec{u}^n - \frac{\Delta t^2}{2}(\mathbf{A} + \mathbf{C})\vec{u}^n + \Delta t \mathbf{M}\vec{v}^n \\ \qquad + \Delta t^2[\zeta(\vec{f}_{t_{n+1}} + \vec{l}_{n+1}) + (\frac{1}{2} - \zeta)(\vec{f}_{t_n} + \vec{l}_n)] \\ \mathbf{M}\vec{v}^{n+1} = \mathbf{M}\vec{v}^n + \Delta t[\theta(\vec{f}_{t_{n+1}} + \vec{l}_{n+1}) + (1-\theta)(\vec{f}_{t_n} + \vec{l}_n)] - \Delta t\theta(\mathbf{A} + \mathbf{C})\vec{u}^{n+1} \\ \qquad - \Delta t(1-\theta)(\mathbf{A} + \mathbf{C})\vec{u}^n \end{cases} \tag{37}$$

where $\vec{l}_n = \vec{l}(t_n)$. We define exactly the same variables as in the case of Dirichlet boundary conditions, only that the dimensions will be different:

$$\begin{aligned} \mathbf{A_u} &:= [\mathbf{M} + \Delta t^2 \zeta(\mathbf{A} + \mathbf{C})] && (38) \\ \vec{b_u} &:= \mathbf{A_u} \cdot \vec{u} && (39) \\ \vec{b_v} &:= \mathbf{M} \cdot \vec{v} \\ A\vec{C}_u &:= (\mathbf{A} + \mathbf{C}) \cdot \vec{u} \end{aligned}$$

Note that $\mathbf{A_u} \in \mathbb{R}^{N_x+1 \times N_x+1}$ and $\vec{b_u}, \vec{b_v}, A\vec{C}_u \in \mathbb{R}^{N_x+1}$. Now, there is the same algorithm presented, but the dimensions of the matrix's and vectors differs.

initial step:

- $\vec{u} = \vec{u_0}$
- $\vec{b_u} = [\mathbf{M} + \Delta t^2 \zeta(\mathbf{A} + \mathbf{C})] \cdot \vec{u}$
- $\vec{b_v} = \mathbf{M} \cdot \vec{u_1}$
- $A\vec{C}_u = (\mathbf{A} + \mathbf{C}) \cdot \vec{u}$

step n:

- $\vec{b_u} = \vec{b_u} - \frac{\Delta t^2}{2} A\vec{C}_u + \Delta t \vec{b_v} + \Delta t^2 [\zeta(\vec{f}_{t_{n+1}} + \vec{l}_{n+1}) + (\frac{1}{2} - \zeta)(\vec{f}_{t_n} + \vec{l}_n)]$
- Solve the system $\mathbf{A_u} \cdot \vec{u} = \vec{b_u}$
- $\vec{b_v} = \vec{b_v} - \Delta t(1-\theta) A\vec{C}_u + \Delta t[\theta(\vec{f_{n+a}} + \vec{l}_{n+1}) + (1-\theta)(\vec{f_n} + \vec{l}_n)]$
- $A\vec{C}_u = (A + C) \cdot \vec{u}$
- $\vec{b_v} = \vec{b_v} - \Delta t \theta A\vec{C}_u$
- Solve the system $\mathbf{M} \cdot \vec{v} = \vec{b_v}$

For additional information on the solution the first derivative in time can be approximated. You solve at each step of time the system $\mathbf{M} \cdot \vec{v}^{n+1} = \vec{b_v}$ with $\vec{b_v}$ the last calculated in the algorithm. The dimensions are $\mathbf{M} \in \mathbb{R}^{N_x+1 \times N_x+1}$ and $\vec{b_v}, \vec{u} \in \mathbb{R}^{N_x+1}$. Thanks to the right dimensions, there is no problem to solve it.

3.3 The Matlab Code

The Matlab Code for the Newmark method calculates the approximation of the wave equation with the Newmark method. All the computations are based on the previous theory. You have as inputs : x_0, x_1, N_x, t_0, t_1, N_t, a, c, θ, ζ, u_0, u_1, f, $bccode$, bc_0 and bc_1. $bccode$ (boundary condition code) is a two dimensional boolean line-vector, where 0 means Dirichlet and 1 Neumann boundary conditions. The first element corresponds to x_0 and the second to x_1. For example $[1 \quad 1]$ means Neumann boundary conditions in x_0 and x_1. The outputs are an error message or a plot for u and $v = \frac{\partial u}{\partial t}$ (the first derivative in space). The plot is the surface defined by $u(x,t)$ resp. $v(x,t)$.

3.3.1 Approximation of $(f(t), \varphi_i)_{L^2}$

The Simpson formula is used to approximate $f_i(t) = (f(t), \varphi_i)_{L^2} = \int_{x_0}^{x_1} f(t)\varphi_i dx$. Let $[a, b] \subset \mathbb{R}$ be an interval and $h = \frac{b-a}{2}$. You like to approximate $\int_a^b g(x)dx$. The Simpson formula is the following:

$$I_3(g) = \frac{h}{3}[g(a) + 4g(\frac{a+b}{2}) + g(b)] \tag{40}$$

The Simpson formula is exact in degree 5, because the points a, b and $\frac{a+b}{2}$ are the zeros of the corresponding orthogonal polynomial of degree 3 in $[a, b]$.

In our case, we approximate $f_i(t)$ in the two intervals $[a_{i-1}, a_i]$ and $[a_i, a_{i+1}]$, in each interval by the Simpson formula. In the rest of the interval $[x_0, x_1]$ the function φ is identic to zero, and also $f(t) \cdot \varphi$. Let $x_{i\pm\frac{1}{2}}$ be the points $x_i \pm \frac{h}{2}$ and $h_2 = \frac{h}{2}$. Note that $\varphi(a_{i-1}) = 0 = \varphi(a_{i+1})$ and

$$\begin{aligned} f_i(t) &\approx \frac{h_2}{3}[4f(x_{i-\frac{1}{2}}, t)\frac{1}{2} + 2 \cdot f(x_i, t) + 4f(x_{i-\frac{1}{2}}, t)\frac{1}{2}] \\ &\approx \frac{h}{3}[f(x_{i-\frac{1}{2}}) + f(x_i) + f(x_{i-\frac{1}{2}})] \end{aligned} \tag{41}$$

For f_1 and f_{N_x+1}, we have

$$\begin{aligned} f_1(t) &\approx \frac{h}{3}[\frac{f(a_1)}{2} + f(a_2)] \\ f_{N_x+1}(t) &\approx \frac{h}{3}[\frac{f(a_{N_x+1})}{2} + f(a_{N_x})] \end{aligned} \tag{42}$$

This is used in the Matlab-file $int\_f.m$. The inputs are f,N_x,h,x_0,t and it returns $\vec{f}(t)$.

3.4 Theoretical study of the Newmark method

The order of the Newmark method depending on the parameters will be determined in this section. We consider the differential equation (18) and let A be the following application:

$$\begin{pmatrix} u^{n+1} \\ v^{n+1} \end{pmatrix} = A \begin{pmatrix} u^n \\ v^n \end{pmatrix} = \begin{pmatrix} u^n + \Delta t v^n + \Delta t^2(\zeta g_{t_{n+1}} + (\frac{1}{2} - \zeta)g_{t_n}) \\ v^n + \Delta t(\theta g_{t_{n+1}} + (1-\theta)g_{t_n}) \end{pmatrix} \tag{43}$$

Let u be the exact solution of (18) and

$$\vec{X}(t) = \begin{pmatrix} u(t) \\ u'(t) \end{pmatrix} \tag{44}$$

where $u' = \frac{\partial u}{\partial t}$.Then we define $\varepsilon_n(\Delta t) = \vec{X}(t_{n+1}) - A(\vec{X}(t_n))$. The application A will be of order p, if $\varepsilon_n(\Delta t) = O(\Delta t^{p+1})$. We already have

$$A(\vec{X}(t_n)) = \begin{pmatrix} u(t_n) + \Delta t u'(t_n) + \Delta t^2(\zeta g_{t_{n+1}} + (\frac{1}{2} - \zeta)g_{t_n}) \\ u'(t_n) + \Delta t(\theta g_{t_{n+1}} + (1-\theta)g_{t_n}) \end{pmatrix} \tag{45}$$

and we use the Taylor expansion of $\vec{X}(t_{n+1})$.

$$\begin{aligned} u(t_{n+1}) &= u(t_n) + \Delta t u'(t_n) + \frac{\Delta t^2}{2} u''(t_n) + \zeta \Delta t^2 u''(t_n) - \zeta \Delta t^2 u''(t_n) \\ &\quad + \frac{\Delta t^3}{6} u'''(t_n) + O(\Delta t^4) \end{aligned} \tag{46}$$

We have also the first order development of $u''(t_{n+1})$:

$$\begin{aligned} & u''(t_{n+1}) = u''(t_n) + \Delta t u'''(t_n) + O(\Delta t^2) \\ \Rightarrow \quad & u''(t_n) = u''(t_{n+1}) - \Delta t u'''(t_n) + O(\Delta t^2) \end{aligned} \tag{47}$$

Now we replace

$$\zeta \Delta t^2 u''(t_n) = \zeta \Delta t^2 u''(t_{n+1}) - \zeta \Delta t^3 u'''(t_n) + O(\Delta t^4) \tag{48}$$

in the equation (46). This leads us to

$$\begin{aligned} u(t_{n+1}) &= u(t_n) + \Delta t u'(t_n) + \Delta t^2 (\zeta u''(t_{n+1}) + (\frac{1}{2} - \zeta) u''(t_n)) \\ &\quad + \Delta t^3 (\frac{1}{6} - \zeta) u'''(t_n) + O(\Delta t^4) \end{aligned} \tag{49}$$

In the same way we can also find

$$\begin{aligned} u'(t_{n+1}) &= u'(t_n) + \Delta t (\theta u''(t_{n+1}) + (1-\theta) u''(t_n)) \\ &\quad + \Delta t^2 (\frac{1}{2} - \theta) + O(\Delta t^3) \end{aligned} \tag{50}$$

and as consequence we have

$$\vec{X}(t_{n+1}) - A(\vec{X}(t_{n+1})) = \begin{pmatrix} \Delta t^3 (\frac{1}{6} - \zeta) u'''(t_n) + O(\Delta t^4) \\ \Delta t^2 (\frac{1}{2} - \theta) u'''(t_n) + O(\Delta t^3) \end{pmatrix} \tag{51}$$

We see that in the case of $\theta = \frac{1}{2}$, the Newmark method is a second order one, and if not, the method is only a first order one.

4 Leap-Frog method

The mathematical problem in this section is exactly the same as in the precedent, we would like to approximate the solution of the wave equation. This time, we don't use the Galerkin method, but the finite differences. We don't differ the two cases of boundary conditions, a later treatment for this two instances will follow. The method is first explained for both cases, but we neglect all difficulties connected to the boundary conditions. Like in previous chapter, let be $t^n = a_t + (n-1)\cdot\Delta t$ and $a_i = x_0 + (i-1)\cdot h$, where $\Delta t = \frac{b_t - a_t}{N_t}$ and $h = \frac{x_1 - x_0}{N_x}$ ($i = 1 \dots N_x + 1$, $n = 1 \dots N_t + 1$). For all functions $g : Q \to \mathbb{R}$ we note g_j^n instead of $g(a_j, t_n)$, where we have the same discretization of space and time as in the previous chapter. The finite differences are given

$$\frac{\partial^2 u}{\partial t^2} \approx \frac{u_j^{n+1} - 2u_j^n + u_j^{n-1}}{\Delta t^2} \tag{52}$$

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{h^2} \tag{53}$$

We replace the two second derivatives in (1) with the above mentioned approximations. This leads to the following scheme

$$\frac{u_j^{n+1} - 2u_j^n + u_j^{n-1}}{\Delta t^2} - a \cdot \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{h^2} + c \cdot u_j^n = f_j^n \tag{54}$$

Advancing in time, we separate the terms u in time t_{n+1} from the other ones in time t_n and t_{n-1}. Let be $\lambda = \frac{\Delta t}{h}$.

$$u_j^{n+1} = -u_j^{n-1} + (2 - 2a\lambda^2 - c\Delta t^2) \cdot u_j^n + a\lambda^2 \cdot (u_{j+1}^n + u_{j-1}^n) + \Delta t^2 \cdot f_j^n \tag{55}$$

Besides the complications on the boundary, the initial step put some additional problems. For calculating u_j^2 we need $u_j^0 = u(a_j, a_t - \Delta t)$, so it have to be approximated. Using the initial condition (3), we apply the centered differences

$$u_1(a_j) \approx \frac{u_j^2 - u_j^0}{2\Delta t} \tag{56}$$

where $u_j^0 = u(a_j, a_t - \Delta t)$. Note that u_1 is the initial condition for the derivative in time of u. So we have $u_j^0 = u_j^2 - 2\Delta t u_1(a_j)$ and as consequence

$$\begin{aligned} u_j^2 &= -u_j^2 + 2\Delta t \cdot u_1(a_j) + (2 - 2a\lambda^2 - c\Delta t^2) \cdot u_0(a_j) \\ &\quad + a\lambda^2 \cdot (u_0(a_{j+1}) + u_0(a_{a_{j-1}})) + \Delta t^2 \cdot f_j^1 \end{aligned} \tag{57}$$

$$\Rightarrow$$

$$\begin{aligned} u_j^2 &= \Delta t \cdot u_1(a_j) + (1 - a\lambda^2 - \frac{c\Delta t^2}{2}) \cdot u_0(a_j) \\ &\quad + \frac{a\lambda^2}{2} \cdot (u_0(a_{j+1}) + u_0(a_{j-1})) + \frac{\Delta t^2}{2} \cdot f_j^1 \end{aligned} \tag{58}$$

4.1 Dirichlet boundary conditions

In the case of Dirichlet boundary conditions, there aren't any complication, because we don't have to calculate u_1^n and $u_{N_x+1}^n$. So we replace simply u_1^n by $bc_0(t_n)$ and $u_{N_x+1}^n$ by $bc_1(t_n)$ in the equation (55) for all $j = 2 \dots N_x$.

4.2 Neumann boundary conditions

In case of Neumann boundary conditions, we need to calculate u_1^n and $u_{N_x+1}^n$. For this, we need $u_0^n = u(x_0 - h, t_n)$ and $u_{N_x+2}^n = u(x_1 + h, t_n)$. Using the centered difference approximation of the boundary conditions

$$bc_0(t_n) = \frac{\partial}{\partial x} u(t_n, x_0)) \approx \frac{u_2^n - u_0^n}{2h} \tag{59}$$

$$bc_1(t_n) = \frac{\partial}{\partial x} u(t_n, x_1)) \approx \frac{u_{N_x+2}^n - u_{N_x}^n}{2h} \tag{60}$$

leads us to

$$u_0^n = u_2^n - 2h \cdot bc_0(t_n) \tag{61}$$

$$u_{N_x+2}^n = u_{N_x}^n + 2h \cdot bc_0(t_n) \tag{62}$$

The following equations results by replacing this in (55)

$$\begin{aligned} u_1^{n+1} &= -u_1^{n-1} + (2 - 2a\lambda^2 - c\Delta t^2) \cdot u_1^n + 2a\lambda^2 \cdot (u_2^n - h \cdot bc_0(t_n)) + \Delta t^2 \cdot f_1^n \\ u_{N_x+1}^{n+1} &= -u_{N_x+1}^{n-1} + (2 - 2a\lambda^2 - c\Delta t^2) \cdot u_{N_x+1}^n + 2a\lambda^2 \cdot (u_2^n + h \cdot bc_0(t_n)) + \Delta t^2 \cdot f_{N_x+1}^n \end{aligned}$$

and for the initial step we obtain

$$\begin{aligned} u_1^2 &= \Delta t \cdot u_1(x_0) + (1 - a\lambda^2 - \frac{c\Delta t^2}{2}) \cdot u_0(x_0) \\ &\quad + a\lambda^2 \cdot (u_0(a_2) - h \cdot bc_0(a_t)) + \frac{\Delta t^2}{2} \cdot f_1^1 \end{aligned} \tag{63}$$

$$\begin{aligned} u_{N_x+1}^2 &= \Delta t \cdot u_1(x_1) + (1 - 1a\lambda^2 - \frac{c\Delta t^2}{2}) \cdot u_0(x_1) \\ &\quad + a\lambda^2 \cdot (u_0(a_{N_x}) + h \cdot bc_1(a_t)) + \frac{\Delta t^2}{2} \cdot f_{N_x+1}^1 \end{aligned} \tag{64}$$

5 Fluid-Structure Interaction

The fluid-structure interaction is studied in this section. A simple radial symmetric 1D model (with a one dimensional wall) is used. For any details see [1], chapter 4. It isn't part of this work to develop the model. Briefly, for the fluid part we use the Stokes equations and for the vessel the wave equation.

5.1 The fluid

Like mentioned above, the Stokes equations are used in a moving domain (the model will be simplified afterwards). The initial domain $\Omega_0 \subset \mathbb{R}^2$ is supposed to be $[0, L] \times [0, R_0]$. Let be $R : [0, T] \times [0, L] \to \mathbb{R}$ the displacement function of the vessel.

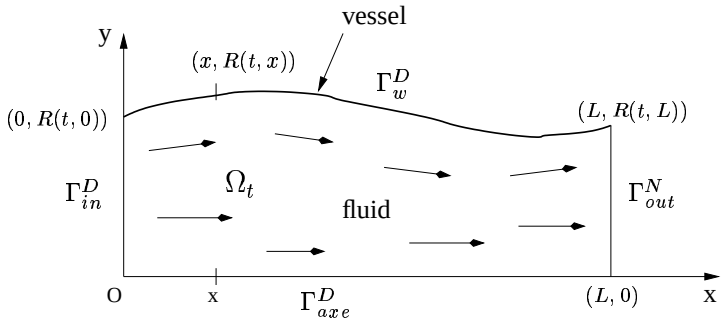

Figure 1: The physical problem

The physical problem for the fluid are the following Stokes equations. We use a Matlab code, which can only treat the cases of Dirichlet (homogeneous or non-homogeneous) or homogeneous Neuman boundary conditions for $\vec{u}$. We put the following boundary conditions.

$$\begin{aligned} u &= 0 \qquad on\ \Gamma_{axe}^D \\ v &= 0 \qquad on\ \Gamma_{axe}^D \\ u &= 8 \cdot \alpha \cdot y(.5 - y)(1 - \cos\frac{2\pi t}{T}) \qquad on\ \Gamma_{in}^D \\ v &= 0 \qquad on\ \Gamma_{in}^D \\ \frac{\partial \vec{u}}{\partial \vec{n}} - p \cdot \vec{n} &= 0 \qquad on\ \Gamma_{out}^N \end{aligned} \tag{65}$$

So the problem becomes to find $\vec{u} = (u, v)$ and p that

$$\begin{aligned} \frac{\partial \vec{u}}{\partial t} - \nu \triangle \vec{u} + \nabla p &= 0 \qquad in\ \Omega_t \\ div(\vec{u}) &= 0 \qquad in\ \Omega_t \\ \vec{u}(x, y, t_0) &= \vec{u_0}(x, y) \qquad in\ \Omega_0 \\ p(x, y, t_0) &= p_0(x, y) \qquad in\ \Omega_0 \\ u &= 0 \qquad on\ \Gamma_{axe}^D \end{aligned} \tag{66}$$

$$
\begin{aligned}
v &= 0 \qquad on\ \Gamma^D_{axe} \\
u &= 4 \cdot \alpha \cdot y(.5-y)(1-\cos\frac{2\pi t}{T}) \qquad on\ \Gamma^D_{in} \\
v &= 0 \qquad on\ \Gamma^D_{in} \\
\frac{\partial \vec{u}}{\partial \vec{n}} - p \cdot \vec{n} &= 0 \qquad on\ \Gamma^N_{out} \\
\vec{u} &= (0, \frac{\partial R}{\partial t}) \qquad on\ \Gamma^D_{w}
\end{aligned}
$$

We would like to treat this problem without moving the domain. To this aim, we fix the domain and we use a transpiration method to approximate the velocity at the upper boundary. Caused by this reason, the domain Ω_t has to be reduced on a fix domain Ω_{red}. Let be $\Omega_{red} = [0, L] \times [0, R_0]$ and $R(t, x) = \eta(t, x) + r$. $\eta : [0.L] \times [0.T] \to \mathbb{R}$ will be the displacement of the vessel from the fixed domain.

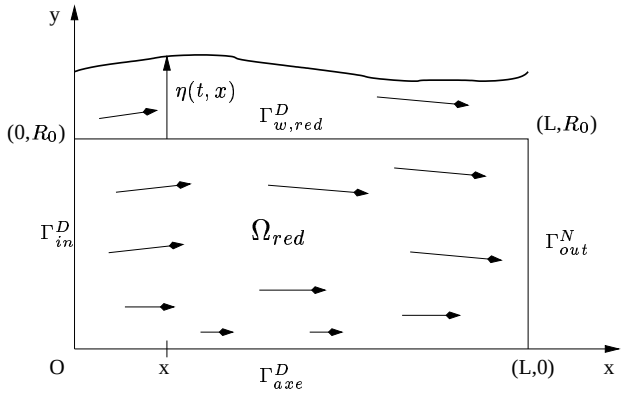

Figure 2: The simplified model

The solutions are only calculated on the reduced domain. This leads us to the simplified problem

$$
\begin{aligned}
\frac{\partial \vec{u}}{\partial t} - \nu \triangle \vec{u} + \nabla p = &\ 0 \qquad in\ \Omega_{red} \qquad (67) \\
div(\vec{u}) &= 0 \qquad in\ \Omega_{red} \\
\vec{u}(x, y, t_0) &= \vec{u_0}(x, y) \qquad in\ \Omega_{red} \\
p(x, y, t_0) &= p_0(x, y) \qquad in\ \Omega_{red} \\
u &= 0 \qquad on\ \Gamma_{axe} \\
v &= 0 \qquad on\ \Gamma_{axe} \\
u &= 8 \cdot \alpha \cdot y(.5-y)(1-\cos\frac{2\pi t}{T}) \qquad on\ \Gamma_{in} \\
v &= 0 \qquad on\ \Gamma_{in} \\
\frac{\partial \vec{u}}{\partial \vec{n}} &= 0 \qquad on\ \Gamma_{out} \\
\vec{u}(x, y = r, t) &= (0, u_w(x, t)) \qquad on\ \Gamma_{w,red}
\end{aligned}
$$

Note that we assume that η is small and as consequence the x component of $\vec{u}$ on the boundary $\Gamma^D_{w,red}$ is supposed to be zero. Now u_w has still to be estimated, supposing that we know $\dot{\eta}$. This is considered to be a problem of the fluid-structure interaction and is treated in the section interaction.

For the numerical computation of the fluid part, the Matlab code *MLife* is used. This code is developed at the Politecnico di Milano, namely from Prof. F. Saleri of the MOX.

5.2 The vessel

For the simulation of the vessel, the wave equation with homogeneous boundary conditions is used. The equation is the following one

$$\frac{\partial^2 \eta}{\partial t^2} - a \cdot \frac{\partial^2 \eta}{\partial x^2} + b \cdot \eta = H_t \tag{68}$$

where $H_t = H(x,t) = \frac{p(x,r+\eta,t)}{\rho_w \cdot h_0}$ depends of x and t.
Determination of the parameters:

$$\begin{aligned} a &= \frac{\sigma^z}{\rho_w h_0} \\ b &= \frac{E}{\rho_w (1-\xi^2) R_0^2} \end{aligned} \tag{69}$$

where

ρ_w: vessel density
h_0: thickness of the vessel
σ_z: tangential component of the longitudinal stress
E: Young modulus
ξ: Poisson coefficient

For the numerical computation of the solution of this equation, the Newmark method is used. The advantage is that we can obtain also $\dot{\eta}$. This will be used in the first order approximation of u_w (see the section interaction).

5.3 Interaction

Remember that we note $\vec{u} = (u, v)$. First, we would like to estimate the velocity on the upper boundary of the fluid domain, $v(x, R_0, t)$, noted $u_w(x, t)$. We suppose to know $\dot{\eta}$ at each step of time t^n. The interface condition is the following

$$\begin{aligned} v(x, R_0 + \eta(x), t) &= \frac{\partial \eta}{\partial t}(x, t) = \frac{\partial R}{\partial t}(x, t) \\ u(x, R_0 + \eta(x), t) &= 0 \end{aligned} \tag{70}$$

Then, the first order Taylor approximation in the point (x, R_0) in the y direction is used

$$v(x, R_0 + \eta(x), t) = v(x, R_0, t) + \frac{\partial v}{\partial y}(x, R_0, t) \cdot \eta(x, t) + O(\eta(x, t)^2) \tag{71}$$

Note that $u_w(x, t) = v(x, R_0, t)$ and as consequence, two approximations are presented

0^{th} order approximation:

$$u_w(x, t^{n+1}) = \frac{\partial \eta}{\partial t}(x, t^{n+1}) \tag{72}$$

1^{st} order approximation:

$$u_w(x, t^{n+1}) = \frac{\partial \eta}{\partial t}(x, t^{n+1}) - \frac{\partial v}{\partial y}(x, R_0, t^{n+1}) \cdot \eta(x, t^{n+1}) \tag{73}$$

5.4 The algorithm

The main problem of the fluid-structure interaction is that we have two different physical laws (one for the fluid part and one for the structure part), which are coupled. For both problems we have a numerical method to do the calculus with given initial and boundary conditions. This algorithm is a simpler version fo thisone described in [2].

The algorithm is presented as N_t steps progressing in time. In the step $n \in \{1, \ldots, N_t + 1\}$ we are at the time t^n and we would like to calculate the solution at time t^{n+1}. Each time step, we have a sub-iteration indexed by k. At every step k (at time t^n) the algorithm is separated in two periods. In the first period, the Stokes problem is solved in one step of time t^n to t^{n+1}. The obtained result in the previous time step is used as initial condition for the Stokes problem (not as initial condition for the whole problem), concretely $\vec{u}^{n,k_{tot}}$ and $p^{n,k_{tot}}$. The boundary conditions in Γ^D_{axe},Γ^D_{in} and Γ^N_{out} are independent of the vessel, so it can be easy calculated. The last boundary condition, this on $\Gamma_{w,red}$ is calculated by transpiration starting from $\eta^{n+1,k}$ and $\dot{\eta}^{n+1,k}$ with result $(0, u_w^{n+1,k})$. The solution of the Stokes problem is $\vec{u}^{n+1,k+1}$ and $p^{n+1,k+1}$. In the second period, using $p^{n+1,k+1}$, the stress H is calculated at time t^{n+1} (note that H is also needed in time t^n, but this is already calculated in the previous time step). Then the vessel problem is solved with initial conditions $\eta^{n,k_{tot}}$ and $\dot{\eta}^{n,k_{tot}}$. This gives us $\eta^{n,k+1}$ and $\dot{\eta}^{n,k+1}$. Then we relax this solution with the solutions obtained in k-step before. The sub-iteration is calculated till a convergence tolerance is satisfied for η or we have more than $Nmax$ steps in k. If this conditions aren't satisfied, we put $k = k+1$ and start at the begin of the sub-iteration. If not, we save the solutions,

$$\vec{u}^{n+1,k_{tot}} = \vec{u}^{n+1,k+1}$$
$$p^{n+1,k_{tot}} = p^{n+1,k+1}$$
$$\eta^{n+1,k_{tot}} = \eta^{n+1,k+1}$$
$$\dot{\eta}^{n+1,k_{tot}} = \dot{\eta}^{n+1,k+1}$$

and we put $n = n + 1$, $k = 1$. Then the next time step is computed and new sub-iterations started and so on. This is done till we have $n = N_t$. Note that at each sub-step k, we approximate $(0, u_w^{n+1})$ by $(0, u_w^{n+1,k})$, because at step $k = 1$, $(0, u_w^{n+1})$ is not given. This is the reason why we have to do this sub-iteration. At step $k = 1$, $(0, u_w^{n+1})$ is approximated by $(0, u_w^{n,k_{tot}})$ and the solution of the vessel problem gives us a solution $\tilde{\eta}^{n+1,2}$. We do a ω-relaxation with $\eta^{n+1,1}$, concrete $\eta^{n+1,2} = \omega \cdot \tilde{\eta}^{n+1,2} + (1 - \omega) \cdot \eta^{n+1,1}$. This gives us an approximation of $(0, u_w^{n+1})$ using transpiration, noted $(0, u_w^{n+1,2})$. Then solving the Stokes and the vessel problem starting from $(0, u_w^{n+1,2})$ as boundary condition gives us a better approximation (after relaxation) of $(0, u_w^{n+1})$, noted as $(0, u_w^{n+1,3})$ and so on, till we have convergence in η. At the end of this k-iterations $(0, u_w^{n+1,k_{tot}})$ should be a good approximation of $(0, u_w^{n+1})$.
For the graphical representation of the algorithm, see fig 3, we use the following notation:

$$(input_1, \ldots, input_r) X (output_1, \ldots, output_l) \tag{74}$$

where X can be T for Transpiration, S for Stokes, H for the calculation of the stress H or V for vessel. It means that we solve the problem X with the inputs $input_1, \ldots, input_r$ and as result we have $output_1, \ldots, output_l$. In fig 3, $error_\eta$ is defined as follows

$$error_\eta = \frac{\max_i(\mid \tilde{\eta}(i)^{n+1,k+1} - \eta(i)^{n+1,k} \mid)}{\mid \tilde{\eta}(i)^{n+1,k+1} \mid} \tag{75}$$

Note that in the programme, η is a vector. Every element correspond to to a point on the upper boundary (see Results).

Figure 3: The algorithm

5.5 Results

In this section some results of two different versions of this algorithm are given. For both versions the parameters are the followings

parameter	value
L	6cm
N_x	48
ν	0.035
h_0	0.1
ρ_w	1.1
σ_z	25000
ξ	0.5
R_0	0.5cm
E	$0.75E \cdot 10^6$
α	100
θ	0.5
ζ	0.25

$N_x + 1$ is number of vertices on the boundary Γ_w^D. This means firstly that the upper boundary is divided in N_x intervals for the simulation of the vessel with the Newmark method. This define $N_x + 1$ points on Γ_w^D. Secondly for the simulation of the fluid, a triangulation is used, where this $N_x + 1$ points on Γ_w^D correspond to the vertices on Γ_w^D of the triangulation. In both cases Δt is equal to 0.1 milli-seconds. N_{max} is chosen as 150 and we use the zero order transpiration for u_w.

5.5.1 First version

In this version, the vessel equation is solved with Neuman boundary conditions.

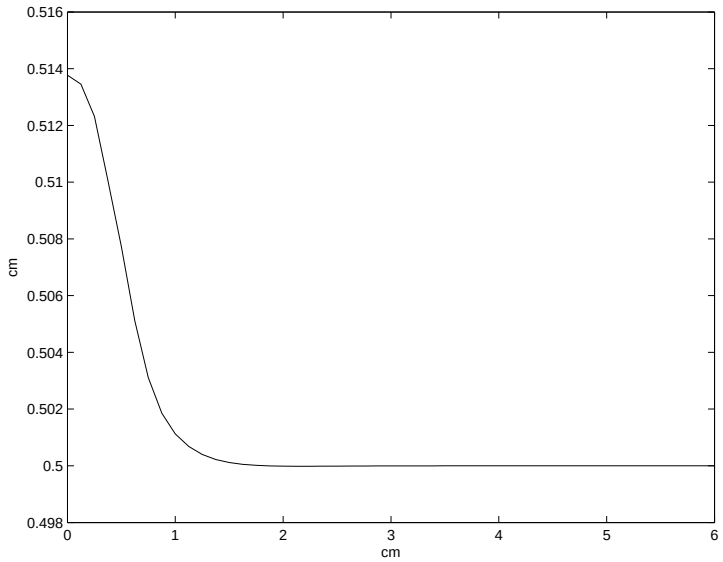

Figure 4: The vessel at time 0.002 seconds

Figures 4-7 show that this version is not a good one. Note the different scales on the y-axis. Following the wave front, you will notice that the vessel begin to swing before the wave front even arrives in this points. This is a fact that could not be observed in

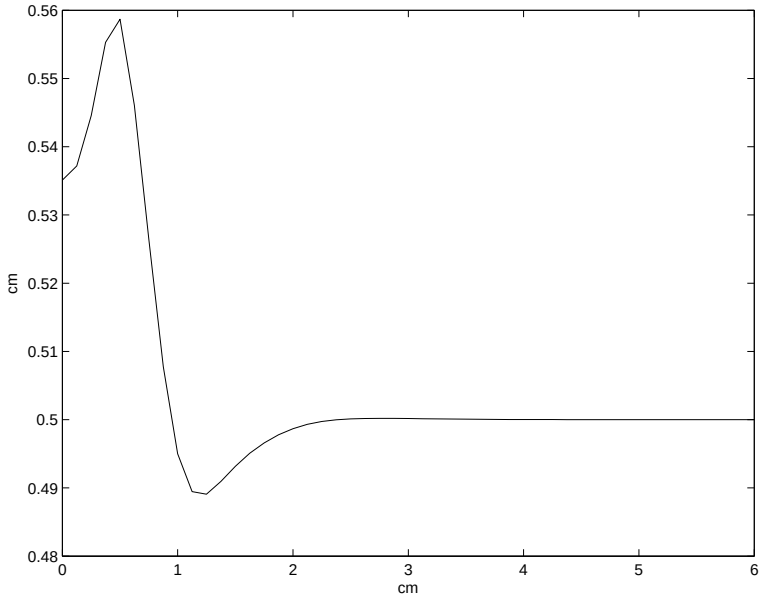

Figure 5: The vessel at time 0.004 seconds

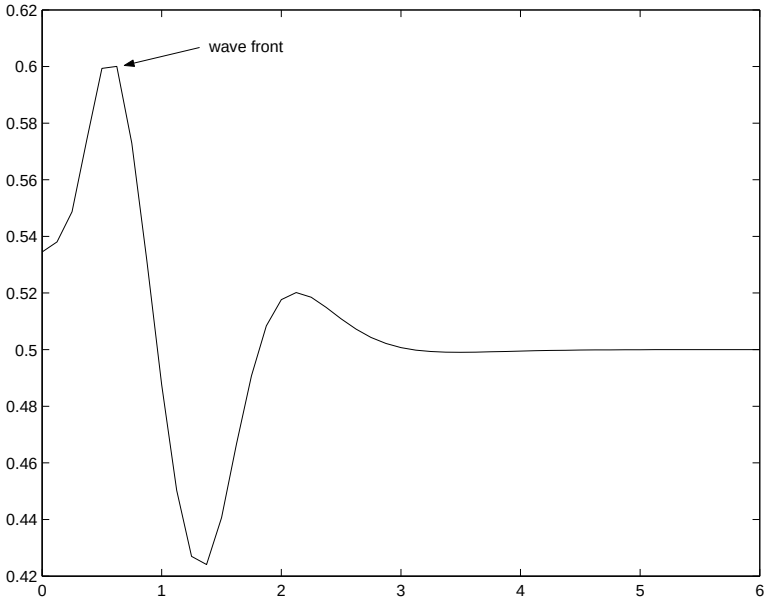

Figure 6: The vessel at time 0.006 seconds

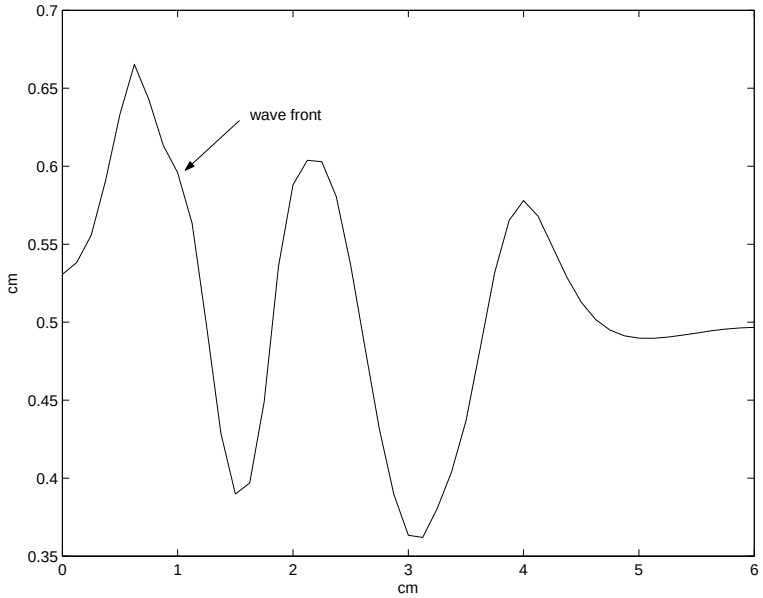

Figure 7: The vessel at time 0.01 seconds

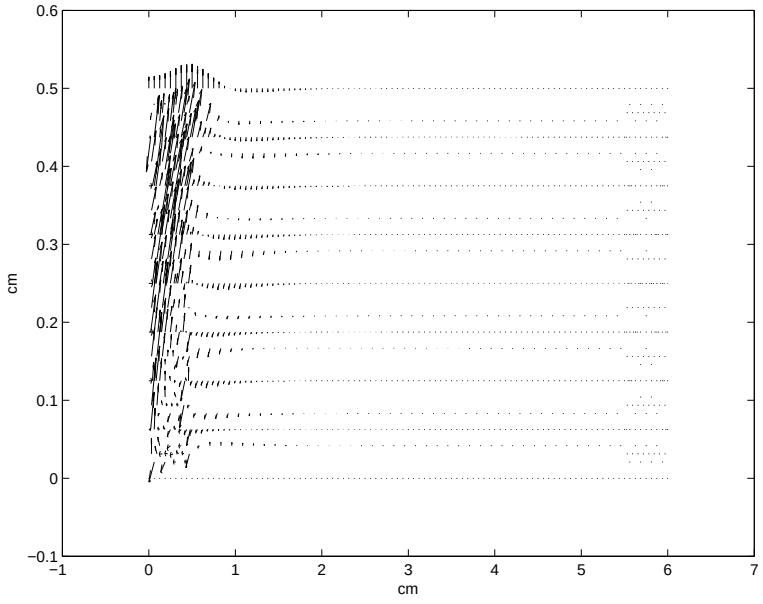

Figure 8: The velocity field at time 0.003 seconds

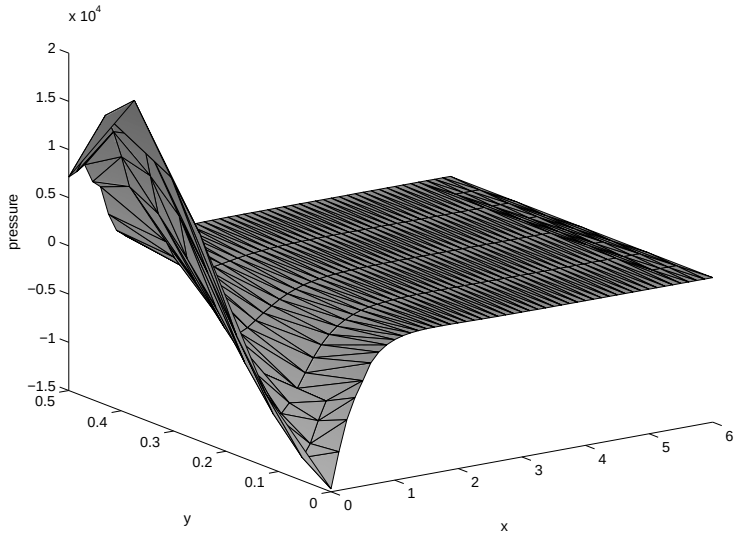

Figure 9: The pressure at time 0.002 seconds

reality. Why the pressure decreases that much in the point $(0, 0)$ in fig.(9)? Why this model does not work? Here are some possible explanations and weak points of this model

- Using Neuman boundary conditions for the vessel, we have a part of the boundary Ω_t, the segment between the points $(0, R_0)$ and $(0, R(t))$, where we have not absolutly an inflow. Maybe you have there some effects, which disturb the whole simulation, like for example that the fluid pour out of the domain in this segment, and this let decrease the pressure in this region.

- Since the vessel swings that much, it is possible that the parameter ρ_w is too small. ρ_w is the density of the vessel and influence also the stiffness of the vessel.

- Do we have really at each n-iteration convergence after the N_{max} steps.

- The zero order transpiration could be bad. Maybe there is an effect that, if η is already large, then the bad approximation support η increasing, this means that the wave-amplitude increases and the pressure would also be high in this region

- It's maybe because we don't describe the incoming wave as a pressure wave, but as a Dirichlet boundary condition on the velocity field.

5.5.2 Second version

Since the first version does not work, an other version is tried. To better control the flux on the inflow, Γ_{in}^D, we try the model with Dirichlet boundary conditions for the vessel. Because of lack of time, only the results of step 1 to 40 could be calculated. This means

a final time at the step 40 of 0.004 seconds, which corresponds to 4 milli-seconds. You can see that at time 0.004, fig.(11), the two solutions are similar and that also in the second version, the vessel increases in the domain between $x = 1$ and $x = 1,5$ like in the first version. It can be suspected that also in this model, the vessel begin to swing. Unfortunately, we had no time to verify it. The computation had no time to finish, yet.

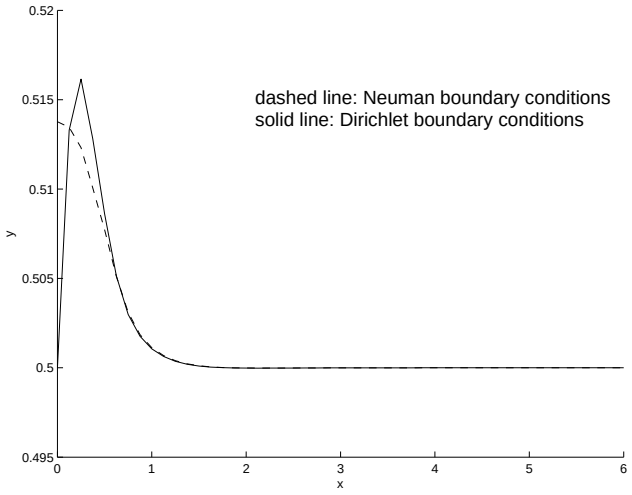

Figure 10: The two vessels at time 0.002 seconds

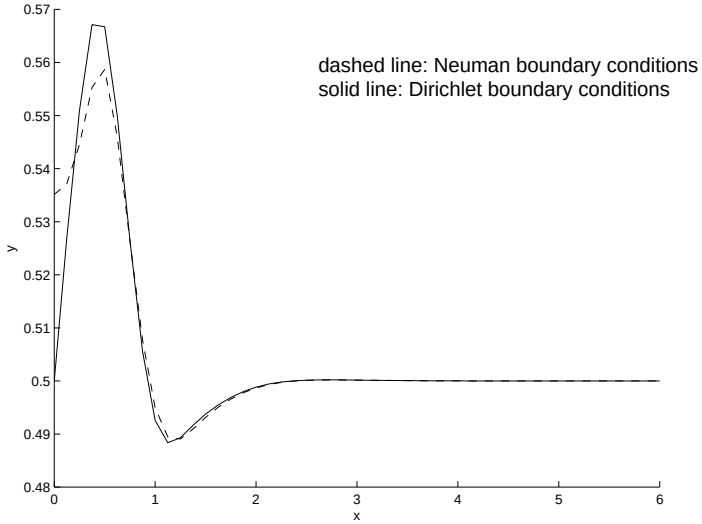

Figure 11: The two vessels at time 0.004 seconds

6 Conclusions

In this project, we analyzed two different methods to approximate the solution of the wave equation, then we developed a simple model of the blood flow in arteries. Unfortunately, the results were not that good. From this point of view, I can not supply meaningful results, but just the modelling of a physical problem and also being confronted with a lot of questions in programming, was a very good experience. From this point of view, I learnt some basics of hyperbolic partial differential equations and solution methods as well as an introduction to fluid-structure interaction, including the difficulties behind that. It was an interesting work, which was a mix between theoretical study of some methods and programming in Matlab. Thanks to this work, I learnt how to argue and solve problems in numerical analysis as well as programming in *Matlab* and write articles with LaTeX. I even could visit Milano.

7 Matlab Codes

7.1 The Newmark Code

```
function res=newmarkbybeni(x0,x1,Nx,t0,t1,Nt,a,c,theta,zeta,...
    ic1,ic2,force,bccode,bcv1,bcv2)

% u_tt - a*u_xx + c*u =f
% approximation P1 of the wave equation with the Newmark method
% Q = (t0,t1)x(x0,x1)
% Nt: number of steps in time
% Nx: number of steps in 1 dimensional space
% theta, zeta: parameters of the Newmark method
% force is a string depending of x and t
% icond1 and icond2 are strings depending of x
% bccode:   vector of two elements(for each boundary point):
%           0: Dirichlet boundary conditions
%           1: Neuman boundary conditions
% bcv1 and bcv2 are strings depending of t
% this program needs the other programs: int_f.m, dcentred.m
% examples:
% newmarkbybeni(0,1,40,0,2,40,1,.5,.5,.25,'x.^2','1+0*x',...
%   '-x.*(1-x)',[0 0],'sin(2*pi*t)','cos(2*pi*t)')
% newmarkbybeni(0,1,40,0,2,40,1,.5,.5,.25,'5*x.*(x-1)',...
%   '1+0*x','-x.*(1-x)',[0 0],'sin(2*pi*t)','0*t')

% --- test for boundary and intials conditions ---
t=t0;
bc1=eval(bcv1);
bc2=eval(bcv2);
x=x0;
ic1x0=eval(ic1);
x=x1;
ic1x1=eval(ic1);
x=x0;

if (bccode(1)==0 & bccode(2)==0)                % if Dirichlet B.C.
    if (bc1~=ic1x0 | bc2~=ic1x1)
        disp('bounary conditions don"t accord to initials conditions');
        return;
    end
end

if (Nx<=2)
    disp('Nx must be bigger than 2'); return;
end


% --- initialisation of global constants ---
h=(x1-x0)/Nx;
```

```
deltat=(t1-t0)/Nt;
x=[x0:h:x1];
time=[t0:deltat:t1];

e=ones(Nx+1,1);
em(2:Nx,1)=ones(Nx-1,1);
em(1,1)=1/2; em(Nx+1,1)=1/2;


M=h*spdiags([1/6*e 2/3*em 1/6*e],-1:1,Nx+1,Nx+1);
A=a*1/h*spdiags([-e 2*em -e],[-1:1],Nx+1,Nx+1);
C=c*M;

Mred=M(2:Nx,2:Nx);
Mrel=M(2:Nx,:);

AC=A+C;
ACrel=AC(2:Nx,:);
A_u=M+zeta*deltat^2*AC;

U=zeros(Nx+1,Nt+1);
V=zeros(Nx+1,Nt+1);
U(:,1)=eval(ic1)';
u=U(:,1);
V(:,1)=eval(ic2)';
v=V(:,1);

% ++++++++++++++++++++++++++++++++++++
% --- dirichlet boudary conditions ---
% ++++++++++++++++++++++++++++++++++++

if (bccode(1)==0 & bccode(2)==0)

    % --- initialisation of U ---
    t=time;
    U(1,:)=eval(bcv1);
    U(Nx+1,:)=eval(bcv2);

    % --- approximation of v on the bound ---
    t=[time t1+deltat];
    b1=eval(bcv1);
    b2=eval(bcv2);
    t=t0;
    V(1,2:Nt+1)=dcentred(b1,deltat,Nt);
    V(Nx+1,2:Nt+1)=dcentred(b2,deltat,Nt);

    A_ured=A_u(2:Nx,2:Nx);
    A_urel=A_u(2:Nx,:);
    A1=A_u(2:Nx,1);
    A2=A_u(2:Nx,Nx+1);
```

```
    % --- step 0 ---
    b_u=A_urel*u;
    b_v=Mrel*v;
    AC_u=ACrel*u;
    v=v(2:Nx,1);

    % --- steps 1 to Nt ---
    for n=1:Nt
        t=t0+n*deltat;                                  % t_(n+1)
        F11=int_f(force,Nx,h,x0,t0+(n-1)*deltat);
        F22=int_f(force,Nx,h,x0,t0+n*deltat);
        F1=F11(2:Nx)';
        F2=F22(2:Nx)';

        b_u=b_u-deltat^2/2*AC_u+deltat*b_v+deltat^2*(zeta*F2+(.5-zeta)*F1);
        b_uu=b_u-U(1,n+1)*A1-U(Nx+1,n+1)*A2;

        u(2:Nx)=A_ured\b_uu;                      % solve A_u*u=b_u
        u(1)=U(1,n+1);
        u(Nx+1)=U(Nx+1,n+1);

        b_v=b_v+deltat*(theta*F2+(1-theta)*F1-(1-theta)*AC_u);
        AC_u=ACrel*u;
        b_v=b_v-deltat*theta*AC_u;

        b_vh=b_v-V(1,n+1)*Mrel(:,1)-V(Nx+1,n+1)*Mrel(:,Nx+1);
        v=Mred\b_vh;

        U(2:Nx,n+1)=u(2:Nx);
        V(2:Nx,n+1)=v;
    end


% ++++++++++++++++++++++++++++++++++
% --- neumann boundary conditions ---
% ++++++++++++++++++++++++++++++++++

elseif(bccode(1)==1 & bccode(2)==1)

    L1(1:Nx+1,1)=zeros(Nx+1,1);
    L2(1:Nx+1,1)=zeros(Nx+1,1);

    % --- step 0 ---
    b_u=A_u*u;
    b_v=M*v;
    AC_u=AC*u;
    L1(1,1)=-a*eval(bcv1);
    L1(Nx+1,1)=a*eval(bcv2);
```

```
    % --- steps 1 to Nt ---
    for n=1:Nt
       t=t0+n*deltat;                                    % t_(n+1)
       F1=int_f(force,Nx,h,x0,t0+(n-1)*deltat)';
       F2=int_f(force,Nx,h,x0,t0+n*deltat)';
       L2(1,1)=-a*eval(bcv1);
       L2(Nx+1,1)=a*eval(bcv2);

       b_u=b_u-deltatˆ2/2*AC_u+deltat*b_v+...
           deltatˆ2*(zeta*(F2+L2)+(.5-zeta)*(F1+L1));

       u=A_u\b_u;                                  % solve A_u*u=b_u

       b_v=b_v+deltat*(theta*(F2+L2)+(1-theta)*(F1+L1)-(1-theta)*AC_u);
       AC_u=AC*u;
       b_v=b_v-deltat*theta*AC_u;

       v=M\b_v;

       U(:,n+1)=u;
       V(:,n+1)=v;

       L1=L2;

   end
else
    disp('Other boundary conditions not yet implemented');
    return;
end

% --- visualization of the results ---
figure(1);
surf(U);
title('aprroximation of u');
xlabel('time');
ylabel('[x0,x1]');
figure(2);
surf(V);
title('approximation of u_t');
xlabel('time');
ylabel('[x0,x1]');


figure(3);

for i=1:Nt+1
    ymin(i)=min(U(:,i));
    ymax(i)=max(U(:,i));
end;
```

```
ym=min(ymin);
yM=max(ymax);

for i=1:Nt+1
    plot(x,U(:,i));
    grid;axis([x0 x1 ym yM]);
    tit=['Time: ', num2str(time(i))];
    title(tit);
    pause(.4);
end;

res=[U V];
return;
```

7.1.1 int_f.m

```
function res=int_f(f,Nx,h,x0,T)
% approximation of (f(t),phi)_L^2 with the simpson formula
% h=(x1-x0)/Nx
% t=T

t=T;
for i=2:Nx
    x=x0+h*[i-2 i-1.5 i-1 i-.5 i];
    F=eval(f);
    res(i)=h/3*(F(2)+F(3)+F(4));
end

% i==1
x=x0+h*[0 .5 1];
F=eval(f);
res(1)=h/3*(F(1)/2+F(2));

% i==Nx+1
x=x0+h*[Nx-1 Nx-.5 Nx];
F=eval(f);
res(Nx+1)=h/3*(F(2)+F(3)/2);
return;
```

7.1.2 dcentred.m

```
function y=dcentred(f,deltat,Nt)
%   approximation by centred difference
%   f:1...Nt+2 !!!! => t1+deltat
%   y(0) given by initial condition
y=zeros(1,Nt);
for i=2:Nt+1
    y(1,i-1)=(f(i+1)-f(i-1))/(2*deltat);
end;
```

7.2 The Leap-Frog Code

```
function res=leap_frog(x0,x1,Nx,t0,t1,Nt,a,c,ic1,ic2,force,bccode,...
                       bc0,bc1)

% u_tt - a*u_xx + c*u =f
% Q = (t0,t1)x(x0,x1)
% Nt: number of steps in time
% Nx: number of steps in 1 dimensional space
% force is a string depending of x and t
% icond1 and icond2 are strings depending of x
% bccode: vector of two elements(for each boundary point),
%         0: Dirichlet boundary conditions
%         1: Neuman boundary conditions
% bcv1 and bcv2 are strings depending of t
% example:
% leap_frog(0,1,20,0,2,80,.8,1,'x-1','1+0*x','0*x',[0 0],...
%           '-cos(2*pi*t)','2*t')

% --- test for boundary and intials conditions ---
t=t0;
bc0_t0=eval(bc0);
bc1_t0=eval(bc1);
x=x0;
ic1x0=eval(ic1);
x=x1;
ic1x1=eval(ic1);
x=x0;

if (bccode(1)==0 & bccode(2)==0)                          % dirichlet
    if (bc0_t0~=ic1x0 | bc1_t0~=ic1x1)
        disp('bounary conditions don"t accord to initials conditions');
        bc0_t0
        bc1_t0
        ic1x0
        ic1x1
        return;
    end
end

if (Nx<=2)
    disp('Nx must be bigger than 1'); return;
end



% --- initialisation of global constants --- %

h=(x1-x0)/Nx;
deltat=(t1-t0)/Nt;
```

```
x=[x0:h:x1];
time=[t0:deltat:t1];
landa=deltat/h;

if(abs(a)>1/landa)
    disp('CFL not respected!!! The leap-frog method will be unstable');
end

u=zeros(Nx+1,1);
U=zeros(Nx+1,Nt+1);
U(:,1)=eval(ic1)';
u=U(:,1);
v=eval(ic2)';

f=zeros(Nx+1,1);
d=zeros(Nx+1,1);
e=zeros(Nx+1,1);

% ------------------------------------ %
% --- Dirichlet boundary conditions --- %
% ------------------------------------ %

if (bccode(1)==0 & bccode(2)==0)

% --- step 1 --- %

    t=t0;
    f=eval(force)';
    d=u;
    d_pos(1:Nx-1,1)=d(3:Nx+1);
    d_neg(1:Nx-1,1)=d(1:Nx-1);
    u(2:Nx)=deltat*v(2:Nx)+(1-a*landa^2-c*deltat^2/2)*d(2:Nx)+...
        a*landa^2/2*(d_pos(1:Nx-1)+d_neg(1:Nx-1))+deltat^2/2*f(2:Nx);
    t=t0+deltat;
    u(1)=eval(bc0);
    u(Nx+1)=eval(bc1);
    U(:,2)=u;

% --- step n --- %

    for n=2:Nt
        t=n*deltat;

        f=eval(force)';
        e=d;
        d=u;
        d_pos(1:Nx-1)=d(3:Nx+1)';
        d_neg(1:Nx-1)=d(1:Nx-1)';

        u(2:Nx)=-e(2:Nx)+(2-2*a*landa^2-c*deltat^2)*d(2:Nx)+...
```

```
            a*landa^2*(d_neg+d_pos)+deltat^2*f(2:Nx);

        t=(n+1)*deltat;
        u(1)=eval(bc0);
        u(Nx+1)=eval(bc1);

        U(:,n+1)=u;
    end

% ---------------------------------- %
% --- Neuman boundary conditions --- %
% ---------------------------------- %

elseif (bccode(1)==1 & bccode(2)==1)

% --- step 1 --- %
    t=t0;
    f=eval(force)';
    d=u;
    d_pos(1:Nx-1,1)=d(3:Nx+1);
    d_neg(1:Nx-1,1)=d(1:Nx-1);
    u(1)=deltat*v(1)+(1-a*landa^2-c*deltat^2/2)*d(1)+...
        a*landa^2*(d(2)-h*bc0_t0)+deltat^2/2*f(1);
    u(Nx+1)=deltat*v(Nx+1)+(1-a*landa^2-c*deltat^2/2)*d(Nx+1)+...
        a*landa^2*(d(Nx)+h*bc1_t0)+deltat^2/2*f(Nx+1);
    u(2:Nx)=deltat*v(2:Nx)+(1-a*landa^2-c*deltat^2/2)*d(2:Nx)+...
        a*landa^2/2*(d_neg+d_pos)+deltat^2/2*f(2:Nx);
    U(:,2)=u;

% --- step n --- %

    for n=2:Nt
        t=n*deltat;

        f=eval(force)';
        e=d;
        d=u;
        d_pos(1:Nx-1)=d(3:Nx+1)';
        d_neg(1:Nx-1)=d(1:Nx-1)';
        bc0_tn=eval(bc0);
        bc1_tn=eval(bc1);
        u(1)=-e(1)+2*(1-a*landa^2-c*deltat^2)*d(1)+...
            2*a*landa^2*(d(2)-h*bc0_tn)+deltat^2*f(1);
        u(Nx+1)=-e(Nx+1)+2*(1-a*landa^2-c*deltat^2)*d(Nx+1)+...
            2*a*landa^2*(d(Nx)+h*bc1_tn)+deltat^2*f(Nx+1);
        u(2:Nx)=-e(2:Nx)+2*(1-a*landa^2-c*deltat^2)*d(2:Nx)+...
            a*landa^2*(d_neg+d_pos)+deltat^2*f(2:Nx);

        U(:,n+1)=u;
    end
```

```
end

figure(1);

for i=1:Nt+1  %length(time)
    ymin(i)=min(U(:,i));
    ymax(i)=max(U(:,i));
end;

ym=min(ymin);
yM=max(ymax);

for i=1:Nt+1  %length(time)
    plot(x,U(:,i));
    grid;axis([x0 x1 ym yM]);
    tit=['Time: ', num2str(time(i))];
    title(tit);
    pause(.4);
end;

figure(2)
surf(U);
title('approximation of u');
xlabel('time');
ylabel('[x0,x1]');
res=U;
```

7.3 The Fluid-Structure Interaction Code

All the codes which are used for running the code of the fluid-structure interaction problem is to large to put in this report.

Aknowledgements

A big THANKS to Simone Deparis, who was always available for questions and showed me a lot of useful things. I would also thank Prof. F. Saleri, who spent his time to receive me in Milano and introduce me into *MLife* and Prof. A. Quarteroni, who made it possible to have the opportunity to do this project.

References

[1] Alfio Quarteroni, Luca Formaggia, *Mathematical Modelling and Numerical Simulation of the Cardiovascular System*, 2002

[2] Simone Deparis, Miguel Ángel Fernández, Luca Formaggia and Fabio Nobile, *Acceleration of a fixed point algorithm for fluid-structure interaction using transpiration conditions*, submitted to CRAS, 2002

[3] Alfio Quarteroni, Alberto Valli, *Numerical Approximation of Partial Differential Equations*, Springer, 1997