Eindhoven University of Technology

BACHELOR

Suitability of the bicharacteristics method for computing the linearised Euler equations

Brekelmans, Jan

Award date:
2018

Link to publication

Disclaimer
This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

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.

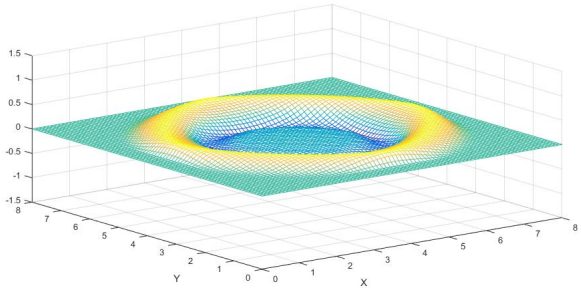
- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain

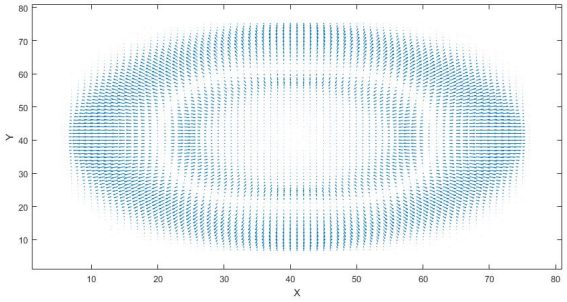
Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 25. Mar. 2025

Suitability of the bicharacteristics method for computing the linearised Euler equations

Author: Jan Brekelmans, 0777149
Supervisors: A.S. Tijsseling & B. Koren
BSc Thesis, August 2018





Department of Mathematics and Computer Science, Eindhoven University

Contents

1 Introduction

In this report we will investigate the suitability of the method of bicharacteristics for computing the linearised flow equations. The report is divided is in three parts. In the first part the method of characteristics for one dimensional scalar and systems of equations is treated. At the end a numerical scheme for the 1D frictionless water hammer will be derived, which will be used to compare the later numerical schemes with, as this is a severe test case.
In the second part, the numerical analysis methods used are given, along with an introduction example, the characteristics like method. As this method is not a true example of the method of bicharacteristics, it is very useful as an introductionto the more difficult method of bicharacteristics. The final part of the report is the theory about the bicharacteristics method. After that, two different approaches are given, one which is based on a finite number of bicharacteristics, and one which takes all the bicharacteristics in consideration.
For the characteristics like method and the bicharacteristic method three test cases are considered. They are aimed at determining how suitable the methods are for computing the linearised flow equations. The first case will be a local pressure rise, given by a bump function. The second will be a simple solution to the wave equation, which will be compared with the analytical solution. In both of these cases the initial conditions are infinitely differentiable, so not many problems are expected. The last case will be a two dimensional simulation of the water hammer equations, which will be compared with the one dimensional water hammer equation solution, and suitability of these methods will be determined from this.

2 The Euler and water hammer equations

In this section we will state the Euler equations for slightly compressible flow and the water hammer equations are derived. The derivation of the Euler equations can be found in many books, like [1], and the derivation of the one dimensional water hammer equations can be found in [2].

2.1 Euler equations

The Euler equations relate velocity, pressure and density in a flowing fluid. An equation of state is needed to relate the pressure to the density to complete the Euler equations.
As we will consider the Euler equations for liquids, we will assume that the medium is slightly incompressible.

2.1.1 Compressible flow in Cartesian coordinates

The two-dimensional compressible flow equations in Cartesian coordinates are given by

$$\begin{aligned}
&\frac{\partial \rho}{\partial t} + \frac{\partial(\rho u)}{\partial x} + \frac{\partial(\rho v)}{\partial y} = 0 \\
&\frac{\partial p}{\partial x} + \rho\left(\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y}\right) = 0 \\
&\frac{\partial p}{\partial y} + \rho\left(\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y}\right) = 0
\end{aligned}$$

The first equation is the continuity equation, and the other two are respectively the x-momentum equation and the y-momentum equation.
We supplement these equations with the equation for the speed of sound c in a medium:

$$c = \sqrt{\frac{dp}{d\rho}}.$$

Combining the Euler equations with the speed of sound equation results in the slightly compressible Euler equations:

$$\begin{aligned}
&\frac{\partial p}{\partial t} + u\frac{\partial p}{\partial x} + v\frac{\partial p}{\partial y} + \rho c^2\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right) = 0 \\
&\frac{\partial p}{\partial x} + \rho\left(\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y}\right) = 0 \\
&\frac{\partial p}{\partial y} + \rho\left(\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y}\right) = 0
\end{aligned}$$

Linearising this equation around the steady state

$$p_0, u_0, v_0 \text{ constant}$$

and a prescribed initial density state

$$\rho = \rho_0$$

gives the linearised two dimensional Euler equations

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

2.2 Water hammer equations

The water hammer equations are given by the following system:

$$\frac{\partial p}{\partial t} + u \frac{\partial p}{\partial x} + \rho c^2 \frac{\partial u}{\partial x} = 0 \qquad (2)$$

$$\frac{\partial u}{\partial t} + V \frac{\partial u}{\partial x} + \frac{1}{\rho} \frac{\partial p}{\partial x} + g \sin \alpha + \frac{f u |u|}{2D} = 0. \qquad (3)$$

Here, c is given by

$$c^2 = \frac{\frac{K}{\rho}}{1 + \frac{DK}{eE}},$$

u is the cross-sectional average of the axial flow velocity and f is the Darcy-Weisbach friction factor. K is the bulk modulus of elasticity of the fluid, D the diameter of the conduit, e the thickness of the conduit walls, and E the Young's modulus of the conduit, and α the upward angle which the pipe makes with the horizontal.

Part I
Method of characteristics

3 Method of characteristics

The method of characteristics is a method to solve first-order partial differential equations. This method transforms the PDE into a family of ordinary differential equations along curves in the domain, which then can be solved to obtain the solution.
The method of characteristics is straightforward to generalize to a system of hyperbolic partial differential equations, as will be done in the second part of this report.

3.1 Scalar case

For the scalar case we will take the same approach as taken [3]. in We consider the non-linear first order partial differential equation

$$\begin{cases} F(\nabla u, u, \mathbf{x}) = 0 & \text{in } U \\ u = g & \text{on } \Gamma, \end{cases} \tag{4}$$

where $\Gamma \subset \partial U$ and $g : \Gamma \to \mathbb{R}$ are given, with F and g smooth functions and $U \subset \mathbb{R}^n$, with $\mathbf{x} = (x_1, \ldots, x_n) \in U$. ∇u is the vector given by

$$\nabla u = \left(\frac{\partial u}{\partial x_1}, \ldots, \frac{\partial u}{\partial x_n} \right)$$

The idea of the method of characteristics is to convert the partial differential equation (4) into a system of first-order ordinary differential equations, where for each ordinary differential equation there is a curve in U associated to it. From these ordinary differential equations and corresponding curves the solution of (4) can be constructed. We will take the same approach as in Evans [3].

We suppose that the boundary Γ is a subset of $\{x_n = \alpha\}$, for α constant, so that Γ is assumed flat near all points $x_0 \in \Gamma$. Furthermore, we assume that the solution to (4) is C^2.

We will consider curves in (a subset of) U of the form:

$$\begin{cases} \mathbf{x}(s) = \mathbf{x}(y, s) \\ \mathbf{p}(s) = \mathbf{p}(y, s) = \nabla \mathbf{x}(s) \\ z(s) = z(y, s) = u(s) \end{cases}$$

where $y = (y_1, \dots, y_{n-1}, 0) \in \Gamma$, to denote the dependence on y. The functions $\mathbf{x}(s), \mathbf{p}(s)$ and $z(s)$ are called the characteristics, and the function $\mathbf{x}(s)$ is called the projected characteristic.
These curves are defined as the solution of the so called characteristic ODE

$$\begin{cases} \frac{dx_j}{ds}(s) = \frac{\partial F}{\partial p_j}(\mathbf{p}(s), z(s), x(s)) & j = 1, \dots, n \\ \frac{dz}{ds}(s) = \sum_{j=1}^{n} p_j(s) \frac{\partial F}{\partial p_j}(\mathbf{p}(s), z(s), \mathbf{x}(s)) & \\ \frac{dp_j}{ds}(s) = -\frac{\partial F}{\partial x_j}(\mathbf{p}(s), z(s), \mathbf{x}(s)) - \frac{\partial F}{\partial z}(\mathbf{p}(s), z(s), \mathbf{x}(s)) p_j(s) & j = 1, \dots, n \end{cases} \tag{5}$$

with admissible initial conditions, defined next.
Assume that the triple (p^0, z^0, x^0), given by $x^0 \in \Gamma$ and

$$\begin{cases} p^0 = \mathbf{p}(x^0, 0) \\ z^0 = z(x^0, 0) = g(x^0) \end{cases}$$

is admissible, thus satisfies

$$\begin{cases} p_i^0 = \frac{\partial g}{\partial x_i}(x_0) & i = 1, \dots, n-1 \\ F(p^0, z^0, x^0) = 0 \end{cases}$$

and is non-characteristic, so is admissible and satisfies

$$\frac{\partial F}{\partial p_n}(p^0, z^0, x^0) \neq 0.$$

Theorem 3.1 (Local invertibility)**.**
There exists an open interval $I \subset \mathbb{R}$ containing 0, a neighbourhood W of x^0 in Γ, and a neighbourhood V of x^0 in $\mathbb{R}^n$, such that for each $x \in V$, there exists unique $s \in I$, $y \in W$ such that

$$x = \mathbf{x}(y, s).$$

The mappings $x \mapsto s, y$ are C^2.

For each $x \in V$, the equation

$$x = \mathbf{x}(y, s)$$

can be locally and uniquely solved for $y = \mathbf{y}(x)$ and $\mathbf{s} = \mathbf{s}(x)$.
Define

$$\begin{cases} u(x) = z(\mathbf{y}(x), \mathbf{s}(x)) \\ \mathbf{p}(x) = (\mathbf{y}(x), s(x)) \end{cases}$$

for $x \in V$.

Theorem 3.2 (Local existence theorem)**.**
The function $u(x)$ defined above is C^2 and solves

$$F(\nabla u, u, x) = 0,$$

for $x \in V$, with the boundary condition

$$u(x) = g(x)$$

for $x \in \Gamma \cap V$.

So what the above theorem says: if a point in U is sufficiently close to a non-characteristic point on Γ, then the solution $u(x)$ can be found with the characteristic ODE (5).

3.1.1 Quasi-linear partial differential equations

A quasi-linear partial differential equation is of the form

$$\begin{cases} \sum_{i=1}^{n} f_i(\mathbf{x}, u)\frac{\partial u}{\partial x_i} + c(x, u) = 0 & \mathbf{x} \in U \subset \mathbb{R}^n \\ u(\mathbf{x}) = g(\mathbf{x}) & \mathbf{x} \in \Gamma \subset \mathbb{R}^{n-1} \times \{x_n = 0\} \end{cases} \tag{6}$$

Using the results of the previous subsection, we get the following requirement for the non-characteristic condition for an admissible triple (p^0, z^0, x^0) on the boundary:

$$f_n(x^0, z^0) \neq 0,$$

so the initial conditions for solving the PDE do not have to depend on any partial derivatives of the solution.
A part of the characteristic ODE becomes:

$$\begin{cases} \frac{dx_j}{ds}(s) = f_j(\mathbf{x}, z) & j = 1, \ldots, n \\ \frac{dz}{ds}(s) = \sum_{j=1}^{n} f_j(\mathbf{x}, z)p_j(s) = -c(\mathbf{x}, z) \end{cases} \tag{7}$$

From this we see that $x(s)$ and $z(s)$ do not depend on $\mathbf{p}(s)$, and so the solution $u(x)$ can be solved independently of $\mathbf{p}$, and we will refer (7) as the reduced characteristic ODE.

3.2 One-dimensional system of equations

We consider a hyperbolic quasi-linear system of n equations in one space variable, thus

$$\begin{cases} \frac{\partial u_i}{\partial t} + \sum_{j=1}^{n} f_j^i(x, t, \mathbf{u})\frac{\partial u_j}{\partial x} + c_i(x, t, \mathbf{u}) = 0 \\ u_i(0, t, \mathbf{u}) = u_i^0(t). \end{cases}$$

for $i = 1, \ldots, n$.
This system can be written in vector notation as follows:

$$\mathbf{u}_t + \mathbf{A}\mathbf{u}_x + \mathbf{c} = \mathbf{0}, \tag{8}$$

where

$$\mathbf{u} = \begin{pmatrix} u_1 \\ \vdots \\ u_n \end{pmatrix}, \quad \mathbf{A} = \begin{pmatrix} f_1^1 & \cdots & f_n^1 \\ \vdots & & \vdots \\ f_1^n & \cdots & f_n^n \end{pmatrix}, \quad \mathbf{c} = \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix},$$

where the subscripts t and x are partial derivatives with respect to t and x respectively.
This system of partial differential equations is hyperbolic if $\mathbf{A}$ has only real eigenvalues and is diagonalizable.

3.2.1 Diagonalization of the system of PDE's

The normal way to bring this system in characteristic form is to diagonalize $\mathbf{A}$ in the form

$$\mathbf{A} = \mathbf{P}^{-1}\mathbf{\Gamma}\mathbf{P},$$

where the $\mathbf{\Gamma}$ is a diagonal matrix with the eigenvalues of $\mathbf{A}$ on the diagonal, so

$$\mathbf{\Gamma} = \begin{pmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{pmatrix}$$

and $\mathbf{P}$ is matrix which columns are eigenvectors of $\mathbf{A}$, where the i-th column corresponds to the i-th diagonal element of $\mathbf{\Gamma}$.
The characteristic variable $\mathbf{w} = \mathbf{P}\mathbf{u}$ is introduced, so that

$$\begin{aligned} & \mathbf{P}(\mathbf{u}_t + \mathbf{A}\mathbf{u}_x + \mathbf{c}) \\ &= \mathbf{P}\mathbf{u}_t + \mathbf{P}\mathbf{A}\mathbf{u}_x + \mathbf{P}\mathbf{c} \\ &= \mathbf{P}\mathbf{u}_t + \mathbf{P}\mathbf{P}^{-1}\mathbf{\Gamma}\mathbf{P}\mathbf{u}_x + \mathbf{P}\mathbf{c} \\ &= \mathbf{P}\mathbf{u}_t + \mathbf{\Gamma}\mathbf{P}\mathbf{u}_x + \mathbf{P}\mathbf{c} \\ &= \mathbf{w}_t + \mathbf{\Gamma}\mathbf{w}_x + \tilde{\mathbf{c}} \\ &= \mathbf{P}\mathbf{0} = \mathbf{0}, \end{aligned}$$

where

$$\tilde{\mathbf{c}} = \mathbf{P}\mathbf{c}(x, t, \mathbf{u}) = \mathbf{P}\mathbf{c}(x, t, \mathbf{P}^{-1}\mathbf{w}).$$

This transforms the system of equations into a set of scalar equations, which can be individually solved in the terms w_i by the scalar 1D method of characteristics, after which the original functions $\mathbf{u}$ can be calculated back by $\mathbf{u} = \mathbf{P}^{-1}\mathbf{w}$.

3.2.2 Multiplier method

An equivalent manner of solving (8) is using a multiplier method. We denote the i-th equations of (8) by L_i, so

$$L_i : \quad \frac{\partial u_i}{\partial t} + \sum_{j=1}^{n} f_j^i(x,t,\mathbf{u}) \frac{\partial u_j}{\partial x} + c_i(x,t,\mathbf{u}) = 0,$$

and introduce $n-1$ multipliers, λ_2 to λ_n.
Consider the equation

$$L_1 + \lambda_2 L_2 + \ldots + \lambda_n L_n = 0. \tag{9}$$

Expanding, ignoring the source terms c_i, gives

$$\begin{aligned} &\frac{\partial u_1}{\partial t} + \sum_{j=1}^{n} f_j^1 \frac{\partial u_j}{\partial x} + \lambda_2 \frac{\partial u_2}{\partial t} + \sum_{j=1}^{n} \lambda_2 f_j^2 \frac{\partial u_j}{\partial x} \\ &+ \ldots + \lambda_n \frac{\partial u_n}{\partial t} + \sum_{j=1}^{n} \lambda_n f_j^n \frac{\partial u_j}{\partial x} = 0. \end{aligned}$$

Grouping the partial derivatives together gives

$$\begin{aligned} &\left(\frac{\partial u_1}{\partial t} + \lambda_2 \frac{\partial u_2}{\partial t} + \ldots + \lambda_n \frac{\partial u_n}{\partial t} \right) \\ &+ (f_1^1 + \lambda_2 f_1^2 + \ldots + \lambda_n f_1^n) \frac{\partial u_1}{\partial x} + \ldots \\ &+ (f_n^1 + \lambda_2 f_n^2 + \ldots + \lambda_n f_n^n) \frac{\partial u_n}{\partial x} = 0. \end{aligned} \tag{10}$$

For applying the method of characteristics this equation has to be of the form

$$\left(\frac{\partial u_1}{\partial t} + \gamma_2 \frac{\partial u_2}{\partial t} \ldots + \gamma_n \frac{\partial u_n}{\partial t} \right) + s \left(\frac{\partial u_1}{\partial x} + \gamma_2 \frac{\partial u_2}{\partial x} \ldots + \gamma_n \frac{\partial u_n}{\partial x} \right) = 0,$$

where s is the characteristic speed and $\left(\frac{\partial u_1}{\partial t} + \gamma_2 \frac{\partial u_2}{\partial t} \ldots + \gamma_n \frac{\partial u_n}{\partial t} \right)$ a linear combination of the partial derivatives of u.
Setting $\mu_i = (f_i^1 + \lambda_2 f_i^2 + \ldots + \lambda_n f_i^n)$, we obtain the following form of equation (10):

$$\begin{aligned} &\left(\frac{\partial u_1}{\partial t} + \lambda_2 \frac{\partial u_2}{\partial t} + \ldots + \lambda_n \frac{\partial u_n}{\partial t} \right) \\ &+ \mu_1 \left(\frac{\partial u_1}{\partial x} + \frac{\mu_2}{\mu_1} \frac{\partial u_2}{\partial x} + \ldots + \frac{\mu_n}{\mu_1} \frac{\partial u_n}{\partial x} \right) = 0. \end{aligned}$$

This is in characteristic form if it satisfies the following system:

$$\begin{cases} \frac{\mu_2}{\mu_1} = \lambda_2 \\ \ldots \\ \frac{\mu_n}{\mu_1} = \lambda_n. \end{cases}$$

If the system does not contain decoupled equations, this can be solved for the λ_i, which gives n different sets of solutions $S^j = (\lambda_2^j, \ldots, \lambda_n^j)$, $j = 1, \ldots, n$. Each S^j gives a linear combination

$$w^j = u_1 + \lambda_2^j u_2 + \ldots + \lambda_n^j u_n$$

and a characteristic speed $s^j = \mu_1(S^j)$, which gives the simple partial differential equation

$$\frac{\partial w^j}{\partial t} + s^j \frac{\partial w^j}{\partial x} = 0,$$

which can be easily solved with the method of characteristics for scalar equations.

Example 3.3. *Consider the system:*

$$\begin{pmatrix} u \\ v \end{pmatrix}_t + \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix}_x + \begin{pmatrix} 2 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

We denote the equations by:

$$\begin{aligned} L_0: &\quad u_t + v_x + 2 = 0 \\ L_1: &\quad v_t + u_x + 1 = 0. \end{aligned}$$

Consider the following equation

$$\begin{aligned} & L_0 + \lambda L_1 \\ &= u_t + v_x + 2 + \lambda v_t + \lambda u_x + \lambda \\ &= (u_t + \lambda v_t) + (\lambda u_x + v_x) + (2 + \lambda) \\ &= (u_t + \lambda v_t) + \lambda(u_x + \frac{1}{\lambda} v_x) + (2 + \lambda) = 0 \end{aligned}$$

Following the multiplier method, we have that

$$\lambda = \frac{1}{\lambda},$$

which has the solutions $\lambda = \pm 1$, and transforms the system into the following set of scalar equations:

$$\begin{cases} L_0 + L_1 = (u_t + v_t) + (u_x + v_x) + 3 = \frac{dw^1}{dt} + \frac{dw^1}{dt} + 3 = 0 \\ L_0 - L_1 = (u_t - v_t) - (u_x - v_x) + 1 = \frac{dw^2}{dt} - \frac{dw^2}{dt} + 1 = 0 \end{cases},$$

where $\frac{dx}{dt} = 1$ in the first equation, and $\frac{dx}{dt} = -1$ in the second one.

4 Characteristics-based finite-difference methods

4.1 One-dimensional scalar problem

We consider the partial differential equation given by

$$\begin{cases} \frac{\partial u}{\partial t} + f(u,x,t)\frac{\partial u}{\partial x} + c(u,x,t) = 0 & (x,t) \in U \subset \mathbb{R} \times (0,\infty) \\ u(x,0) = g(x) & x \in \Gamma \subset \mathbb{R} \times \{t = 0\} \end{cases} \tag{11}$$

U is taken as a rectangle of the form $U = (x_l, x_r) \times (0,T)$, and the left and right boundary conditions $u_l(t) = u(x_l,t)$ and $u_r(t) = u(x_r,t)$ are given for all $t \in (0,T)$.
A finite-difference method based on the method of characteristics is given which will be suitable for constant coefficient hyperbolic equations with a sufficiently smooth solution, which we will assume for this section.

4.1.1 General approach

The general approach for finite difference based characteristics is tracing back the characteristics in time, illustrated by the following picture:

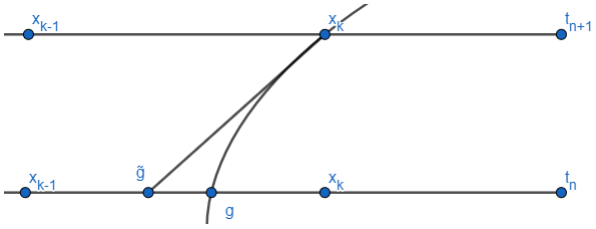


Figure 1: Characteristic curve and approximation

The characteristic curve through the point (x_k, t_{n+1}) is approximated by a straigth line, for which the crossing line of this curve with the line $(x, t = t_n)$ is a reasonable approximation of the point g by $\tilde{g}$, where g is the crossing point of the characteristic curve with the line $(\cdot, t_n)$, and $\tilde{g}$ the crossing point of the approximation curve with the same line $(x, t = t_n)$.
For a constant coefficient equation, or an equation where $f(u,x,t)$ only depends on x and t, the characteristic curve can be solved exactly, and the computational grid can be chosen such that it passes through the computational points.

4.1.2 The Courant-Isaacson-Rees method

The simplest example of a characteristics-based method is the Courant-Isaacson-Rees method. Consider problem (11) with $c \equiv 0$, and consider the grid variable v_k^{n+1}. Suppose that $f(v_k^n, x_k, t_n) > 0$, then the characteristic curve going

through (x_k, t_{n+1}) will cross the line $(x, t = t_n)$ between (x_{k-1}, t_n) and (x_k, t_n), given that Δt_{n+1} is sufficiently small.
As an approximation to the characteristic curve we will take the function

$$\gamma(t) = x_k - f(v_k^n, x_k, t_n)(t - t_{n+1}).$$

Then $\gamma(t_n) = x_k - f(v_k^n, x_k, t_n)\ \Delta t_{n+1}$, and we will use linear interpolation to estimate v_k^{n+1} to get

$$\begin{aligned} v_k^{n+1} &= v_k^n + (\gamma(t_n) - x_{k-1})\frac{v_k^v - v_{k-1}^n}{x_k - x_{k-1}} \\ &= v_k^n - \frac{\Delta t_{n+1}}{\Delta x} f(v_k^{n+1}, x_k, t_n)(v_k^n - v_{k-1}^n). \end{aligned}$$

Setting Δt constant, and $f(v_k^{n+1}, x_k, t_n) = f_k^n$, we get the scheme:

$$v_k^{n+1} = v_k^n - \frac{\Delta t}{\Delta x} f_k^n\ (v_k^n - v_{k-1}^n). \tag{12}$$

Other methods can be derived from this method, for instance the Lax-Wendroff method, which uses a quadratic interpolation polynomial.

4.2 One-dimensional system

The method specified in the previous subsection can be easily expanded to one-dimensional systems of equations. Consider the following system

$$\begin{cases} \frac{\partial u_i}{\partial t} + \sum_{j=1}^n f_j^i(x, t, \mathbf{u}) \frac{\partial u_j}{\partial x} + c_i(x, t, \mathbf{u}) = 0 \\ u_i(0, t, \mathbf{u}) = u_i^0(t). \end{cases}$$

With the multiplier method this can be transformed into n equations of the form

$$\frac{\partial w_i}{\partial t} + s_i \frac{\partial w_i}{\partial t} + c = 0 \qquad\qquad \text{along } \frac{dx}{dt} = s_i$$

for $j = 1, \ldots, n$, $w_i = u_1 + \sum_{i=2}^n \lambda_i u_i$ and c can depend on $x, t, u_1, \ldots, u_n$.

Integrate each of these equations from t_n to t_{n+1}, and making use of the total derivative, gives

$$w_i(x_k, t_{n+1}) - w_i(g_i, t_n) + \int_{t_{n-1}}^{t_n} c\ dt = 0,$$

for each $\frac{dx}{dt} = s_i$, where g_i is the crossing point of $x(t)$ with $(\cdot, t_n)$. Here we have assumed that λ_i is independent of $\mathbf{u}, x, t$.

Denoting $\int_{t_n}^{t_{n+1}} c \; dt$ by c_i for each case of s_i, and writing $w_i(x_k, t_n) = w_i$, $w_i(x_k, g_i) = w_i'$ gives us n equations in the n unknowns $w_1, \ldots, w_n$:

$$\begin{cases} w_1 - w_1' + c_i = 0 \\ \vdots \\ w_n - w_n' + c_n = 0. \end{cases}$$

As each w_i is a linear combination of u_i, and they are all linearly independent, this can be solved for each u_i at (x_k, t_n).

5 Method of characteristics for the frictionless water hammer equations

In this section we will consider the water hammer equations given by

$$\frac{\partial p}{\partial t} + u\frac{\partial p}{\partial x} + \rho c^2 \frac{\partial u}{\partial x} = 0$$
$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + \frac{1}{\rho}\frac{\partial p}{\partial x} + g\sin\alpha + \frac{fu|u|}{2D} = 0.$$

We will consider a pipe with a fixed diameter, so D is a constant, and assume that the pipe is horizontal, so $\alpha = 0$. Furthermore, for sake of clarity, we will denote the derivative of H with respect to y by H_y. The equations then become

$$p_t + up_x + \rho c^2 u_x = 0 \tag{13}$$
$$u_t + uu_x + \frac{1}{\rho}p_x + \frac{fu|u|}{2D} = 0.$$

The general problem we will be investigating is flow in a pipe which suddenly halts due to the closing of a valve, which is clarified in the following picture:

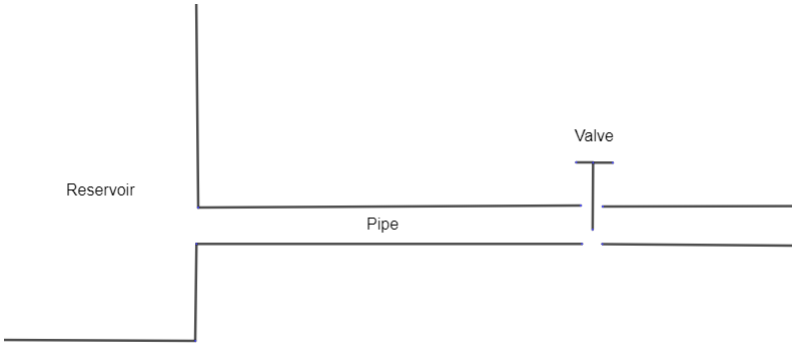
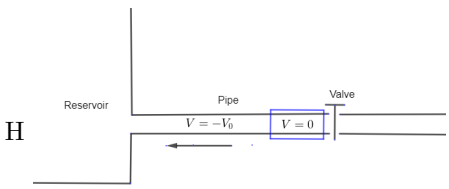
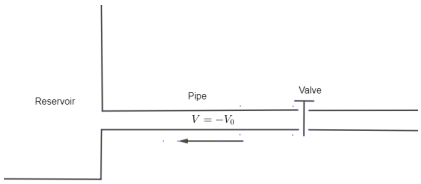


Figure 2: Setup for the water hammer effect

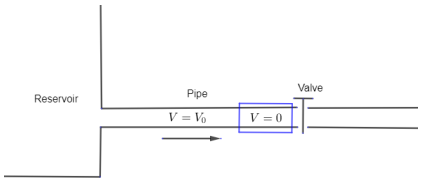
The pipeline under consideration is connected on the left side to a reservoir, and to the right side connected to a valve. The reservoir supplies a steady pressure into the pipe, and which we will consider to be constant during the simulations. At the start of the simulation, the valve will be instantly closed, so that the local fluid velocity at the valve will be zero. This will create al spring like effect, where a growing section of fluid is at rest, until it reaches the reservoir, after which the pressure differential of the pipe will induce flow towards the reservoir, which is characterised by the fluid velocity being negative. This is illustrated by the following pictures.



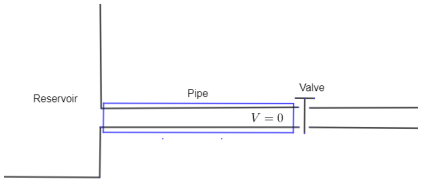
Time $t = \frac{L}{a} + \varepsilon$



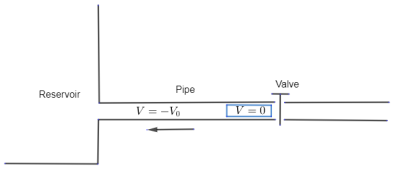
Time $t = \frac{2L}{a}$



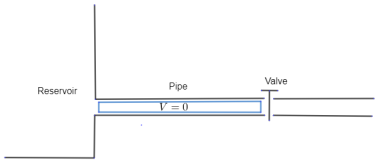
Time $t = \varepsilon$



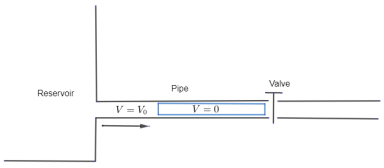
Time $t = \frac{L}{a}$



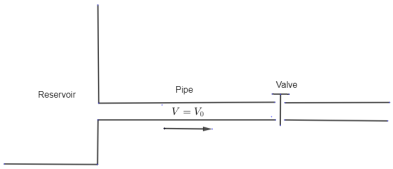
Time $t = \frac{2L}{a} + \varepsilon$



Time $t = \frac{3L}{a}$



Time $t = \frac{3L}{a} + \varepsilon$



Time $t = \frac{4L}{a}$

Figure 3: Theoretical results

The blue box around the pipe indicates the increase/decrease of the pressure and the radial expansion/contraction of the pipe. A larger box indicates a rise in pressure and an expansion of the pipe, and a smaller box indicates a decrease in pressure and a contraction of the pipe.

The case where the convective acceleration terms are ignored will be considered. The impact of those terms on the unsteady pipe flow will be investigated for the parameters:

- Density : $\rho = 1000$ kg/m^3.
- Speed of sound: $c = 1000$ m/s.

The initial flow in the pipe is 0.1 m/s, and the initial pressure is given by 300,000 Pa, and this pressure is constant where the pipe meets the reservoir. Furthermore, the velocity at the closed valve will be taken as to be zero constantly.

Furthermore, we will introduce the discharge Q, defined by

$$Q = uA,$$

where A is the the cross-sectional area of the pipe.
This quantity is useful to compare the one-dimensional model to the two-dimensional model later on.

5.1 Multiplier method

We will apply the multiplier method to (13), where the convective acceleration terms have been ignored, namely

$$\begin{aligned} L_0 : &\quad p_t + \rho c^2 u_x = 0 \\ L_1 : &\quad u_t + \frac{1}{\rho} p_x = 0. \end{aligned} \tag{14}$$

Note that f is taken to be zero for frictionless water hammer.
Now consider $L_0 + \lambda L_1$:

$$\begin{aligned} & L_0 + \lambda L_1 \\ =& p_t + \rho c^2 u_x + \lambda u_t + \lambda \frac{1}{\rho} p_x \\ =& \left(p_t + \lambda \frac{1}{\rho} p_x \right) + \frac{1}{\lambda} \left(u_t + \frac{1}{\lambda} \rho c^2 u_x \right) \\ =& (p_t + s_1 p_x) + (u_t + s_2 u_x) = 0, \end{aligned}$$

where

$$\begin{aligned} s_1 &= \lambda \frac{1}{\rho} \\ s_2 &= \frac{1}{\lambda} \rho c^2. \end{aligned}$$

Setting $s_1 = s_2$ and solving for λ gives $\lambda = \pm\rho c$, and gives us the following set of equations:

$$(p_t + cp_x) - \rho a(u_t + cu_x) = 0$$
$$(p_t + cp_x) + \rho a(u_t + cu_x) = 0.$$

along the characteristic curves $\frac{dx}{dt} = a$ and $\frac{dx}{dt} = -a$ respectively.
This can be rewritten, using the total derivatives

$$\frac{dp}{dt} = \frac{\partial p}{\partial t} + \frac{\partial p}{\partial x}\frac{dx}{dt},$$
$$\frac{du}{dt} = \frac{\partial u}{\partial t} + \frac{\partial u}{\partial x}\frac{dx}{dt},$$

as

$$\frac{dp}{dt} - \rho c\frac{du}{dt} = 0 \qquad \text{along } \frac{dx}{dt} = -a, \tag{15}$$
$$\frac{dp}{dt} + \rho c\frac{du}{dt} = 0 \qquad \text{along } \frac{dx}{dt} = a. \tag{16}$$

Consider a grid $\{x_k\}$, where $x_0 = 0$, and $x_{K+1} = L$, and $x_k = k\ \Delta x$, $k = 1, \ldots, K$.
Eq. (15) will be called the left characteristic, and Eq. (16) will be called the right characteristic.

5.2 Numerical scheme

Let x_k be an interior grid point. Integrating (15) from t_n to t_{n+1}, with $\Delta t = t_{n+1} - t_n = \frac{1}{c}\Delta x$ and requiring that $x(t_{n+1}) = x_k$ gives us

$$\left(p_k^{n+1} - \rho c u_k^{n+1}\right) - \left(p_{k+1}^n - \rho c u_{k+1}^n\right) = 0,$$

where $p_k^n = p(x_k, t_n)$, and similar for u_k^{n+1}.
Similar, for (16) we have

$$\left(p_k^{n+1} + \rho c u_k^{n+1}\right) - \left(p_{k-1}^n + \rho c u_{k-1}^n\right) = 0.$$

Combining these allows us to solve for p_k^{n+1}, u_k^{n+1}:

$$p_k^{n+1} = \frac{p_{k+1}^n + p_{k-1}^n}{2} - \frac{\rho c(u_{k+1}^n - u_{k-1}^n)}{2},$$
$$u_k^{n+1} = -\frac{p_{k+1}^n - p_{k-1}^n}{2\rho c} + \frac{u_{k+1}^n + u_{k-1}^n}{2}.$$

Now consider the point x_0, where the pipe meets the reservoir. Here the pressure p is constant, say equal to p_0, and only has the left characteristic going into it. This gives the following boundary value:

$$u_0^{n+1} = \frac{p_0 - p_1^n}{\rho c} + u_1^n.$$

Similar for the point x_{K+1}, where the pipe is connected to the valve, the velocity is equal to 0, and the pressure is given there by

$$p_{K+1}^{n+1} = p_K^n + \rho c u_K^n.$$

5.3 Numerical results

For the numerical computation we take 100 grid points, so $\Delta x = 0.2$ m, and $\Delta t = \frac{\Delta x}{c} = 2 \cdot 10^{-5}$ s.
The corresponding results for velocity u, pressure p and discharge Q are shown for different times $t = 0.01 \approx \frac{.5L}{c}$, $t = 0.03 = \frac{(1+.5)L}{c}$, $t = 0.05 = \frac{(2+0.5)L}{c}$, $t = 0.07 = \frac{(3+0.5)L}{c}$.

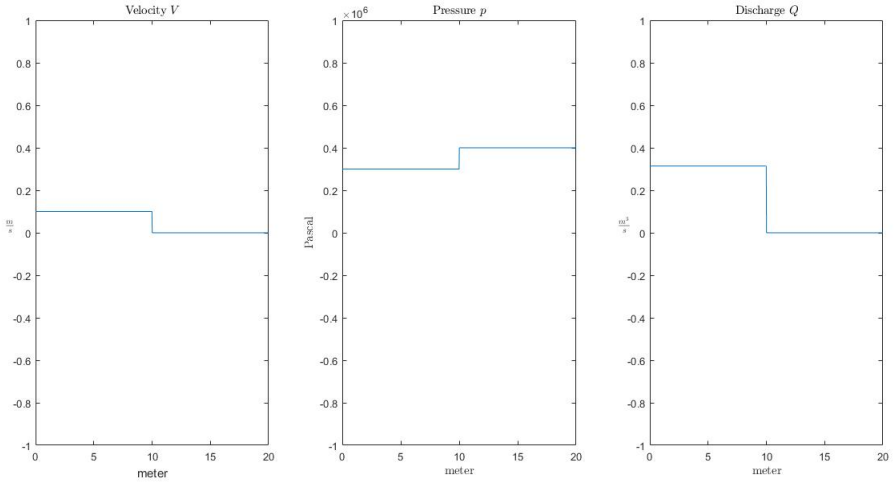


Figure 4: Time $t = 0.01$ s

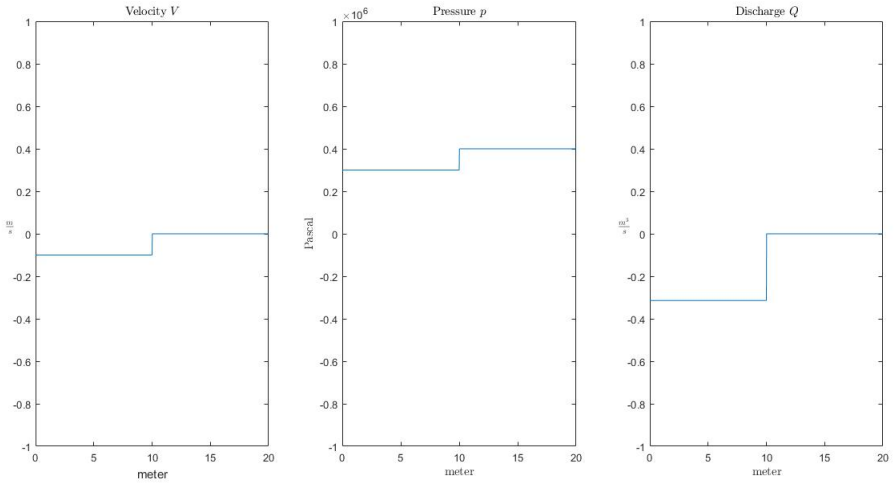


Figure 5: Time $t = 0.03$ s

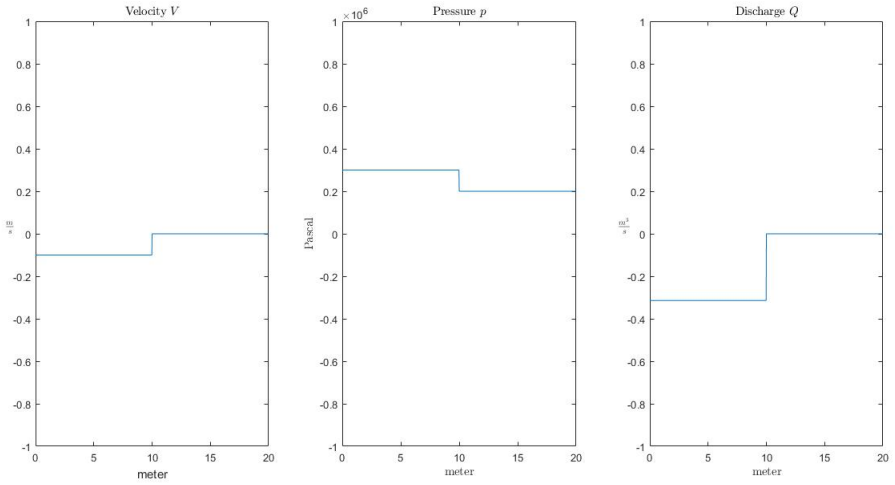


Figure 6: Time $t = 0.05$ s

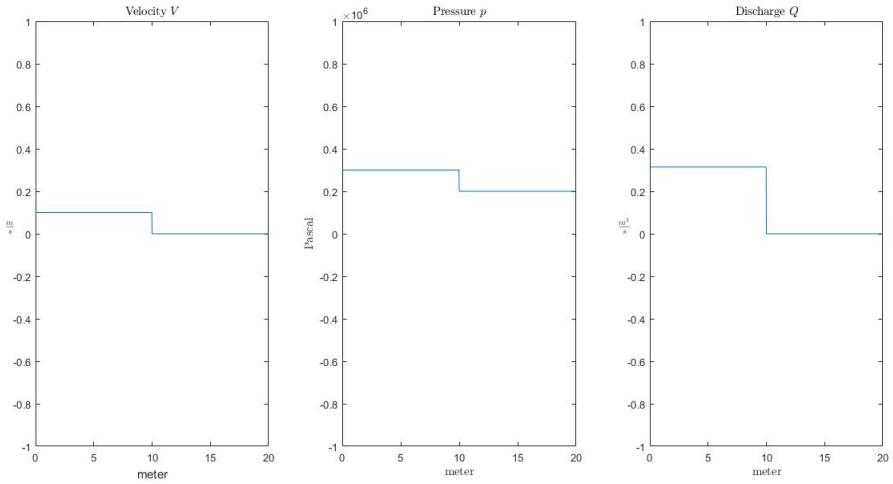


Figure 7: Time $t = 0.07$ s

The numerical results obtained match closely the theoretical results of Figure 3. This is due to the fact that the results are exact solutions of the frictionless water hammer equations, since no interpolations or approximations are used in the computations.

These results will be used in the two-dimensional case for verification of accuracy.

Part II
Introductory example

6 Numerical methods analysis

In this section basis results and theorems for convergence, accuracy and stability of the numerical methods are presented, which are used to solve the following partial differential equation:

$$\begin{cases} \frac{\partial u}{\partial t} + B_1 \frac{\partial u}{\partial x} + B_2 \frac{\partial u}{\partial y} = 0 & (x, y, t) \in U \subset \mathbb{R} \times (0, \infty) \\ u(x, y, 0) = g(x, y) & x \in \Gamma \subset \mathbb{R}. \end{cases} \tag{17}$$

We consider a computational grid V, uniformly spaced in the space variable (x, y), given by

$$V = \{(x, y, t) \mid \quad x \in [0, M],\ y \in [0, M],\ t \in [0, T]\}.$$

Here M and T are positive constants. The values of the true solution of (17) at the the point $(x_k, t_n) \in V$ are denoted by v_k^n, and the approximate values obtained by a numerical method at the same point are denoted by u_k^n.
An explicit numerical scheme is of the form

$$u_k^{n+1} = g(u_{k-l}^n, \ldots, u_k^n, \ldots, u_{k+m}^n),$$

so u_k^{n+1} depends on $m + l + 1$ previous values. We will only consider linear difference schemes, so

$$u_{k,m}^{n+1} = \sum_{i=-l_1}^{m_1} \sum_{j=-l_2}^{m_2} g_{ij} u_{k+i_1, m+i_2}^n.$$

In the rest of this section we will only consider the numerical schemes with $l_1 = l_2 = 1$, $m_1, m_2 = 1$, and will write the numerical scheme as

$$u_{k,m}^{n+1} = \sum_{i,j=-1}^{1} C_{ij} u_{k+i,m+j}^n. \tag{18}$$

As $u_{k,m}^n$ is a vector, then C_{ij} is to be considered as a vector, and $C_{ij} u_{k+i,m+j}^n$ is taken as the inner product of C_{ij} and $u_{k+i,m+j}^n$.

6.1 Consistency

A numerical scheme is called consistent of order $(\Delta t^a, \Delta x^b)$ if

$$|u_{k,m}^{n+1} - u(k\Delta x, m\Delta x, t + \Delta t)| = O(\Delta t^{a+1}, \Delta x^{b+1}),$$

and will be consistent if $a > 0$ and $b > 0$.
We will take $\Delta x = \Delta y = \frac{\Delta t}{R}$, and will compute the requirements for C_{ij} such that the numerical scheme is first or second order consistent.
Expanding (18) in a Taylor series to first order gives

$$v + \Delta t\ v_t + O(\Delta t^2) = \sum_{i,j=-1}^{1} C_{ij}\,(v + i\Delta x\ v_x + j\Delta y\ v_y) + O(\Delta t^2).$$

We have used $v = (k\Delta x, m\Delta y, n\Delta t)$, and noting that $\Delta x = \Delta y = \frac{\Delta t}{R}$ gives

$$v + \Delta t\ v_t + O(\Delta t^2) = \sum_{i,j=-1}^{1} C_{ij}\left(v + i\frac{\Delta t}{R}v_x + j\frac{\Delta t}{R}v_y\right) + O(\Delta t^2).$$

Substracting $v_t + B_1 v_x + B_2 v_y$ on the left side gives

$$v - \Delta t B_1 v_x - \Delta t B_2 v_y + O(\Delta t^2) = \sum_{i,j=-1}^{1} C_{ij} v + \Delta t \sum_{i,j=-1}^{1} i\frac{C_{ij}}{R}v_x + \Delta t \sum_{i,j=-1}^{1} \frac{C_{ij}}{R}v_y + O(\Delta t^2)$$

Equating the coefficients gives the $O(\Delta t)$-consistency condition:

$$\sum_{i,j=-1}^{1} C_{ij} = 1$$
$$\sum_{i,j=-1}^{1} iC_{ij} = -B_1 R$$
$$\sum_{i,j=-1}^{1} jC_{ij} = -B_2 R.$$

For second order consistency we need the following Taylor series equation:

$$v + \Delta t\ v_t + \frac{\Delta t^2}{2}v_{tt} + O(\Delta t^3) = \sum_{i,j=-1}^{1} C_{ij}\left(v + i\Delta x\ v_x + j\Delta y\ v_y + \frac{(i\Delta x)^2}{2})v_{xx} + ij\Delta x\Delta y\ v_{xy} + \frac{(j\Delta y)^2}{2}v_{yy}\right) + O(\Delta t^3).$$

Using the fact that

$$\begin{aligned} v_{tt} &= (-B_1 v_x - B_2 v_y)_t \\ &= -B_1 v_{tx} - B_2 v_{ty} \\ &= -B_1(-B_1 v_x - B_2 v_y)_x - B_2(-B_1 v_x - B_2 v_y)_y \\ &= B_1^2 v_{xx} + (B_1 B_2 v + B_2 B_1)v_{yx} + B_2^2 v_{yy}, \end{aligned}$$

gives the Taylor series

$$v - \Delta t(B_1 v_x + B_2 v_y) + \frac{\Delta t^2}{2}(B_1^2 v_{xx} + (B_1 B_2 v + B_2 B_1)v_{yx} + B_2^2 v_{yy}) + O(\Delta t^3)$$
$$= \sum_{i,j=-1}^{1} C_{ij} \left(v + i\Delta x\ v_x + j\Delta y\ v_y + \frac{(i\Delta x)^2}{2})v_{xx} + \ ij\Delta x\Delta y\ v_{xy} + \frac{(j\Delta y)^2}{2} v_{yy} \right) + O(\Delta t^3)$$

Equating the coefficients results in the $O(\Delta t^2)$-consistency condition:

$$\sum_{i,j=-1}^{1} C_{ij} = 1$$
$$\sum_{i,j=-1}^{1} iC_{ij} = -B_1 R$$
$$\sum_{i,j=-1}^{1} jC_{ij} = -B_2$$
$$\sum_{i,j=-1}^{1} i^2 C_{ij} = B_1 R^2$$
$$\sum_{i,j=-1}^{1} ijC_{ij} = \frac{B_1 B_2 + B_2 B_1}{2} R^2$$
$$\sum_{i,j=-1}^{1} j^2 C_{ij} = B_2 R^2.$$

6.2 Stability

A numerical scheme is stable if the results remain bounded in the limit $\Delta t \to 0$. A method for determining a necessary, but not sufficient, condition for stability is the discrete Fourier transform.
The discrete Fourier transform of the grid function $u_{k,m}^n$ is given by

$$u_{k,m}^n = \sum_{r=-L}^{L} \sum_{s=-L}^{L} \overline{u}_{k,m}^n e^{r\ i\frac{\pi}{L}x_k} e^{s\ i\frac{\pi}{L}y_m},$$

with

$$x_k = k\Delta x, \quad y_m = m\Delta x.$$

Each term $u_{k,m}^n$ in the numerical scheme is replaced by the corresponding Fourier series, and we obtain

$$\sum_{r=-L}^{L} \sum_{s=-L}^{L} \overline{u}_{k,m}^{n+1} e^{r\ i\frac{\pi}{L}x_k} e^{s\ i\frac{\pi}{L}y_m} = \sum_{r=-L}^{L} \sum_{s=-L}^{L} A_{r,s} \overline{u}_{k,m}^n e^{r\ i\frac{\pi}{L}x_k} e^{s\ i\frac{\pi}{L}y_m},$$

where $A = (A_{r,s})$ is a matrix which depends on the numerical scheme.
Factoring all terms containing

$$e^{r\ i\frac{\pi}{L}x_k} e^{s\ i\frac{\pi}{L}y_m}$$

gives an equation

$$\overline{u}_{k,m}^{n+1} = G_{k,m}\overline{u}_{k,m}^{n}.$$

This $G_{k,m}$ is a scalar if the equation (17) is scalar, and a matrix if it is not.
If it is a scalar, the stability condition is

$$|G_{k,m}| \leq 1.$$

If it is a matrix, the stability condition requires that the spectral radius satisfies

$$\rho(G_{k,m}) \leq 1$$

where the spectral radius is the largest eigenvalue of $G_{k,m}$ in absolute value.
If this condition is satisfied for the following numerical schemes, we will consider it as stable, because if a scheme is stable in this sense it will be stable for our test cases.

6.3 Convergence

A numerical scheme is convergent if, when taking the limits $\Delta t \to 0, \Delta x \to 0$, the numerical solution converges to the exact solution.
The Lax equivalence theorem states that if a numerical scheme for a linear partial differential equations is consistent and stable, then it converges to the analytical solution.
As (18) is a linear partial differential equation, the requirements for the following numerical scheme are consistency and stability.

6.4 Heuristic analysis

Besides looking at the theoretical convergence, we will also look at three test problems.
The first of these problems is the two dimensional linearised Euler equations with a local pressure rise, as shown in the following picture:

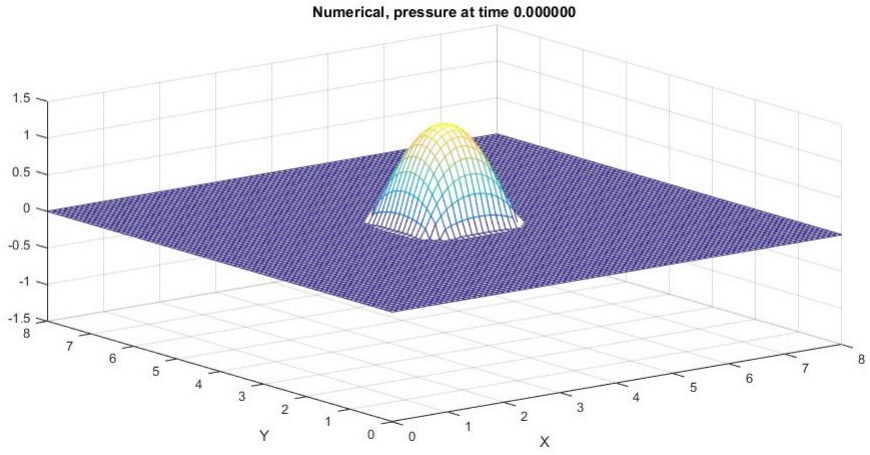


Figure 8: Time $t = 0$ s

The initial conditions are given by:

$$\begin{cases} u(x,y,0) = 0 \\ v(x,y,0) = 0 \\ p(x,0) = \begin{cases} 10 \cdot e^{-\frac{1}{1-(x-4)^2}} \cdot e^{-\frac{1}{1-(y-4)^2}} & (x-4)^2 + (y-4)^2 < 1 \\ 0 & \text{otherwise} \end{cases} \end{cases}$$

and the computational domain is given by

$$U = [0,8] \times [0,8] \times [0,T],$$

$dX = 0.1$ m, and reflecting boundaries, enforced by ghost cells.
A numerical scheme solves this problem if the numerical solution at times $t = 0.001$ s, $t = 0.005$ s and $t = 0.009$ s does not contain any spurious oscillations, which is a sign of a non-stable scheme, and the graph of the pressure and the graph of the velocity vectors are symmetrical about the point $(4, 4)$.

The second problem concerns the two dimensional wave equation

$$p_{tt} = c^2(p_{xx} + p_{yy}).$$

This equation can be obtained from the linearised Euler equations (1) by eliminating the velocities u and v.
The computational domain is taken to be

$$U = [0,1] \times [0,1] \times [0,T],$$

The initial condition is

$$p(x, y, 0) = \sin(\pi x) \sin(\pi y),$$

and the boundary conditions are take nas

$$p(x, 0, 0) = p(x, 1, 0) = p(0, y, 0) = p(1, y, 0) = 0,$$

where $x \in [0, 1]$ and $y \in [0, 1]$.
The corresponding solution for p is given by

$$p(x, y, t) = \sin(\pi x) \sin(\pi y) \cos(\sqrt{2}\pi c t)$$

From this solution and Eq. (1) the solutions for u and v can be found:

$$u(x, y, t) = -\frac{1}{\sqrt{2}c\ \rho} \cos(\pi\ x) \sin(\pi\ y) \sin(\sqrt{2}\pi c t)$$
$$v(x, y, t) = -\frac{1}{\sqrt{2}c\ \rho} \sin(\pi\ x) \cos(\pi\ y) \sin(\sqrt{2}\pi c t).$$

This problem will be used to obtain the so-called experimental order of convergence (EOC) of the method. The error in the discrete 2-norm

$$\left(\sum_{k,l=1}^{N} |v_{k,l}^n - u_{k,l}^n|^2 \right)^{\frac{1}{2}}$$

is taken for different values of dX, and a linear relation is obtained between the values of the logarithm of dX and the logarithm of the error values of the pressure by using linear least squares to fit this line.This and will be compared with the theoretical order of convergence.
The different values for dX which we take will be $0.1, 0.05, 0.04, 0.02, 0.01$, and the errors will be computed at the time $t = 0.005$.

Finally, for the third test problem we will consider is the two dimensional water hammer problem, with the pipe laying in the direction of the y-axis, with a length of 20 meters, and a diameter of 1 meter, as shown in the following figure.

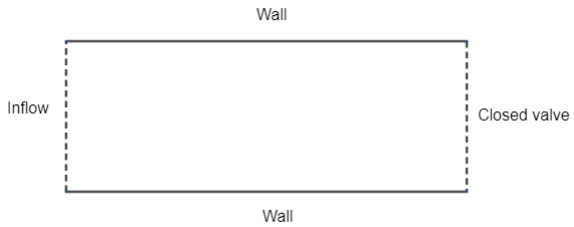


Figure 9: Two-dimensional water hammer setup

The initial conditions are taken to be a constant pressure, $p = 3 \cdot 10^5$ Pa, and a constant velocity in the direction of the closed valve, $u = 0.1$ m/s.
Along the wall and closed valve boundaries we impose a reflective boundary, and at the reservoir we impose a non-reflective boundary with prescribed constant pressure.
The average pressure and velocity in the cross-section of the pipe are taken, and compared to the results of the one dimensional case at the times $t = 0.01$ and $t = 0.03$.

7 Characteristics-like method

In this section we will use the method outlined by Wylie in [4], for the linearised two-dimensional Euler equations. This method is not an example of the bicharacteristic method, but it is a good introductory example to the thought behind it. The reasoning is that a 2D hyperbolic system of partial differential equations can be transformed into a characteristic form by adding to both sides of the equal sign a certain term for which one side of the equation can be written in a total derivative form. This side of the equation can then be solved exactly, while the other side of the equation must be approximated, usually by a numerical integration method.

7.1 Cartesian coordinates in 2D

We consider the linearised weakly compressible flow equations:

$$L_0 = p_t + \rho c^2 u_x + \rho c^2 v_y = 0 \tag{19}$$
$$L_1 = p_x + \rho u_t = 0 \tag{20}$$
$$L_2 = p_y + \rho v_t = 0, \tag{21}$$

already given in Chapter 2.
We complement these equations with the following identity:

$$L_3 = u_y + v_x - (u_y + v_x) = 0. \tag{22}$$

We take a linear combination of these equations to obtain:

$$\begin{aligned} &L_0 + \lambda_1 L_1 + \lambda_2 L_2 + \lambda_3 L_3 \\ &= (\lambda_1 p_x + \lambda_2 p_y + \lambda p_t) + \rho\lambda_1 \left(\frac{c^2}{\lambda_1} u_x + \frac{\lambda_3}{\rho\lambda_1} u_y + u_t \right) \\ &+ \rho\lambda_2 \left(\frac{\lambda_3}{\rho\lambda_2} v_x + \frac{c^2}{\lambda_2} v_y + v_t \right) - \lambda_3 (u_y + v_x) = 0. \end{aligned}$$

This can be written in the total derivative form if the following conditions hold:

$$\frac{dx}{dt} = \lambda_1 = \frac{c^2}{\lambda_1} = \frac{\lambda_3}{\rho\lambda_2},$$
$$\frac{dy}{dt} = \lambda_2 = \frac{\lambda_3}{\rho\lambda_1} = \frac{c^2}{\lambda_2}.$$

The solutions are given by $\lambda_1 = \lambda_2 = \pm c$, $\lambda_3 = \pm\rho c^2$.
This gives four equations along a 'characteristic path', which we will separately denote:

A-equation:

$$\frac{dx}{dt} = c, \quad \frac{dy}{dt} = c$$
$$\frac{dp}{dt} + \rho c \frac{du}{dt} + \rho c \frac{dv}{dt} - \rho c^2 (u_y + v_x) = 0.$$

B-equation:

$$\frac{dx}{dt} = c, \quad \frac{dy}{dt} = -c$$
$$\frac{dp}{dt} + \rho c \frac{du}{dt} - \rho c \frac{dv}{dt} + \rho c^2 (u_y + v_x) = 0.$$

C-equation:

$$\frac{dx}{dt} = -c, \quad \frac{dy}{dt} = -c$$
$$\frac{dp}{dt} - \rho c \frac{du}{dt} - \rho c \frac{dv}{dt} - \rho c^2 (u_y + v_x) = 0.$$

D-equation:

$$\frac{dx}{dt} = -c, \quad \frac{dy}{dt} = c$$
$$\frac{dp}{dt} - \rho c \frac{du}{dt} + \rho c \frac{dv}{dt} + \rho c^2 (u_y + v_x) = 0.$$

We consider a computational grid $V = (x_k, y_l, t_n)$, with $x_k - x_{k-1} = y_l - y_{l-1} = \Delta x$ and $t_n - t_{n-1} = \Delta t = \frac{1}{c}\Delta x$. The name for these equations is clarified by the following picture:

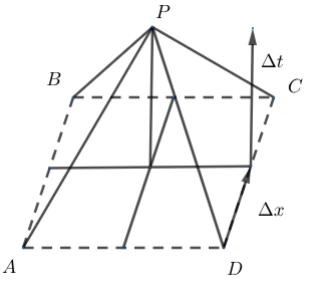


Figure 10: Characteristic lines

Point P at time t_{n+1} can be connected to point A at time t_n along the A-characteristic path, along which the A-equation holds. This holds similarly for the B,C and D-equations.

It now comes apparent why the identity (22) is chosen like that. It guarantees that the "characteristic curves" for this problem will go through the computational points, and all the equations are coupled together.

Before discussing the numerical scheme, we introduce one more variable, namely $q = \frac{u_y + v_x}{2}$, so that there are four equations for the four variables u, v, p, q.

Integrating the A,B,C,D-equations along there respective characteristic paths from t_n to t_{n+1} gives the following equations:

$$\frac{p_P}{\rho c} + u_P + v_P - q_P \Delta x = e_1, \tag{23}$$
$$\frac{p_P}{\rho c} + u_P - v_P + q_P \Delta x = e_2, \tag{24}$$
$$\frac{p_P}{\rho c} - u_P - v_P - q_P \Delta x = e_3, \tag{25}$$
$$\frac{p_P}{\rho c} - u_P + v_P + q_P \Delta x = e_4, \tag{26}$$

where

$$\begin{aligned} e_1 &= \frac{p_A}{\rho c} + u_A + v_A + q_A \Delta x, \\ e_2 &= \frac{p_B}{\rho c} + u_B - v_B - q_B \Delta x, \\ e_3 &= \frac{p_C}{\rho c} - u_C - v_C + q_C \Delta x, \\ e_4 &= \frac{p_D}{\rho c} - u_D + v_D - q_D \Delta x. \end{aligned}$$

The integration in these equations has been exact, except for the last term $\rho c^2(u_y + v_x)$, where the trapezoidal rule has been used:

$$\int_{t_n}^{t_{n+1}} c(u_y + v_x)dt = \int_{t_n}^{t_{n+1}} 2c\ q\ dt \approx c\Delta t \left(q(t_{n+1}) + q(t_n)\right) = \Delta x \left(q(t_{n+1}) + q(t_n)\right).$$

Solving (23) for p_P, u_P, v_P, q_P gives

$$\begin{aligned} p_P &= \frac{\rho c(e_1 + e_2 + e_3 + e_4)}{4} \\ u_P &= \frac{e_1 + e_2 - e_3 - e_4}{4} \\ v_P &= \frac{e_1 - e_2 - e_3 + e_4}{4} \\ q_P &= \frac{-e_1 + e_2 - e_3 + e_4}{4\Delta x}. \end{aligned}$$

7.2 Problem 1

We will consider first the local pressure rise problem defined in Section 6.4. With the requirement that $dX = 0.1$ m, dT is given by

$$dT = \frac{dX}{c} = \frac{0.1}{1000} = 1 \cdot 10^{-4} \text{ s}.$$

The graphs of the density and the velocity vectors at times $t = 0.001$ s, $t = 0.005$ s and $t = 0.009$ s are given in the following figures:

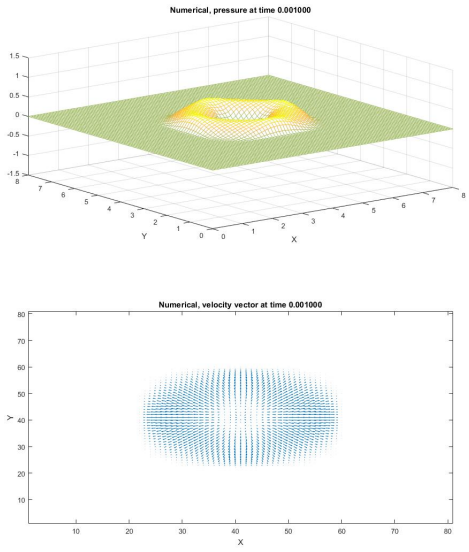


Figure 11: Problem 1: $t = 0.001$ s

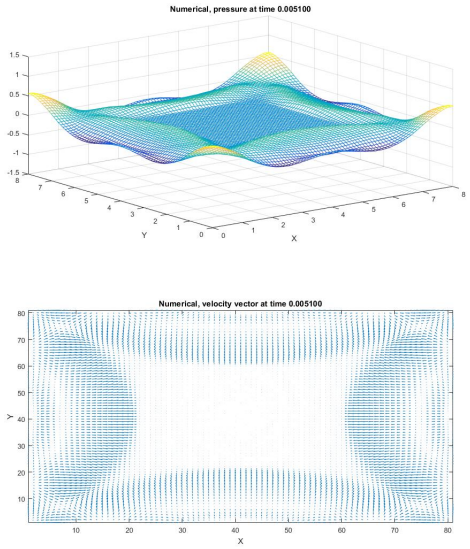


Figure 12: Problem 1: $t = 0.005$ s

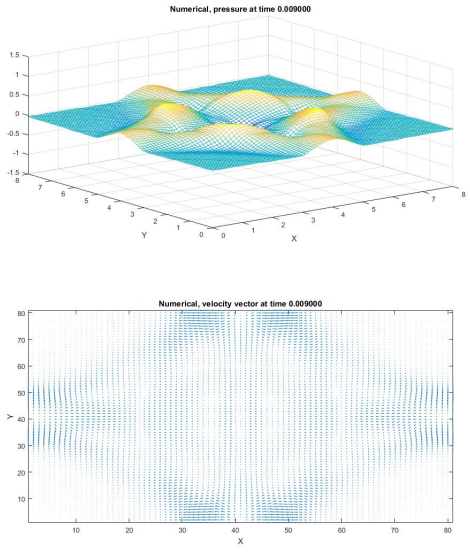


Figure 13: Problem 1: $t = 0.009$ s

We see that the density plots are nice and smooth, and do not display any spurious oscillations. Furthermore, the velocity vector plots are symmetrical, and do not display any strange behaviour.

7.3 Problem 2

For the second problem defined in Section 6.4, the value of dT stays the same, and we obtain the following error values:

dX	0.1	0.05	0.04	0.02	0.01
Absolute error	$1.52 \cdot 10^{-2}$	$3.86 \cdot 10^{-3}$	$2.31 \cdot 10^{-3}$	$5.82 \cdot 10^{-4}$	$1.49 \cdot 10^{-4}$

The log values of dX, or h in the following graph, and the error values are given in the following figure.

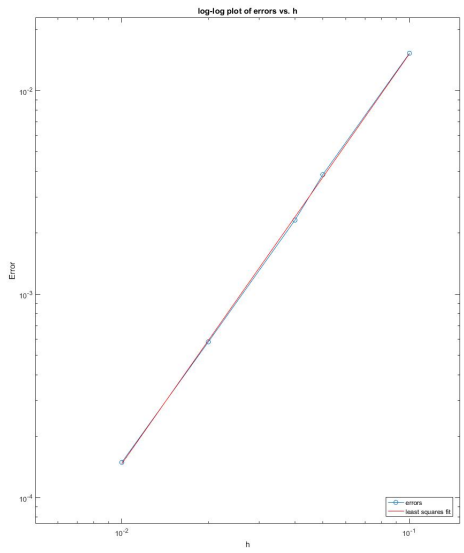


Figure 14: dX versus errors.

The least-squares line is given by the equation

$$E(h) = 1.57155 \cdot h^{2.01546},$$

so the estimated order of convergence is 2.0, which corresponds to the fact that the only approximations made in the numerical scheme is a second order numerical integration method, which therefore gives us a theoretical order of convergence of two.

7.4 Problem 3

The results of the two-dimensional water hammer simulation at a time around $t = 0.01$ s is given in the following figure:

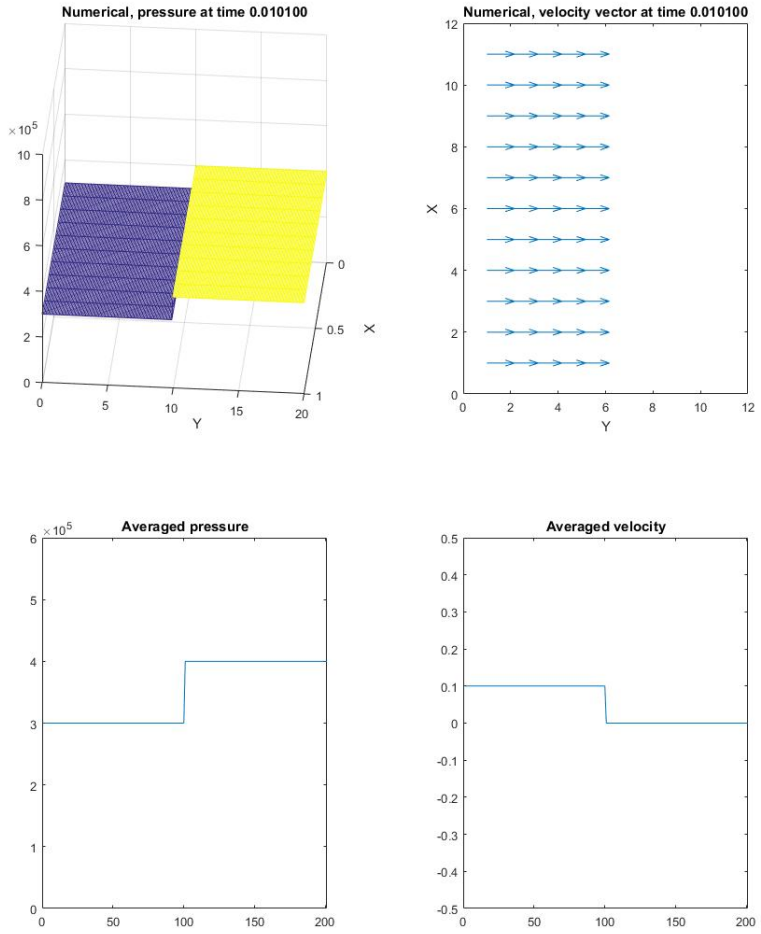


Figure 15: Results at $t \approx 0.01$ s

In the upper left graph, the pressure is plotted, in the upper right the velocity vectors are graphed. In the bottom two graphs the pressure and the velocity along the y-axis are given.
The results obtained are the same as for the one-dimensional case, and espe-

cially the discontinuity is very sharp.

The results at $t \approx 0.03$ s are shown in the next figure:

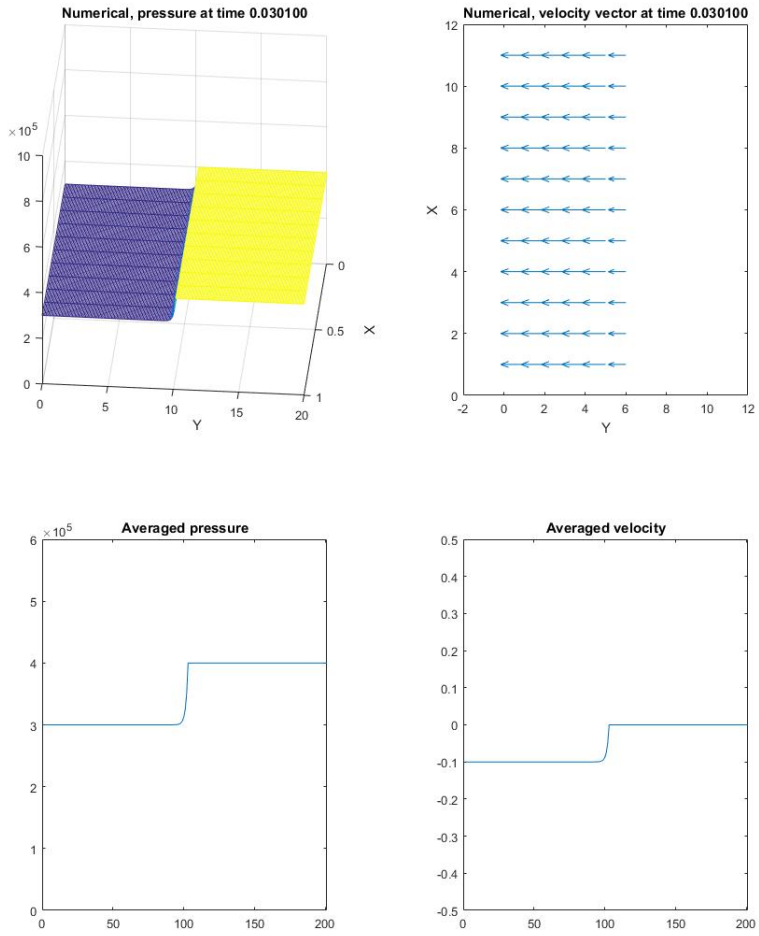


Figure 16: Results at $t \approx 0.03$ s

The results again are very similar to the one dimensional case, but the discontinuity is not that sharp any more. This is most likely due to a flaw in the ghost cells at the inflow boundary condition.

7.5 Conclusion

Overall, the characteristics-like method is easy to implement, and gives very good results on both smooth and non-smooth solutions for the linearised Euler equations.
The downsides of this method are the introduction of the function $q = u_y + v_x$ and the restriction to rectangular grids. This q cannot be obviously related to a physical quantity, and this method does not easily generalize to other problems because of its specific form.

Part III
Method of bicharacteristics

8 Derivation

In this section the method of bicharacteristics is given, which is a generalization of the method of characteristics for hyperbolic multi-dimensional partial differential equations.
This generalization will be based on the the essay of Delaney [5], and similar notation will be used as in that essay.
This method is, like the regular method of characteristics, based on taking a linear combination of the partial differential equations. In the scalar case, the method of characteristics transforms the partial differential equations into ordinary differential equations along characteristic curves. For a system of one-dimensional hyperbolic PDE's the method of multipliers transforms the system in a set of ordinary differential equations along a number of characteristic curves. The method of bicharacteristics transforms an $(n+1)$-dimensional system of hyperbolic partial differential equations into a set of compatibility equations which holds along a surface in the $(n+1)$-dimensional space. These compatibility equations are comparable to the ODE obtained with the method of characteristics, however they are still partial differential equations in n dimensions. This simplification of the set of equations can be used to make a numerical scheme to compute the numerical solution.

In this section the Einstein summation convention will be used, that is, whenever an index appears twice in a single term, it implies summation. For example:

$$c_i x_i = \sum_{i=1}^{n} c_i x_i.$$

For the method of bicharacteristics we consider the following system of quasi-linear hyperbolic partial differential equations:

$$a_{\mu\nu i} \frac{\partial u_\nu}{\partial x_i} = b_\mu. \tag{27}$$

Here i runs over 1 to 3, and μ and ν runs over 1 to n, and $a_{\mu\nu i}$ and b_μ are functions of u_ν and x_i.
We consider a linear combination of (27):

$$w_\mu a_{\mu\nu i} \frac{\partial u_\nu}{\partial x_i} = w_\mu b_\mu, \tag{28}$$

where w_μ can be seen as the left eigenvector of $a_{\mu\nu i}$.

The elements w_μ have to be chosen such that the vectors $w_{\nu i}$ are linearly dependent. This is equivalent to the vectors $w_{\nu i}$ lying in a plane. This plane containing the $w_{\nu i}$ is known as the characteristic plane, and the normal N_i to $w_{\mu i}$ is called the characteristic normal.
A surface which is everywhere tangent to a characteristic plane is called a characteristic surface, and the line of intersection between a characteristic plane and a characteristic surface is called a bicharacteristic.

Let N_i be a characteristic normal to a characteristic plane spanned by $w_{\nu i}$. Then it satisfies

$$N_i w_{\nu i} = N_i w_\mu a_{\mu\nu i} = (a_{\mu\nu i} N_i) w_\mu = 0.$$

for $\nu = 1, 2, \ldots, n$.
As we are interested in non-trivial solutions w_μ, we require that the determinant of the matrix $a_{\mu\nu i} N_i$ vanishes:

$$\det(a_{\mu\nu i} N_i) = 0.$$

This gives a polynomial equation in the variables N_i, and we assume that it is of the form

$$(U_k N_k)^{n-2} A_{ij} N_i N_j = 0,$$

where n is the dimension of (27).
This gives rise to two different characteristic surfaces, namely one specified by

$$U_k N_k = 0,$$

and another specified by

$$A_i j N_i N_j = 0.$$

The first one corresponds to the particle path, the so called characteristic flow surface, and the second one corresponds to the characteristic cone, the characteristic wave surface.

8.1 Characteristic flow surface

The normals N_i to the characteristic flow surface satisfy the equation

$$U_i N_i = 0,$$

where we will assume that $U_3 = 1$.
Along with this equation we require that the normals are of unit length, so $N_i N_i = 1$. So two equations are given for the three unknowns N_i, such that there are an infinite number of normals, which corresponds to the fact that the

characteristic flow envelope is a line.
Let $dS_i = (dS_1, dS_2, dS_3)$ be a differential element of this line, then it satisfies

$$dS_i N_i = (dS_1 N_1 + dS_2 N_2) + dS_3 N_3 = 0.$$

Dividing this by S_3 and comparing it with $U_i N_i = 0$ gives the equations

$$\frac{dS_1}{dS_3} = U_1$$
$$\frac{dS_2}{dS_3} = U_2.$$

Furthermore, if $U_2 \neq 0$, then we also have

$$\frac{dS_1}{dS_2} = \frac{U_1}{U_2}.$$

Thus the particle path is the envelope of the characteristic flow surfaces, and the bicharacteristic is given by this path.

8.2 Characteristic wave surface

The normals N_i to the characteristic wave surface satisfy the quadric cone equation

$$A_{ij} N_i N_j = 0, \tag{29}$$

and will be called the normal cone.
At each point there are infinite characteristic wave surfaces with normals satisfying (29). The envelope of these characteristic surfaces is called the characteristic conoid.
The reciprocal cone of the cone of normals is called the characteristic cone, which is locally tangent to the characteristic conoid at the vertex of the cone. A differential element of this conoid is given by

$$A^{-1}_{ij} dx_i dx_j = 0. \tag{30}$$

The following parametrisation is introduced for this differential element:

$$dx_i = (\lambda_i + \mu_i \cos\theta + \nu_i \sin\theta)\, d\tau, \quad i = 1, 2, 3. \tag{31}$$

Here θ is to be taken constant, and corresponds to one specific bicharacteristic.

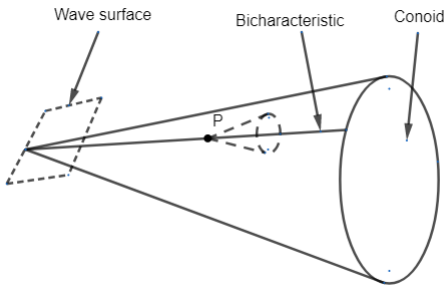


Figure 17: Characteristic cone

The vectors $\{\lambda_i, \mu_i, \nu_i\}$ must satisfy (30), thus

$$A_{ij}^{-1}(\lambda_i\lambda_j + \mu_i\mu_j\cos^2\theta + \nu_i\nu_j\sin^2\theta + \\ 2\lambda_i\mu_j\cos\theta + 2\lambda_i\mu_j\sin\theta + 2\mu_i\nu_j\cos\theta\sin\theta) = 0.$$

This is satisfied if λ_i, μ_i and ν_i are chosen such that

$$-A_{ij}^{-1}\lambda_i\lambda_j = A_{ij}^{-1}\mu_i\mu_j = A_{ij}^{-1}\nu_i\nu_j \tag{32}$$

and

$$A_{ij}^{-1}\lambda_i\mu_j = A_{ij}^{-1}\lambda_i\nu_j = A_{ij}^{-1}\mu_i\nu_j = 0. \tag{33}$$

8.2.1 Bicharacteristic tangency condition

Besides requiring that (31) satisfies (32) and (33), we also require that the integral

$$x_i - x_i^0 = \int_0^\tau (\lambda_i + \mu_i\cos\theta + \nu_i\sin\theta)\, d\tau, \quad i = 1, 2, 3$$

defines a bicharacteristic.
This integral relation is called the bicharacteristic tangency condition.
The differential element dx_i is a function of both θ and τ, so

$$dx_i = \frac{\partial x_i}{\partial\theta}d\theta + \frac{\partial x_i}{\partial\tau}d\tau.$$

The second term can be obtained by differentiating the integral with respect to τ:

$$\frac{\partial x_i}{\partial\tau} = \lambda_i + \mu_i\cos\theta + \nu_i\sin\theta. \tag{34}$$

This corresponds to a vector tangent to the bicharacteristic corresponding to θ.
Consider the point P in Figure 17, and the corresponding differential conoid,

denoted by a dotted line. The surface element tangent to the conoid in the direction $\lambda_i + \mu_i \cos\theta + \nu_i \sin\theta$ is given by

$$A_{ij}^{-1}(\lambda_j + \mu_j \cos\theta + \nu_j \sin\theta)dx_i = 0.$$

This surface is also tangent to the whole differential conoid, and therefore we have

$$\begin{aligned} &A_{ij}^{-1}(\lambda_j + \mu_j \cos\theta + \nu_j \sin\theta)\frac{\partial x_i}{\partial\theta}d\theta \\ +\ &A_{ij}^{-1}(\lambda_j + \mu_j \cos\theta + \nu_j \sin\theta)\frac{\partial x_i}{\partial\tau}d\tau = 0. \end{aligned}$$

Substituting (34) and using (32) and (33) gives the so called bicharacteristic tangency condition

$$A_{ij}^{-1}(\lambda_j + \mu_j \cos\theta + \nu_j \sin\theta)\frac{\partial x_i}{\partial\theta} = 0.$$

This condition is used to determine the vectors μ_i and ν_i along the bicharacteristic given by a fixed reference at the vertex of the conoid.

8.2.2 The wave surface compatibility relation

The method of bicharacteristics transforms the original partial differential equation into a partial differential equation with one less derivative in a variable along a surface. Thus the original equation transforms into

$$E_\nu \frac{\partial u_\nu}{\partial x_i'} + F_\nu \frac{\partial u_\nu}{\partial x_2'} = D,$$

where E_ν and F_ν depend on the choice of directions x_1' and x_2'.
We will choose as directions the bicharacteristic

$$\lambda_i + \mu_i \cos\theta + \nu_i \sin\theta,$$

and

$$M_i = \nu_i \cos\theta - \mu_i \sin\theta.$$

The compatibility equation takes the form

$$A_\nu(\lambda_i + \mu_i \cos\theta + \nu_i \sin\theta)\frac{\partial u_\nu}{\partial x_i} \tag{35}$$

$$= B + C_\nu \left(\cos\theta\nu_i \frac{\partial u_\nu}{\partial x_i} - \sin\theta\mu_i \frac{\partial u_\nu}{\partial x_i}\right)\frac{\partial u_\nu}{\partial x_i}, \tag{36}$$

where

$$\begin{aligned} A_\nu &= A_{1\nu} + A_{2\nu}\cos\theta + A_{3\nu}\sin\theta, \\ B_\nu &= B_{1\nu} + B_{2\nu}\cos\theta + B_{3\nu}\sin\theta, \\ C_\nu &= C_{1\nu} + C_{2\nu}\cos\theta + C_{3\nu}\sin\theta. \end{aligned}$$

λ_i is chosen such that $C_{1\nu} = 0$, and the final form of the compatibility relation becomes

$$A_\nu d_L u_\nu = [B + C_\nu(\nu_i \cos\theta - \mu_i \sin\theta)\frac{\partial u_\nu}{\partial x_i}]d\tau.$$

This form enables us to eliminate the cross derivatives $\frac{\partial u_n}{\partial x_i}$ at the solution point, which will be further explained in the following subsection.

8.3 Numerical scheme

For discussion of the numerical scheme we will consider the case of three partial differential equations.

There are two choices for defining the numerical scheme. The first is to take a finite number of bicharacteristics, usually equal to 5. The second choice is to integrate along all the bicharacteristics.

8.3.1 Finite number of bicharacteristics

We will consider 5 bicharacteristics, namely one given by the characteristic flow surface and 4 given by the characteristic wave surface, clarified by the following picture:

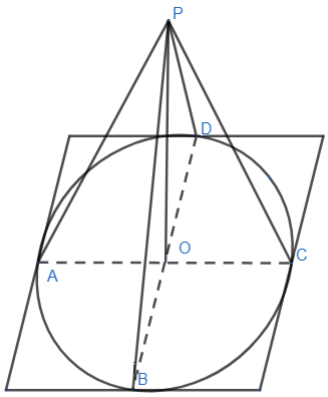


Figure 18: Bicharacteristic scheme

The computational point will be denoted by P, which lies on the plane $\{t = t_n + \Delta t\}$, and a point on the known surface $\{t = t_n\}$ will be denoted by Q.

The compatibility equation for the characteristic wave surface is given by

$$A_\nu du_\nu = \left[B + C_\nu \left(\cos\theta\nu_i \frac{\partial u_\nu}{\partial x_i} - \sin\theta\mu_i \frac{\partial u_\nu}{\partial x_i}\right)\right] d\tau. \tag{37}$$

This equation is integrated from t to $t + \Delta t$, giving

$$\int_t^{t+\Delta t} A_\nu \frac{du_\nu}{dt} dt = \int_t^{t+\Delta t} \left[B + C_\nu \left(\cos\theta \nu_i \frac{\partial u_\nu}{\partial x_i} - \sin\theta \mu_i \frac{\partial u_\nu}{\partial x_i} \right) \right] dt.$$

This can be written in a finite difference form, using the modified Euler scheme:

$$\overline{A}_\nu [u_\nu(P) - u_\nu(Q)] = \left(\overline{B} + \frac{1}{2}[S(P) + S(Q)] \right) \Delta t, \tag{38}$$

where

$$\begin{aligned} S &= C_\nu(\nu_i \cos\theta - \mu_i \sin\theta) \frac{\partial u_\nu}{\partial x_i} \\ \overline{A}_\nu &= \frac{1}{2}[A_\nu(P) + A_\nu(Q)] \\ \overline{B} &= \frac{1}{2}[B(P) + B(Q)]. \end{aligned}$$

Noting that

$$C_\nu = C_{2\nu} \cos\theta + C_{3\nu} \sin\theta,$$

we have that S is of the form

$$S = (C_{2\nu} \cos\theta + C_{3\nu} \sin\theta)(\nu_i \cos\theta - \mu_i \sin\theta) \frac{\partial u_\nu}{\partial x_i}.$$

We will take 4 bicharacteristics, each corresponding to one of the initial angles $\theta = 0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}$. This is chosen such that the compatibility conditions have a sufficiently nice form.
To each such angle there corresponds a point Q on $\{t = t_n\}$. This point Q can be computed from the bicharacteristic tangency condition.
In the original problem there are 3 unknowns, but in these 4 equations there are 5 unknowns, namely the original u_i, and the cross derivatives. Therefore, one more equation must be specified, and we will use the characteristic flow surface.

So the numerical scheme consists of 5 equations, solving 5 unknowns, and this will be explored further in the next chapter.

8.3.2 Integrating bicharacteristics

Another, more elegant way to numerically solve these equations is by using integration around the characteristic cone.
The compatibility relation (38) is multiplied by a suitable weighting factor $f_1(\theta)$, and then integrated from $\theta = 0$ to $\theta = 2\pi$, so

$$\int_0^{2\pi} f_1(\theta) \overline{A}_\nu [u_\nu(P) - u_\nu(Q)] d\theta = \Delta t \int_0^{2\pi} f_1(\theta) \left(\overline{B} + \frac{1}{2}[S(P) + S(Q)] \right) d\theta.$$

$f(\theta)$ is chosen such that a term $Cu_\nu(P)$ appears in the equation, where C is a constant, so

$$u_\nu(P)\int_0^{2\pi} f_1(\theta)\overline{A}_\nu d\theta = \int_0^{2\pi} f(\theta)u_\nu(Q)\overline{A}_\nu d\theta$$
$$+\ \Delta t\frac{1}{2}\int_0^{2\pi} f(\theta)S(P)d\theta + \Delta t\frac{1}{2}\int_0^{2\pi} f(\theta)S(Q))d\theta + \Delta t\int_0^{2\pi} f(\theta)\overline{B}d\theta.$$

Most of the terms can be calculated or approximated, except the term

$$\Delta t\frac{1}{2}\int_0^{2\pi} f(\theta)S(P)d\theta,$$

which depends on the cross derivatives at the solution point.
This problem can be solved by adding a suitable multiple of the finite-difference form of the compatibility relation for the characteristic flow surface, which eliminates these cross-derivative terms.
This numerical scheme will be investigated in Section 10: "Method of bicharacteristics, Butler's scheme".

9 Numerical scheme with a finite number of bicharacteristics

We consider the linearised slightly compressible flow equations

$$p_t + \rho c^2 u_x + \rho c^2 v_y = 0 \tag{39}$$

$$\frac{1}{\rho} p_x + u_t = 0 \tag{40}$$

$$\frac{1}{\rho} p_y + v_t = 0. \tag{41}$$

In Einstein notation this is written as

$$a_{\mu\nu i} \frac{\partial u_\nu}{\partial x_i} = 0, \tag{42}$$

where

$$\begin{cases} x_1 = x, & u_1 = u, \\ x_2 = y, & u_2 = v, \\ x_3 = t, & u_3 = p, \end{cases}$$

and

$$\begin{cases} a_{111} = \rho c^2, & a_{231} = \frac{1}{\rho}, \\ a_{122} = \rho c^2, & a_{323} = 1, \\ a_{133} = 1, & a_{332} = \frac{1}{\rho}, \\ a_{213} = 1. \end{cases}$$

and $a_{\mu\nu i} = 0$ otherwise.
The equations (45) can be written as

$$\begin{pmatrix} \rho c^2 \delta_{1i} & \rho c^2 \delta 2i & \delta_{3i} \\ \delta_{3i} & 0 & \frac{1}{\rho}\delta_{1i} \\ 0 & \delta_{3i} & \frac{1}{\rho}\delta_{2i} \end{pmatrix} \begin{pmatrix} \frac{\partial u}{\partial x_i} \\ \frac{\partial v}{\partial x_i} \\ \frac{\partial p}{\partial x_i} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}.$$

We consider a linear combination of these equations:

$$(\rho c^2 \delta_{1i} w_1 + \delta_{3i} w_2) \frac{\partial u}{\partial x_i} + (\rho c^2 \delta_{2i} w_1 + \delta_{3i} w_3) \frac{\partial v}{\partial x_i}$$
$$+ (\delta_{3i} w_1 + \frac{1}{\rho}\delta_{1i} w_2 + \frac{1}{\rho}\delta_{2i} w_3) \frac{\partial p}{\partial x_i} = 0.$$

The normals N_i to the characteristic plane w_i satisfy

$$\begin{cases} (\rho c^2 \delta_{1i} w_1 + \delta_{3i} w_2) N_i = 0 \\ (\rho c^2 \delta_{2i} w_1 + \delta_{3i} w_3) N_i = 0 \\ (\delta_{3i} w1 + \frac{1}{\rho}\delta_{1i} w_2 + \frac{1}{\rho}\delta_{2i} w_3) N_i = 0, \end{cases}$$

or in matrix form:

$$\begin{pmatrix} \rho c^2 N_1 & N_3 & 0 \\ \rho c^2 N_2 & 0 & N_3 \\ N_3 & \frac{1}{\rho} N_1 & \frac{1}{\rho} N_2 \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}. \tag{43}$$

The determinant of this matrix is given by

$$\rho^2 N_3 (N_3^2 - c^2(N_1^2 + N_2^2)) = 0.$$

The normal conditions for the characteristic flow surface is given by

$$\rho^2 N_3 = 0,$$

and for the characteristic wave surface it is given by

$$N_3^2 = c^2(N_1^2 + N_2^2).$$

9.1 Characteristic flow surface

Let the differential element for this curve be given by $dS_i = (dS_1, dS_2, dS_3)$. Take the inner product with the normal vector $N = (N_1, N_2, N_3)$ to get

$$dS_1 N_1 + dS_2 N_2 + dS_3 N_3 = 0$$

Dividing by dS_3 and setting N_3 equal to zero gives

$$N_1 \frac{dS_1}{dS_3} + N_2 \frac{dS_2}{dS_3} = 0.$$

As the normal condition for the characteristic flow surface holds for all possible combinations of N_1 and N_2 we see that

$$\frac{dS_1}{dS_3} = 0, \quad \frac{dS_2}{dS_3} = 0,$$

so we obtain the following differential element form for the particle path:

$$dS_1 = dS_2 = 0, \quad dS_3 = 1.$$

The bicharacteristic is given by the differential equation

$$dx = (dx_1, dx_2, dx_3) = (0, 0, 1).$$

For the compatability condition we consider the equation

$$p_t + \rho c^2 u_x + \rho c^2 v_y = 0.$$

Along this characteristic, the following differential equation holds:

$$dp = -\rho c^2 (u_x + v_y) d\tau.$$

9.2 Characteristic wave surface

The normal condition for the characteristic wave surface is given by

$$N_3^2 = c^2(N_1^2 + N_2^2),$$

or in matrix form

$$\begin{pmatrix} N_1 \\ N_2 \\ N_3 \end{pmatrix}^T \begin{pmatrix} c^2 & 0 & 0 \\ 0 & c^2 & 0 \\ 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} N_1 \\ N_2 \\ N_3 \end{pmatrix} = 0.$$

The equation for the differential conoid is given by the quadric equation with the inverse of this matrix, the differential element cone satisfies the equation

$$\begin{pmatrix} dx \\ dy \\ dt \end{pmatrix}^T \begin{pmatrix} \frac{1}{c^2} & 0 & 0 \\ 0 & \frac{1}{c^2} & 0 \\ 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} dx \\ dy \\ dt \end{pmatrix} = 0.$$

As the parametric representation of a differential element of the bicharacteristic curve we will take

$$\begin{aligned} dx &= c\cos\theta\alpha_1 + c\sin\theta\beta_1 \\ dy &= c\cos\theta\alpha_2 + c\sin\theta\beta_2 \\ dt &= 1. \end{aligned}$$

Here α_i and β_i are orthonormal vectors, so

$$\alpha_i\beta_i = 0, \quad \alpha_i\alpha_i = 1, \quad \beta_i\beta_i = 1.$$

The bicharacteristic tangency condition is given by

$$(\alpha_i\cos\theta + \beta_i\sin\theta)\frac{\partial x_i}{\partial\theta} = 0, \quad i = 1, 2.$$

The term $\frac{\partial x_i}{\partial\theta}$ is approximated to first-order accuracy, after which suitable approximations for α_i and β_i are determined.
The integral relation for the bicharacteristic is given by

$$x_i(\theta, t) - x_i^0 = \int_0^\tau (c\cos\theta\alpha_i + c\sin\theta\beta_i)d\tau, \quad i = 1, 2.$$

Here x_i^0 is the vertex of the conoid, and $\tau = 0$ corresponds to this vertex.
The integral can be approximated by the following "power series expansions":

$$\begin{aligned} \alpha_i &= \alpha_i^0 + O(t) \\ \beta_i &= \beta_i^0 + O(t). \end{aligned}$$

Here we have used the notation

$$\alpha_i(\theta) = \left.\frac{\partial \alpha_i}{\partial t}\right|_{t=0}.$$

Substituting this in the integral and integrating gives

$$x_i(\theta, t) - x_i^0 = c(\alpha_i^0 \cos\theta + \beta_i^0 \sin\theta)t + O(t^2).$$

Differentiating this with respect to θ gives

$$\frac{\partial x_i}{\partial \theta} = c(\beta_i^0 \cos\theta - \alpha_i^0 \sin\theta)t + O(t^2).$$

These equalities can be substituted in the bicharacteristic tangency condition to get

$$(\alpha_i \cos\theta + \beta \sin\theta)c(\beta_i^0 \cos\theta - \alpha_i^0 \sin\theta)t = 0.$$

Using the orthonormal properties of α_i and β_i gives an equation which is equal to zero for all orthonormal combinations. So for first-order accuracy no restrictions are imposed on α_i and β_i.
Therefore, we will take

$$\begin{cases} \alpha_1 = \cos\psi, & \alpha_2 = \sin\psi \\ \beta_1 = -\sin\psi, & \beta_2 = \cos\psi, \end{cases} \tag{44}$$

with $\psi = 0$.
The bicharacteristic curve representation is thus

$$\begin{aligned} dx &= c\cos\theta \\ dy &= c\sin\theta \\ dt &= 1. \end{aligned}$$

The corresponding normal N_i at the vertex of the conoid is then given by

$$\begin{aligned} N_1 &= \cos\theta \\ N_2 &= \sin\theta \\ N_3 &= -c. \end{aligned}$$

Substituting this in equation (43) gives

$$\begin{pmatrix} \rho c^2 \cos\theta & -c & 0 \\ \rho c^2 \sin\theta & 0 & -c \\ -c & \frac{1}{\rho}\cos\theta & \frac{1}{\rho}\sin\theta \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}.$$

Solving for w_i gives

$$\begin{aligned} w_1 &= k \\ w_2 &= k\ c\rho\cos\theta \\ w_3 &= k\ c\rho\sin\theta. \end{aligned}$$

Here k can be any number, and we will set it equal to 1.
With these the compatibility relation for the characteristic wave surface can be calculated, which is given by

$$\begin{aligned} & w_1(p_t + \rho c^2 u_x + \rho c^2 v_y) + w_2(\frac{1}{\rho}p_x + u_t) + w_3(\frac{1}{\rho}p_y + v_t) \\ &= p_t + \rho c^2 u_x + \rho c^2 v_y + c\rho\cos\theta(\frac{1}{\rho}p_x + u_t) + c\rho\sin\theta(\frac{1}{\rho}p_y + v_t) \\ &= 0. \end{aligned}$$

This can be written in the compatability relation form (37):

$$\begin{aligned} & p_t + c\cos\theta p_x + c\sin\theta p_y + c\rho\cos\theta(u_t + c\cos\theta u_x + c\sin\theta u_y) + \ c\rho\sin\theta(v_t + c\cos\theta v_x + c\sin\theta v_y) \\ &= -\rho c^2\sin^2\theta u_x + \rho c^2\cos\theta\sin\theta u_y + \rho c^2\cos\theta\sin\theta v_x - \rho c^2\cos^2\theta v_y. \end{aligned}$$

This equation can be written in the directional derivative direction $(1, c\cos\theta, c\sin\theta)$ as

$$dp + c\rho\cos\theta du + c\rho\sin\theta dv = (-\rho c^2\sin^2\theta u_x + \rho c^2\cos\theta\sin\theta u_y + \rho c^2\cos\theta\sin\theta v_x - \rho c^2\cos^2\theta v_y)d\tau.$$

9.3 Numerical scheme

Considering the cases of $\theta = 0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}$, we obtain 4 equations with the 5 unknowns u, v, p and the cross derivative terms u_x, v_y.

$$\begin{cases} dp + c\rho du = (-\rho c^2 v_y)d\tau \\ dp + c\rho dv = (-\rho c^2 u_x)d\tau \\ dp - c\rho du = (-\rho c^2 v_y)d\tau \\ dp - c\rho dv = (-\rho c^2 u_x)d\tau. \end{cases}$$

Along with the differential equation from the characteristic flow compatability relation

$$dp = -\rho c^2(u_x + v_y)d\tau$$

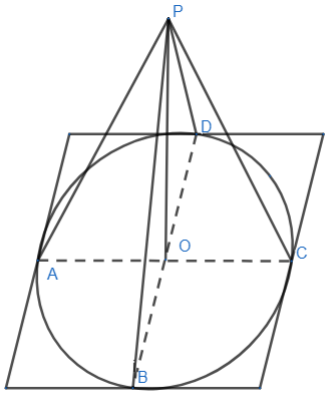
this gives a full system for the unknowns u, v, p and u_x, v_y.
The first 4 equations are respectively valid along the 4 curves

$$\begin{cases} (dx, dy, dt) = (1, 0, c) \\ (dx, dy, dt) = (0, 1, c) \\ (dx, dy, dt) = (-1, 0, c) \\ (dx, dy, dt) = (0, -1, c), \end{cases}$$

and the last equation is valid along the curve

$$(dx, dy, dt) = (0, 0, 1).$$

These equations are integrated from t_n to $t_{n+1} = t_n + \Delta t$ along their respective bicharacteristic, denoted in the next figure.



This gives the following set of equations:

$$\begin{cases} p(P) - p(A) + c\rho(u(P) - u(A)) = -\rho c^2 \frac{\Delta t}{2}(v_y(P) + v_y(A)) \\ p(P) - p(B) + c\rho(v(P) - v(B)) = -\rho c^2 \frac{\Delta t}{2}(u_x(P) + u_x(B)) \\ p(P) - p(C) - c\rho(u(P) - u(C)) = -\rho c^2 \frac{\Delta t}{2}(v_y(P) + v_y(C)) \\ p(P) - p(D) + c\rho(v(P) - v(D)) = -\rho c^2 \frac{\Delta t}{2}(u_x(P) + u_x(D)) \\ p(P) - p(O) = -\rho c^2 \frac{\Delta t}{2}(u_x(P) - u_x(O) + v_y(P) + v_x(O)) \end{cases}$$

First we will rewrite this as

$$\begin{cases} p(P) + c\rho u(P) + \rho c^2 \frac{\Delta t}{2} v_y(P) = e1 \\ p(P) + c\rho v(P) + \rho c^2 \frac{\Delta t}{2} u_x(P) = e2 \\ p(P) - c\rho u(P) + \rho c^2 \frac{\Delta t}{2} v_y(P) = e3 \\ p(P) - c\rho v(P) + \rho c^2 \frac{\Delta t}{2} u_x(P) = e4 \\ p(P) + \rho c^2 \frac{\Delta t}{2} u_x(P) + \rho c^2 \frac{\Delta t}{2} v_y(P) = e5. \end{cases}$$

with

$$\begin{cases} e1 = p(A) + c\rho u(A) - \rho c^2 \frac{\Delta t}{2} v_y(A) \\ e2 = p(B) + c\rho v(B) - \rho c^2 \frac{\Delta t}{2} u_x(B) \\ e3 = p(C) - c\rho u(C) - \rho c^2 \frac{\Delta t}{2} v_y(C) \\ e4 = p(D) - c\rho v(D) - \rho c^2 \frac{\Delta t}{2} u_x(D) \\ e5 = p(O) - \rho c^2 \frac{\Delta t}{2} u_x(O) - \rho c^2 \frac{\Delta t}{2} v_y(O) \end{cases}$$

Solving for the variables $p(P), u(P), v(P), u_x(P), v_y(P)$ gives

$$\begin{cases} p(P) = \frac{e_1+e_2+e_3+e_4-2e_5}{2} \\ u(P) = \frac{e_3-e_1}{2c\rho} \\ v(P) = \frac{e_4-e_2}{2c\rho} \\ u_x(P) = \frac{2e_5-e_1-e_3}{c^2\Delta t\rho} \\ v_y(P) = \frac{2e_5-e_2-e_4}{c^2\Delta t\rho}. \end{cases}$$

It seems that this is an elegant numerical scheme to solve the linearised Euler equations, and very similar to the characteristics-like method. However, this numerical scheme is unconditionally unstable, as will be shown in the following subsection.

9.4 Von Neumann analysis

For the von Neumann analysis we will consider the general bicharacteristic relations

$$\begin{aligned} &d_{L_A} p + \rho c \alpha_j d_{L_A} u_j + \rho c^2 \beta_i \beta_j \frac{\partial u_j}{\partial x_i} dt = 0 \\ &d_{L_B} p + \rho c \beta_j d_{L_B} u_j + \rho c^2 \alpha_i \alpha_j \frac{\partial u_j}{\partial x_i} dt = 0 \\ &d_{L_C} p - \rho c \alpha_j d_{L_C} u_j + \rho c^2 \beta_i \beta_j \frac{\partial u_j}{\partial x_i} dt = 0 \\ &d_{L_D} p - \rho c \beta_j d_{L_D} u_j + \rho c^2 \alpha_i \alpha_j \frac{\partial u_j}{\partial x_i} dt = 0 \\ &d_O p + \rho a^2 (\alpha_i \alpha_j + \beta_i \beta_j) \frac{\partial u_j}{\partial x_i} dt = 0. \end{aligned}$$

Note here that ψ in

$$\begin{cases} \alpha_1 = \cos\psi, & \alpha_2 = \sin\psi \\ \beta_1 = -\sin\psi, & \beta_2 = \cos\psi, \end{cases}$$

is left open.
We first eliminate the cross derivative terms to make the following equations

more readable:

$$
\begin{aligned}
&d_{L_A}p - d_{L_C}p + \rho c\alpha_j(d_{L_A}u_j + d_{L_C}u_j) = 0\\
&d_{L_B}p - d_{L_D}p + \rho c\beta_j(d_{L_B}u_j + d_{L_D}u_j) = 0\\
&d_{L_A}p + d_{L_B}p + d_{L_C}p + d_{L_D}p - 2d_Up\\
&+ \rho c\left(\alpha_j(d_{L_A}u_j - d_{L_C}u_j) + \beta_j(d_{L_B}u_j - d_{L_D}u_j\right) = 0
\end{aligned}
$$

Using the modified Euler scheme gives the following difference equations

$$
\begin{aligned}
&p(C) - p(A) + \rho c\alpha_j(2u_j(P) - u_j(A) - u_j(C)) = 0\\
&p(D) - p(B) + \rho c\beta_j(2u_j(P) - u_j(B) - u_j(D)) = 0\\
&2p(P) + 2p(O) - p(A) - p(B) - p(C) - p(D)\\
&+ \rho c\left(\alpha_j(u_j(C) - u_j(A)) + \beta_j(u_j(D) - u_j(B))\right) = 0.
\end{aligned}
$$

We use the von Neumann analysis to check if the method is stable. We only have to investigate one frequency index, and the corresponding term is given by

$$\overline{U} = e^{im\pi\frac{x}{L}}e^{in\pi\frac{y}{L}}\overline{T}(t),$$

where

$$\overline{U} = \begin{bmatrix} u \\ v \\ p \end{bmatrix}, \quad \overline{T}(t) = \begin{bmatrix} U(t) \\ V(t) \\ P(t) \end{bmatrix}.$$

We investigate the amplification factor gained from computing the time step from $t = 0$ to $t = \Delta t$. The values at the four points A, B, C and D have to be interpolated, and we will use a bi-parabolic interpolation polynomial of the form

$$h(x, y) = \sum_{i=0}^{2}\sum_{j=0}^{2} a_{ij}x^i y^j.$$

At time $t = 0$, the 9 grid variables are given

$$
\begin{aligned}
&\overline{U}(\frac{\Delta x}{L}, \frac{\Delta x}{L}, 0) = \zeta\eta\overline{T}(0), && \overline{U}(\frac{\Delta x}{L}, 0, 0) = \zeta\overline{T}(0),\\
&\overline{U}(\frac{\Delta x}{L}, -\frac{\Delta x}{L}, 0) = \zeta\eta^{-1}\overline{T}(0), && \overline{U}(0, \frac{\Delta x}{L}, 0) = \eta\overline{T}(0),\\
&\overline{U}(0, 0, 0) = \overline{T}(0), && \overline{U}(0, -\frac{\Delta x}{L}, 0) = \eta^{-1}\overline{T}(0),\\
&\overline{U}(-\frac{\Delta x}{L}, \frac{\Delta x}{L}, 0) = \zeta^{-1}\eta\overline{T}(0), && \overline{U}(-\frac{\Delta x}{L}, 0, 0) = \zeta^{-1}\overline{T}(0),\\
&\overline{U}(-\frac{\Delta x}{L}, -\frac{\Delta x}{L}, 0) = \zeta^{-1}\eta^{-1}\overline{T}(0),
\end{aligned}
$$

with

$$\zeta = e^{im\pi\frac{\Delta x}{L}}, \quad \eta = e^{in\pi\frac{\Delta x}{L}}.$$

With these points, the interpolation polynomial is calculated. The coefficients of this polynomial are omitted, as they are easily computed and not needed for the analysis, except the term a_{00}. The reason for this comes from the following two equations

$$\overline{U}(0,0,0) = a_{11}\overline{T}(0), \quad \overline{U}(0,0,0) = \overline{T}(0).$$

From this it follows that

$$\overline{U}(0,0,0) = a_{11}\overline{U}(0,0,0).$$

So if $a_{11} > 1$, the interpolating polynomial is not stable. Luckily, the term a_{11} is equal to 1, so the interpolation is stable.

We now return to the difference equations. We have the following 5 base points, given by the four surface wave bicharacteristic base points, and one flow surface base point:

Point A:

$$x_A = -\cos\psi \; C_r\Delta x, \quad y_A = \sin\psi \; C_r\Delta x.$$

Point B:

$$x_B = \sin\psi \; C_r\Delta x, \quad y_B = -\cos\psi \; C_r\Delta x.$$

Point C:

$$x_C = -\cos\psi \; C_r\Delta x, \quad y_C = \sin\psi \; C_r\Delta x.$$

Point D:

$$x_D = -\sin\psi \; C_r\Delta x, \quad y_D = \cos\psi \; C_r\Delta x.$$

Point O:

$$x_O = 0, \quad y_O = 0.$$

Here C_r is the Courant number given by

$$C_r = c\frac{\Delta t}{\Delta x}.$$

After substituting these coordinates and equations, and using the notation

$h(A) = h(x_A, y_A)$ and so on, we arrive at the following system:

$$\begin{aligned}
&\cos\psi\ U(\Delta t) + \sin\psi V(\Delta t) + \frac{1}{2\rho c}(h(C) - h(A))P(0) \\
&- \frac{1}{2}(h(A) + h(C))(\cos\psi U(0) + \sin\psi V(0)) = 0, \\
&- \sin\psi U(\Delta t) + \cos\psi V(\Delta t) + \frac{1}{2\rho c}(h(D) - h(B))P(0) \\
&- \frac{1}{2}(h(B) + h(D)(-\sin\psi U(0) + \cos\psi V(0))) = 0 \\
&P(\Delta t) + \left(h(O) - \frac{1}{2}(h(A) + h(B) + h(C) + h(D)\right) P(0) \\
&+ \frac{\rho c}{2}(\cos\psi(h(C) - h(A)) - \sin\psi(h(D) - h(B)))U(0) \\
&+ \frac{\rho c}{2}(\sin\psi(h(C) - h(A)) + \cos\psi(h(D) - h(B)))V(0) = 0.
\end{aligned}$$

As this is a rather large set of equations, we write it in matrix form:

$$A\ \overline{T}(\Delta t) = B\ \overline{T}(0),$$

where

$$A = \begin{pmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

$$B = \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix}$$

where b_{ij} is given by

$$
\begin{aligned}
b_{11} &= -\frac{1}{2}(h(A)+h(C))\cos\psi \\
b_{12} &= -\frac{1}{2}(h(A)+h(C))\sin\psi \\
b_{13} &= \frac{1}{2\rho c}(h(C)-h(A)) \\
b_{21} &= \frac{1}{2}(h(B)+h(D))\sin\psi \\
b_{22} &= -\frac{1}{2}(h(B)+h(D))\cos\psi \\
b_{23} &= \frac{1}{2\rho c}(h(D)-h(B)) \\
b_{31} &= \frac{\rho c}{2}(\cos\psi(h(C)-h(A)) - \sin\psi(h(D)-h(B))) \\
b_{32} &= \frac{\rho c}{2}(\sin\psi(h(C)-h(A)) + \cos\psi(h(D)-h(B))) \\
b_{33} &= -\frac{1}{2}(h(A)+h(B)+h(C)+h(D).
\end{aligned}
$$

The amplification factor is given by $A^{-1}B$, and the numerical scheme is stable if the spectral radius of this matrix is less than 1.
We will consider three cases, where the Courant number is 1, 0.8 and 0.05. Furthermore, we will set

$$dx = 0.05, \quad L = 3, \quad m = 2, \quad n = 1.$$

The spectral radius for each case is plotted against the value ψ, running from 0 to π, as α_i and β_i are periodic with period π.

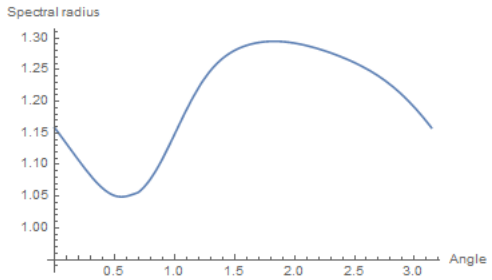


Figure 19: Courant number $C_r = 1$

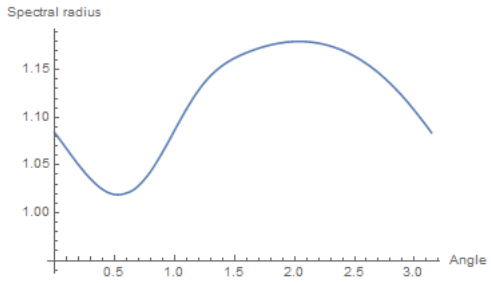


Figure 20: Courant number $C_r = 0.8$

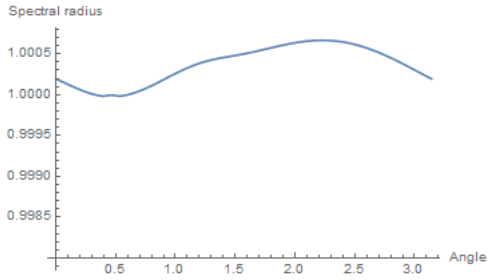


Figure 21: Courant number $C_r = 0.05$

From the spectral radius, we see that it is larger than 1 for most values of ψ, and that the numerical scheme is therefore unconditionally unstable. To check this, we have tried to solve Problem 1 with this scheme, and this gave the following numerical results:

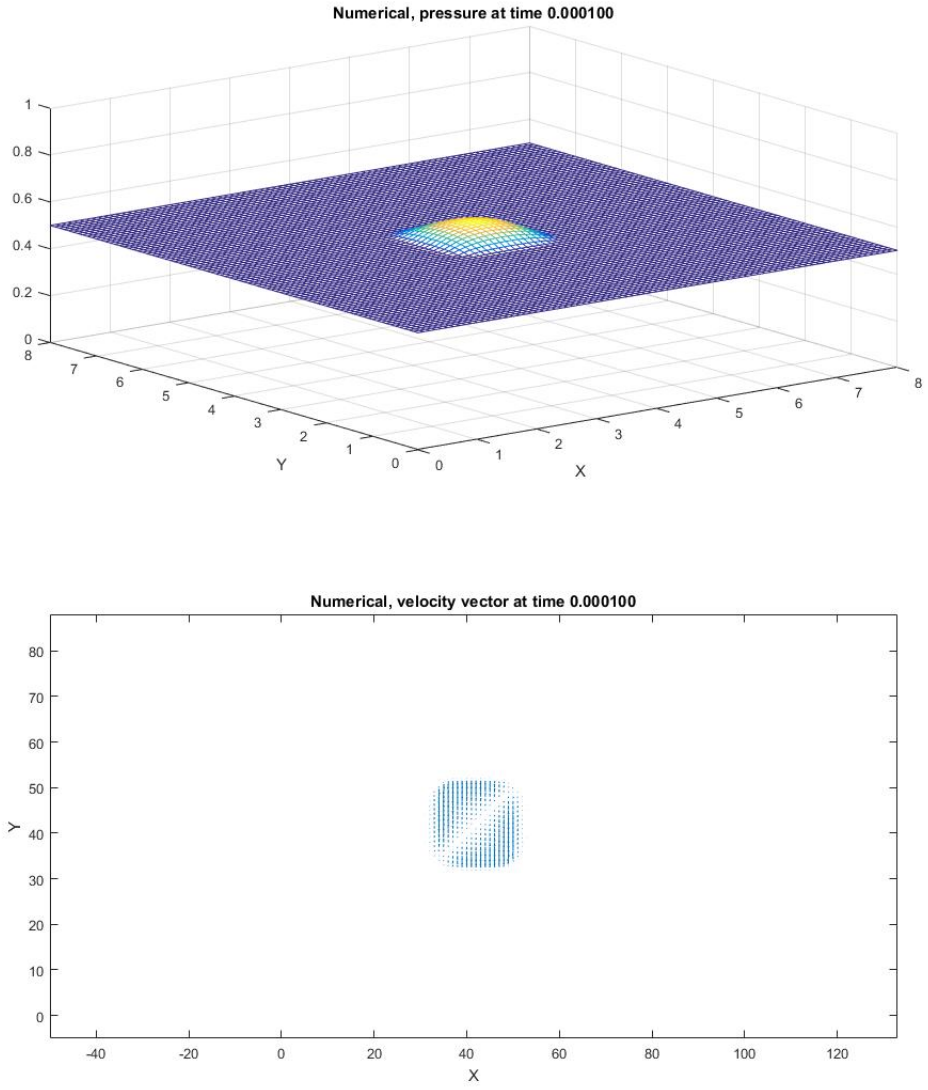


Figure 22: Time $t = 0.0001$ s

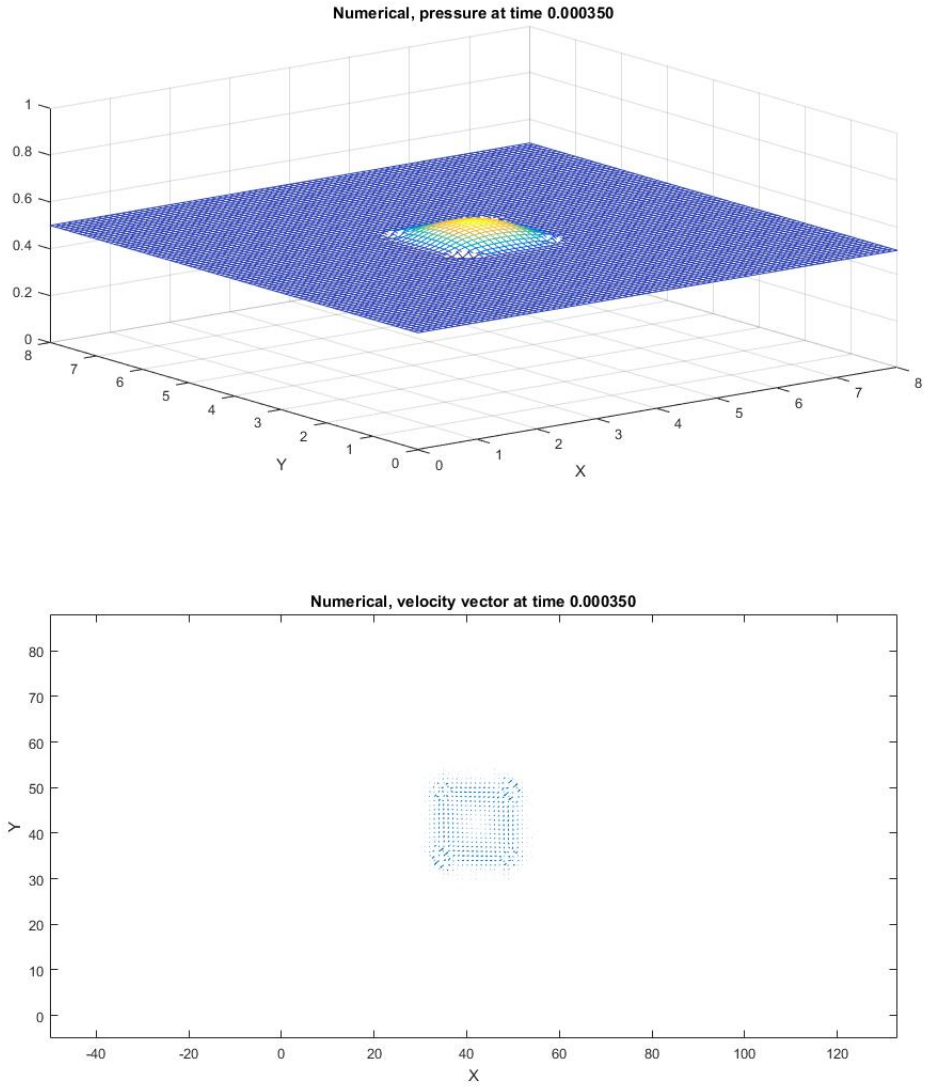


Figure 23: Time $t = 0.00035$ s

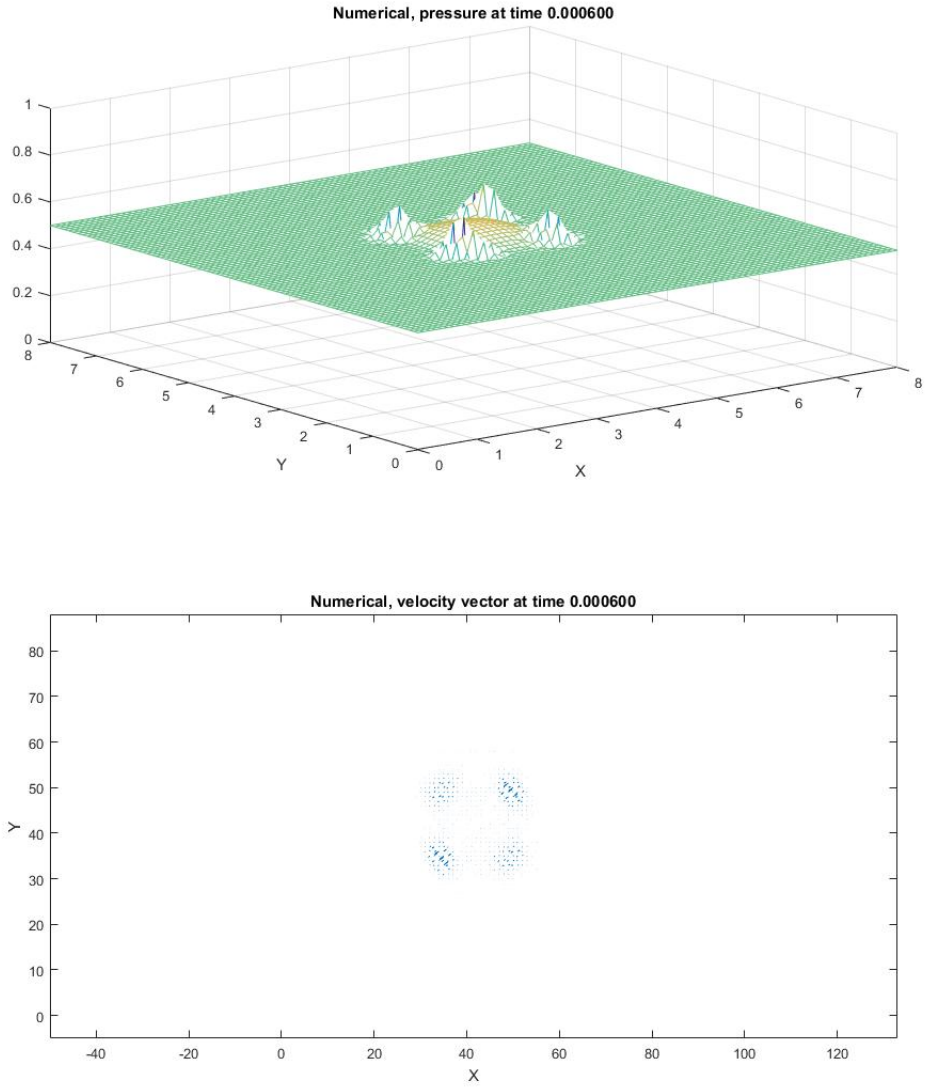


Figure 24: Time $t = 0.0006$ s

From these graphs we see that the numerical solution quickly develops oscillations, due to the instability of the scheme.

9.5 Conclusion

Due to the instability of this scheme, it is not usable for any numerical computation. The theoretical foundation of bicharacteristics however can be used to make stable numerical schemes, like Butler's scheme.

10 Butler's scheme

In this section Butler's scheme for the linearised Euler equations will be derived. This scheme was first introduced by Butler in 1960 [6]. However, his work is very abstract and was not of much use for this report. Therefore, the following section will be based on [7] and [8]. This last report introduces a finite volume scheme based on the same method as the one Butler uses, so it is still useful.

We again state the linearised Euler equations in Einstein notation:

$$a_{\mu\nu i}\frac{\partial u_\nu}{\partial x_i} = 0, \tag{45}$$

where

$$\begin{cases} x_1 = x, & u_1 = u, \\ x_2 = y, & u_2 = v, \\ x_3 = t, & u_3 = p, \end{cases}$$

and

$$\begin{cases} a_{111} = \rho c^2, & a_{231} = \frac{1}{\rho}, \\ a_{122} = \rho c^2, & a_{323} = 1, \\ a_{133} = 1, & a_{332} = \frac{1}{\rho}, \\ a_{213} = 1. \end{cases}$$

and $a_{\mu\nu i} = 0$ otherwise.
The compatability relation in general terms reads

$$A_\nu d_L u_\nu = [B + C_\nu(\nu_i \cos\theta - \mu_i \sin\theta)\frac{\partial u_\nu}{\partial x_i}]d\tau,$$

and for the equations under consideration is given by

$$dp + c\rho\cos\theta du + c\rho\sin\theta dv = (-\rho c^2 \sin^2\theta u_x + \rho c^2 \cos\theta\sin\theta u_y + \rho c^2\cos\theta\sin\theta v_x - \rho c^2\cos^2\theta v_y)d\tau.$$

After using the modified Euler scheme and integrating this equation with respect to θ from 0 to 2π we obtain:

$$u_\nu(P)\int_0^{2\pi} f_1(\theta)\overline{A}_\nu d\theta = \int_0^{2\pi} f(\theta)u_\nu(Q)\overline{A}_\nu d\theta \tag{46}$$

$$+\ \Delta t\frac{1}{2}\int_0^{2\pi} f(\theta)S(P)d\theta + \Delta t\frac{1}{2}\int_0^{2\pi} f(\theta)S(Q))d\theta + \Delta t\int_0^{2\pi} f(\theta)\overline{B}d\theta. \tag{47}$$

Here

$$S = -\rho c^2 \sin^2\theta u_x + \rho c^2\cos\theta\sin\theta u_y + \rho c^2\cos\theta\sin\theta v_x - \rho c^2\cos^2\theta v_y,$$

and the point P is the solution point at $(x, y, t + \Delta t)$ and Q is a point on the characteristic circle given by

$$Q = (c\Delta t \cos\theta, c\Delta t \sin\theta, t).$$

As the partial differential equation system has constant coefficients, the terms $\overline{A}_\nu$ and $\overline{B}$ are constant and given by

$$\overline{A}_\nu = \begin{cases} 1 & \nu = 1 \\ \rho c \cos\theta & \nu = 2 \\ \rho c \sin\theta & \nu = 3 \end{cases}$$
$$\overline{B} = 0.$$

As weighting factors we will take $f(\theta) = 1, \cos\theta$ and $\sin\theta$.
Equation (46) with weighting function $f(\theta) = 1$ becomes, after some trivial integrations:

$$2\pi\ p(P) = \int_0^{2\pi} p(Q) + c\rho\cos\theta\ u(Q) + c\rho\sin\theta\ v(Q)d\theta + \Delta t\pi(-\rho c^2 u_x(P) - \rho c^2 v_y(P)) + \frac{\Delta t}{2}\int_0^{2\pi} S(Q)d\theta.$$

Dividing both sides by 2π gives

$$p(P) = \frac{1}{2\pi}\int_0^{2\pi} p(Q) + c\rho\cos\theta\ u(Q) + c\rho\sin\theta\ v(Q)d\theta + \frac{\Delta t}{2}(-\rho c^2 u_x(P) - \rho c^2 v_x(P)) + \frac{\Delta t}{4\pi}\int_0^{2\pi} S(Q)d\theta. \tag{48}$$

Like in the finite-difference method for the bicharacteristics method, we retain the problem of the cross-derivatives at the solution point P. Again, we will use the first equation of the PDE system,

$$p_t + \rho c^2 u_x + \rho c^2 v_y = 0.$$

Integrating this along the characteristic flow surface (x, y, t) to $(x, y, t + \Delta t)$ gives

$$p(P) - p(O) + \frac{\Delta t}{2}\rho c^2\left((u_x(P) + v_y(P)) + (u_x(O) + v_y(O))\right) = 0,$$

where O is the origin at $(0, 0, t)$. Eliminating the cross-derivative terms of (48) gives

$$p(P) = \frac{1}{\pi}\int_0^{2\pi} p(Q) + c\rho\cos\theta u(Q) + c\rho\sin v(Q)d\theta + \frac{\rho c^2\Delta t}{2}(u_x(O) + v_y(O)) - p(O) + \frac{\Delta t}{2\pi}\int_0^{2\pi} S(Q)d\theta. \tag{49}$$

We denote Q evaluated at 0 by Q_0, $\frac{\pi}{2}$ by Q_1 and so on. The numerical scheme for $p(P)$ then becomes

$$p(P) = \frac{1}{2}\left(p(Q_0) + p(Q_1) + p(Q_2) + p(Q_3)\right) + \frac{c\rho}{2}(u(Q_0) + v(Q_1) - u(Q_2) - v(Q_3))$$
$$+ \rho c^2 \frac{\Delta t}{2}(u_x(O) + v_y(O)) - p(O) - \frac{\Delta t}{4}(S(Q_0) + S(Q_1) + S(Q_2) + S(Q_3)),$$

where we have used the following numerical integration rule

$$\int_0^{2\pi} g(\theta)d\theta \approx \frac{\pi}{2}(g(0) + g(\frac{\pi}{2}) + g(\pi) + g(\frac{3\pi}{2})).$$

Substituting

$$S = -\rho c^2 \sin^2\theta u_x + \rho c^2 \cos\theta \sin\theta\ u_y + \rho c^2 \cos\theta \sin\theta\ v_x - \rho c^2 \cos^2\theta\ v_y$$

gives the following set of equations:

$$p(P) = \frac{1}{2}\left(p(Q_0) + p(Q_1) + p(Q_2) + p(Q_3)\right) + \frac{c\rho}{2}(u(Q_0) + v(Q_1) - u(Q_2) - v(Q_3))$$
$$+ \frac{\rho c^2 \Delta t}{2}(u_x(O) + v_y(O)) - p(O) - \rho c^2 \frac{\Delta t}{4}(v_y(Q_0) + u_x(Q_1) + v_y(Q_2) + u_x(Q_3))$$

Using the weight function $f(\theta) = \cos\theta$ gives

$$\pi\rho c\ u(P) = \int_0^{2\pi} + \cos\theta p(Q) + \rho c \cos^2\theta u(Q) + \rho c \cos\theta \sin\theta v(Q) d\theta$$
$$+ \frac{\Delta t}{2}\int_0^{2\pi} \cos\theta\ S(Q)d\theta.$$

So $u(P)$ is given by

$$u(P) = \frac{1}{\pi}\int_0^{2\pi} \frac{1}{\rho c}\cos\theta p(Q) + \cos^2\theta u(Q) + \cos\theta \sin\theta v(Q) d\theta$$
$$+ \frac{\Delta t}{2\pi\rho c}\int_0^{2\pi} \cos\theta\ S(Q)d\theta.$$

Using the weight function $f(\theta) = \sin\theta$ gives

$$\pi\rho c v(P) = \int_0^{2\pi} \sin\theta p(Q) + \rho c \sin\theta \cos\theta u(Q) + \rho c \sin^2\theta v(Q) d\theta$$
$$+ \frac{\Delta t}{2}\int_0^{2\pi} \sin\theta\ S(Q)d\theta.$$

And finally, $v(P)$ becomes

$$v(P) = \frac{1}{\pi}\int_0^{2\pi} -\frac{1}{\rho c}\sin\theta p(Q) + \sin\theta \cos\theta u(Q) + \sin^2\theta v(Q) d\theta$$
$$- \frac{\Delta t}{2\pi\rho c}\int_0^{2\pi} \sin\theta\ S(Q)d\theta.$$

We denote Q evaluated at 0 by Q_0, $\frac{\pi}{2}$ by Q_1 and so on. The numerical schemes for $p(P), u(P)$ and $v(P)$ then become

$$p(P) = \frac{1}{2}\left(p(Q_0) + p(Q_1) + p(Q_2) + p(Q_3)\right) + \frac{c\rho}{2}(u(Q_0) + v(Q_1) - u(Q_2) - v(Q_3)) + \rho c^2 \frac{\Delta t}{2}(u_x(O) + v_y(O)) - p(O) + \frac{\Delta t}{4}(S(Q_0) + S(Q_1) + S(Q_2) + S(Q_3))$$

$$u(P) = \frac{1}{2\rho c}(p(Q_0) - p(Q_2)) + \frac{1}{2}(u(Q_0) + u(Q_2)) + \frac{\Delta t}{4\rho c}(S(Q_0) - S(Q_2))$$

$$v(P) = \frac{1}{2\rho c}(p(Q_1) - p(Q_3)) + \frac{1}{2}(v(Q_1) + v(Q_3)) + \frac{\Delta t}{4\rho c}(S(Q_1) - S(Q_3))$$

Substituting

$$S = -\rho c^2 \sin^2\theta u_x + \rho c^2 \cos\theta \sin\theta \; u_y + \rho c^2 \cos\theta \sin\theta \; v_x - \rho c^2 \cos^2\theta \; v_y$$

gives the following set of equations:

$$p(P) = \frac{1}{2}\left(p(Q_0) + p(Q_1) + p(Q_2) + p(Q_3)\right) + \frac{c\rho}{2}(u(Q_0) + v(Q_1) - u(Q_2) - v(Q_3)) + \frac{\rho c^2 \Delta t}{2}(u_x(O) + v_y(O)) - p(O) - \rho c^2 \frac{\Delta t}{4}(v_y(Q_0) + u_x(Q_1) + v_y(Q_2) + u_x(Q_3))$$

$$u(P) = \frac{1}{2\rho c}(p(Q_0) - p(Q_2)) + \frac{1}{2}(u(Q_0) + u(Q_2)) - c\frac{\Delta t}{4}(v_y(Q_0) - v_y(Q_2))$$

$$v(P) = \frac{1}{2\rho c}(p(Q_1) - p(Q_3)) + \frac{1}{2}(v(Q_1) + v(Q_3)) - c\frac{\Delta t}{4}(u_x(Q_1) - u_x(Q_3))$$

We will take the same interpolation formula used in the previous section, the biparabolic interpolation polynomial $h(x, y)$. With the derivative terms we will take the respective derivative of $h(x, y)$. This will not decrease the order of accuracy, since the difference is multiplied by Δt.
The final numerical scheme becomes

$$p^{n+1}_{k,m} = \sum_{i,j=-1}^{1} (A^1_{i,j} p^n_{k+i,m+j} + B^1_{i,j} u^n_{k+i,m+j} + C^1_{i,j} v^n_{k+i,m+j})$$

$$u^{n+1}_{k,m} = \sum_{i,j=-1}^{1} (A^2_{i,j} p^n_{k+i,m+j} + B^2_{i,j} u^n_{k+i,m+j} + C^2_{i,j} v^n_{k+i,m+j})$$

$$v^{n+1}_{k,m} = \sum_{i,j=-1}^{1} (A^3_{i,j} p^n_{k+i,m+j} + B^3_{i,j} u^n_{k+i,m+j} + C^3_{i,j} v^n_{k+i,m+j}),$$

with the 3×3-matrices $A^1, A_2, \ldots$ given by

$$
\begin{aligned}
A^1 &= \begin{pmatrix} 0 & \frac{R^2}{2} & 0 \\ \frac{R^2}{2} & 1-2R^2 & \frac{R^2}{2} \\ 0 & \frac{R^2}{2} & 0 \end{pmatrix}, \\
B^1 &= \begin{pmatrix} -c\rho\frac{R^3}{8} & 0 & c\rho\frac{R^3}{8} \\ -\frac{c\rho R}{4}(2-R^2) & 0 & \frac{c\rho R}{4}(2-R^2) \\ -c\rho\frac{R^3}{8} & 0 & c\rho\frac{R^3}{8} \end{pmatrix} \\
C^1 &= \begin{pmatrix} c\rho\frac{R^3}{8} & \frac{c\rho R}{4}(2-R^2) & c\rho\frac{R^3}{8} \\ 0 & 0 & 0 \\ -c\rho\frac{R^3}{8} & -\frac{c\rho R}{4}(2-R^2) & -c\rho\frac{R^3}{8} \end{pmatrix} \\
A^2 &= \begin{pmatrix} 0 & 0 & 0 \\ -\frac{R}{2c\rho} & 0 & \frac{R}{2c\rho} \\ 0 & 0 & 0 \end{pmatrix} \\
B^2 &= \begin{pmatrix} 0 & 0 & 0 \\ \frac{R^2}{2} & 1-R^2 & \frac{R^2}{2} \\ 0 & 0 & 0 \end{pmatrix} \\
C^2 &= \begin{pmatrix} -\frac{R^2}{8} & 0 & \frac{R^2}{8} \\ 0 & 0 & 0 \\ \frac{R^2}{8} & 0 & -\frac{R^2}{8} \end{pmatrix} \\
A^3 &= \begin{pmatrix} 0 & \frac{R}{2c\rho} & 0 \\ 0 & 0 & 0 \\ 0 & -\frac{R}{2c\rho} & 0 \end{pmatrix} \\
B^3 &= \begin{pmatrix} -\frac{R^2}{8} & 0 & \frac{R^2}{8} \\ 0 & 0 & 0 \\ \frac{R^2}{8} & 0 & -\frac{R^2}{8} \end{pmatrix} \\
C^3 &= \begin{pmatrix} 0 & \frac{R^2}{2} & 0 \\ 0 & 1-R^2 & 0 \\ 0 & \frac{R^2}{2} & 0 \end{pmatrix},
\end{aligned}
$$

where $R = \frac{c\ dt}{dx} = C_R$.

10.1 Numerical analysis

We can determine the consistency order by calculating the consistency matrices given in Consistency section of the Numerical methods analysis, Section 9.4. As these calculations are rather tedious and so we used Mathematica to solve it. Tthe result is that these equations are at least third order consistent, so the total error of this method is $O(\Delta t^2)$.
The amplification matrix is given by, with taking $\zeta = e^{i\alpha dx}$, $\eta = e^{i\beta dy}$, $dx = dy$,

and $0 \leq x, y \leq 2\pi$:

$$\begin{pmatrix} 1-2R^2+R^2\cos(\alpha dx)+R^2\cos(\beta dx) & -\frac{i}{2}(2-R^2+R^2\cos(\beta dx))\sin(\alpha dx) & \frac{i}{2}R(2-R^2+R^2\cos(\alpha dx))\sin(\beta dx) \\ iR\sin(\alpha dx) & 1-R^2+R^2\cos(\alpha dx) & -\frac{1}{2}R^2\sin(\alpha dx)\sin(\beta dx) \\ iR\sin(\beta dx) & -\frac{1}{2}R^2\sin(\alpha dx)\sin(\beta dx) & 1-R^2+R^2\cos(\beta dx) \end{pmatrix}$$

The largest eigenvalue in absolute value of a matrix similar to the one above in [7] was found when $\alpha dx = \beta dx = \pi$. The matrix then becomes

$$\begin{pmatrix} 1-4R^2 & 0 & 0 \\ 0 & 1-2R^2 & 0 \\ 0 & 0 & 1-2R^2 \end{pmatrix}.$$

The eigenvalues of this matrix are

$$\lambda_1 = 1-4R^2$$
$$\lambda_{2,3} = 1-2R^2.$$

The requirement that the largest eigenvalue is equal or less than1 boils down to the requirement that

$$|R| \leq \frac{1}{\sqrt{2}}.$$

So, given a fixed dx, the value of dt will be taken as to be

$$dt = \frac{dx}{\sqrt{2}c},$$

and this will be used in Problems 1 to 3.

10.2 Problem 1

As in the previous two methods, we first look at how the numerical solution behaves when computing a sudden local pressure rise and its propagation. The results are given in the next three figures:

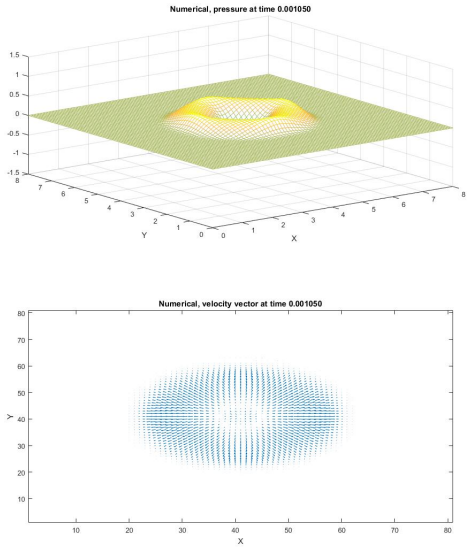


Figure 25: $t = 0.001$ s

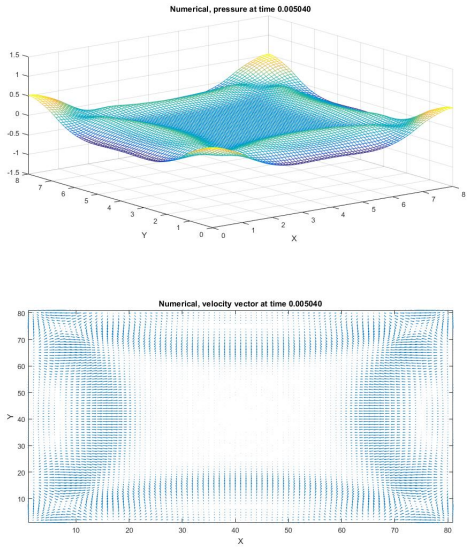


Figure 26: $t = 0.005$ s

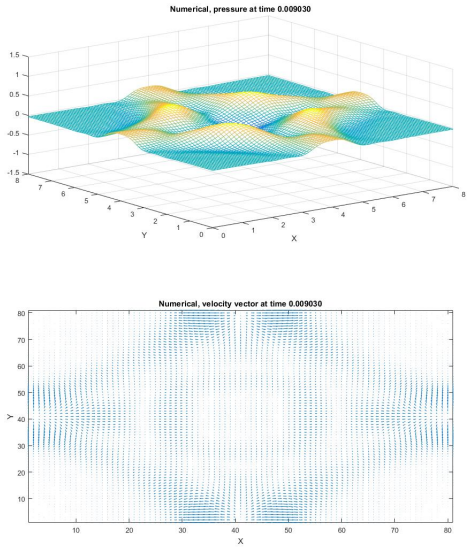


Figure 27: $t = 0.009$ s

The results are smooth, and (almost) equal to the results obtained in the characteristics-like method in Section 7.2.

10.3 Problem 2

The error values obtained for this method are given in the following table:

dX	0.1	0.05	0.04	0.02	0.01
Absolute error	$5.15 \cdot 10^{-3}$	$1.23 \cdot 10^{-3}$	$8.26 \cdot 10^{-4}$	$2.03 \cdot 10^{-4}$	$5.05 \cdot 10^{-5}$

The log values of dx, or h in the following graph, and the error values are given in the following figure:

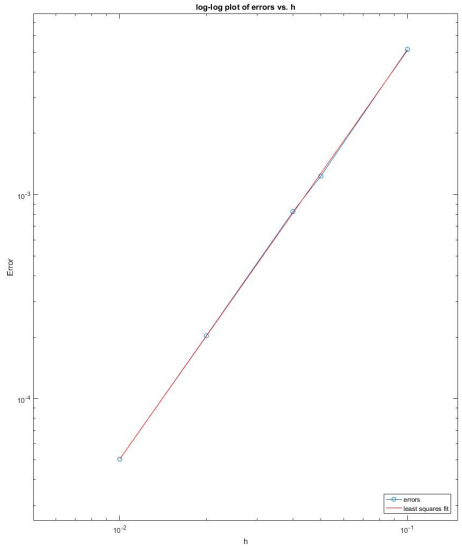


Figure 28: $t = 0.009$ s

The computed least-squares line is

$$E(h) = 0.515621 \cdot h^{2.00471},$$

and the estimated order of convergence is 2.0.
This agrees with the theoretical results from Chapter 6. This scheme, after a lot of tedious computations omitted here, is of second-order accuracy.

10.4 Problem 3

The results of the two dimensional waterhammer simulation at times $t = 0.01$ s and $t = 0.03$ s are given in the following figures:

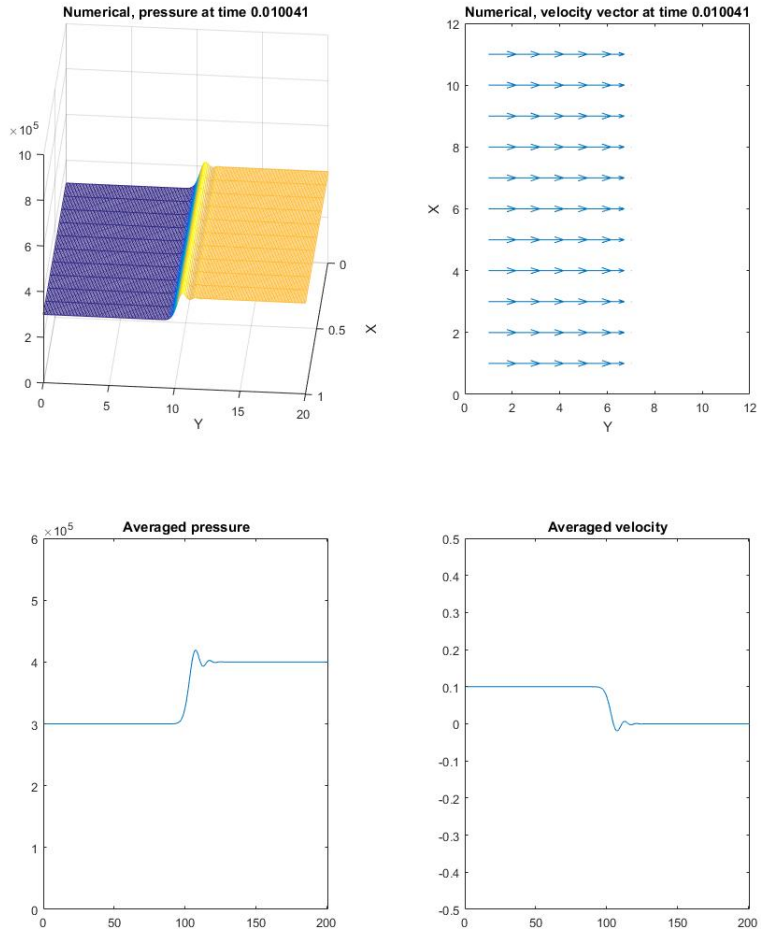


Figure 29: $t = 0.01$ s

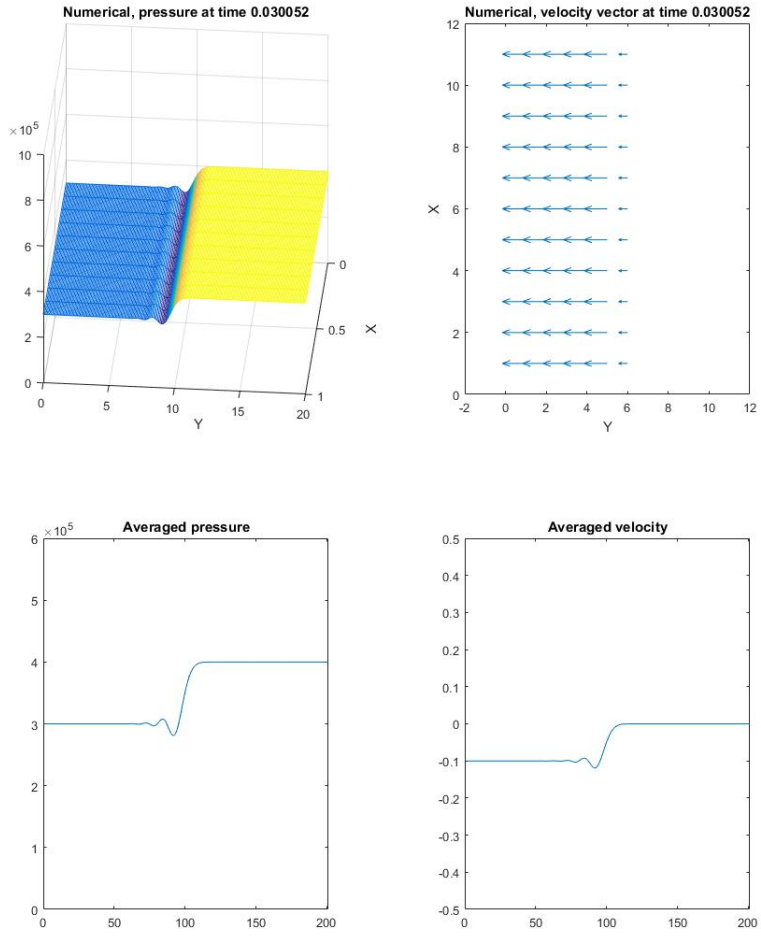


Figure 30: $t = 0.03$ s

We see that the results are not as sharp as in the case of the characteristic-like method, as considerable diffusion and non-physical oscillations occur at the shock. This is due to the fact that this method uses a considerable number of interpolations to calculate each step, which is known problem of linear second order numerical schemes, as they are not monotonicity preserving.

10.5 Conclusion

Butler's numerical scheme gives good results for smooth solutions, however when a discontinuity exists in the initial condition it gives non-physical results in the form of oscillations around the shock.
This method, contrary to the characteristic-like method, is straightforward to generalize to other problems, as shown in [8], and can also be implemented on numerical grids which are not rectangle shaped. This is due to integrating around a (part of) circle at the previous time level.
Furthermore, this scheme is easily extended to a conservative finite-volume scheme like in [8], and can also be reliably used for nonlinear partial differential equations, like the Euler equations, as shown in [9].

11 Conclusion

The goal of this work was to investigate the suitability of the bicharacteristics method for computing the linearised Euler equations. Three tests were devised for the numerical methods: the first was for checking any instabilities in the scheme, the second was for getting the practical order of accuracy and the last was for examining the suitability for discontinuous solutions, and how good the two-dimensional case compares to the one-dimensional case.
The main method of the report, the bicharacteristic method, is unconditionally unstable and therefore is not usable for solving the linearised Euler equations. Both the characteristic-like method and Butler's scheme give good results for smooth solutions, but only the characteristic-like method gives good results for discontinuous initial conditions. This is due to the use of interpolation between grid points in Butler's scheme.
However, the characteristic-like method can not easily be generalised to non-rectangular grids and other partial differential equations, while it is straightforward for Butler's scheme, for example the numerical schemes EG1, EG2 and EG3 in [8].
Furthermore, when computing the non-linearised Euler equations, M. Lukáčová-Medvid'ová showed that the scheme EG3 based on Butler's work [6] is suitable for computing solutions involving shocks.
Therefore we may conclude that the characteristic-like method is best for computing the linearised Euler equations on rectangular grids, but when the grid is not rectangular, or when one is interested in other hyperbolic partial differential equations, Butler's scheme is best suited, as it can be quite easily modified to give good results.

References

[1] Alexandre J. Chorin, Jerrol E. Marsden, *A Mathematical Introduction to Fluid Mechanics*, Springer-Verlag New York Inc., 1990

[2] M. Hanif Chaudhry, *Applied Hydraulic Transients*, Springer, 2014

[3] Lawrence C. Evans, *Partial Differential Equations*, American Mathematical Society, 1998

[4] E. Benjamin Wylie, Victor L. Streeter, Lisheng Suo, *Fluid Transients in Systems*, Prentice Hall, 1993

[5] Robert Anthony Delaney, *A second-order method of characteristics for two-dimensional unsteady flow with application to turbomachinery cascades*, Iowa State University, PhD thesis, 1974

[6] D.S. Butler, *The Numerical Solution of Hyperbolic Systems of Partial Differential Equations in Three Independent Variables*,Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences Sciences, Vol. 255, No. 1281 (Apr. 5, 1960)

[7] A. Sivasankara Redday, V.G. Tikekar and Phoolan Prasad *Numerical solution of hyperbolic equations by method of bicharacteristics*, Jour. Math. Phy. Sci. Vol. 16, No.6, 1982

[8] M. Lukáčová-Medvid'ová, K. W. Morton, and G. Warnecke, *Evolution Galerkin Methods for Hyperbolic Systems in Two Dimensions*, Mathematics of Computation, 200 Volume 69, Number 232, Pages 1355-1384

[9] M. Lukáčová-Medvid'ová, Jitka Saibertová, *Finite volume schemes for multi-dimensional hyperbolic systems based on the use of bicharacteristics*, Applications of Mathematics, Vol. 51, No. 3, Pages 205–228, 2006

A Mathematica code

In this appendix the Mathematica code will be given used in the report.

A.1 water-hammer.m

Main code for simulating the one dimensional water hammer equations.

```
% Solve the frictionless water hammer equations with the
    method of characteristics
% No interpolation

clear

%% Global variables and constants

makeMovie = 0; % make a movie of the plots
movieName = 'water hammer.avi';
frameRate = 25;

draw = 0; % if draw==drawLimit, update the the plots, and
    set draw=0
          % every computational loop draw is incremented
              by 1
drawLimit = 250;

%% Define fluid specific constants

rho = 1000; % Density of the fluid
diameter = 1; % Inner diameter of the pipe in meter
K = 2e9; % Bulk modulus
e = 8; % Thickness of the conduit walls
E = 200e9; % Young's modulus
length = 20; % length of the pipe in meter

c2 = (K/rho)/(1+(diameter*K)/(e*E));
c = sqrt(c2); % Sound of speed in the fluid

c = 1000; % Override the sound of speed

p0 = 3e5; % Initial pressure in the pipe
u0 = .1; % Initial velocity

%% Define grid variables

m = 999; % Number of interior grid points
```

```
36 m2 = m + 2; % Total number of grid points
37 dX = length/(m+1);
38 dT = dX/c;
39
40 t = 0;
41 T = (3 + .5)*length/c; % Total computation time
42
43 %% Set computational grid
44
45 x = (0:dX:length);
46
47 pN = zeros(m2,1); % computational grid for density
48 pO = pN; % store old values of pN
49
50 uN = zeros(m2,1); % computational grid for velocity
51 uO = zeros(m2,1);
52
53 %% Set initial conditions
54
55 for i = 1:m2
56     pN(i) = p0;
57     uN(i) = u0;
58 end
59
60 pO = pN;
61 uO = uN;
62
63 %% Movie options
64
65 if makeMovie
66     writerObj = VideoWriter(movieName);
67     writerObj.FrameRate = frameRate;
68     open(writerObj);
69 end
70
71 %% Plot figures
72
73 subplot(1,3,1)
74     plot(x,uN)
75     title('Velocity $V$', 'interpreter', 'latex')
76     xlabel('meter')
77     ylabel('$\frac{m}{s}$','interpreter','latex','
           rotation',0)
78     axis([0 length -1 1])
79
80     subplot(1,3,2)
```

```
81      plot(x,pN)
82      title('Pressure $p$', 'interpreter', 'latex')
83      xlabel('meter', 'interpreter','latex')
84      ylabel('Pascal', 'interpreter','latex')
85      axis([0 length -1e6 1e6])
86
87  subplot(1,3,3)
88      plot(x,uN*pi*diameter^2)
89      title('Discharge $Q$', 'interpreter', 'latex')
90      xlabel('meter', 'interpreter','latex')
91      ylabel('$\frac{m^3}{s}$', 'interpreter','latex','
            rotation',0)
92      axis([0 length -1 1])
93
94  pause(0.5)
95
96  %% Save to movie
97
98  if makeMovie
99      frame = getframe(gcf);
100     writeVideo(writerObj, frame);
101 end
102
103 %% Computational loop
104
105 while t <= T
106
107     %% Update time
108
109     t = t + dT;
110
111     %% Numerical computation
112
113     % Compute interior points
114     for i = 2:m2-1
115         pN(i) = .5*(pO(i+1) + pO(i-1)) + .5*rho*c*(uO(i
                -1) - uO(i+1));
116         uN(i) = .5*(pO(i-1) - pO(i+1))/(rho*c) + .5*(uO(i
                -1) + uO(i+1));
117     end
118
119     % Compute boundary conditions
120
121     pN(1) = p0;
122     uN(1) = (p0-pO(2))/(rho*c) + uO(2);
123
```

```
124     uN(m2) = 0;
125     pN(m2) = pO(m2-1) + rho*c*uO(m2-1);
126
127     % Update old values
128
129     pO = pN;
130     uO = uN;
131
132     %% Plot figures
133
134     draw = draw + 1;
135
136     if draw == drawLimit
137         subplot(1,3,1)
138             plot(x,uN)
139             title('Velocity $V$', 'interpreter', 'latex')
140             xlabel('meter')
141             ylabel('$\frac{m}{s}$','interpreter','latex',
                    'rotation',0)
142             axis([0 length -1 1])
143
144         subplot(1,3,2)
145             plot(x,pN)
146             title('Pressure $p$', 'interpreter', 'latex')
147             xlabel('meter', 'interpreter','latex')
148             ylabel('Pascal', 'interpreter','latex')
149             axis([0 length -1e6 1e6])
150
151         subplot(1,3,3)
152             plot(x,uN*pi*diameter^2)
153             title('Discharge $Q$', 'interpreter', 'latex'
                    )
154             xlabel('meter', 'interpreter','latex')
155             ylabel('$\frac{m^3}{s}$', 'interpreter','
                    latex','rotation',0)
156             axis([0 length -1 1])
157
158         drawnow
159         draw = 0;
160
161         %% Save to movie
162         if makeMovie
163             frame = getframe(gcf);
164             writeVideo(writerObj, frame);
165         end
166     end
```

```
167
168 end
169
170 %% Plot figures
171
172 subplot(1,3,1)
173     plot(x,uN)
174     title('Velocity $V$', 'interpreter', 'latex')
175     xlabel('meter')
176     ylabel('$\frac{m}{s}$','interpreter','latex','
        rotation',0)
177     axis([0 length -1 1])
178
179     subplot(1,3,2)
180     plot(x,pN)
181     title('Pressure $p$', 'interpreter', 'latex')
182     xlabel('meter', 'interpreter','latex')
183     ylabel('Pascal', 'interpreter','latex')
184     axis([0 length -1e6 1e6])
185
186 subplot(1,3,3)
187     plot(x,uN*pi*diameter^2)
188     title('Discharge $Q$', 'interpreter', 'latex')
189     xlabel('meter', 'interpreter','latex')
190     ylabel('$\frac{m^3}{s}$', 'interpreter','latex','
        rotation',0)
191     axis([0 length -1 1])
192
193 %% Save to movie and close writerObj
194 if makeMovie
195     frame = getframe(gcf);
196     writeVideo(writerObj, frame);
197
198     close(writerObj);
199 end
```

A.2 drop.m

Main code for simulation the local pressure rise.

```
1 % Solve drop problem of the wave equation, and plots the
    numerical
2 % solution
3
4 clear
```

```
5
6  %% Global variables and constants
7
8  makeMovie = 0; % make a movie of the plots
9  movieName = 'drop.avi';
10 frameRate = 25;
11
12 draw = 0; % if draw==drawLimit, update the the plots, and
       set draw=0
13 % every computational loop draw is incremented by 1
14 drawLimit = 10;
15
16 method = 'wylie'; % which method to compute the solution
17 % available: 'wylie', 'butler'
18
19 courantNumber = 0.7; % Only used in case of 'butler'
       method
20
21 %% Define problem specific constants
22
23 rho = 1000; % density
24 c = 1000; % speed of sound in the fluid
25
26 %% Define grid variables
27
28 xLength = 8;
29 yLength = 8;
30
31 m = 79; % number of interior grid points in the x-
       direction
32 m2 = m + 2; % total number of grid points
33
34 dX = xLength/(m+1);
35 dY = dX;
36 dT = dX/c;
37
38 if strcmp(method,'butler')
39     dT = courantNumber*dT;
40 end
41
42 nGridX = m2;
43 nGridY = ceil(yLength/dY) + 1;
44
45 t = 0;
46 T = 0.0025; % total computation time
47
```

```
48 %% Set computational grid
49
50 x = 0:dX:xLength;
51 y = 0:dY:yLength;
52 [X,Y] = meshgrid(x,y);
53
54 pN = zeros(nGridX,nGridY);
55 pO = pN;
56
57 uN = zeros(nGridX,nGridY);
58 uO = pN;
59
60 vN = zeros(nGridX,nGridY);
61 vO = pN;
62
63 if strcmp(method, 'wylie')
64     qN = zeros(nGridX,nGridY);
65     qO = pN;
66 end
67
68 % Set the analytical solution grid
69 pSol = pN;
70 uSol = uN;
71 vSol = vN;
72
73 %% Set initial conditions
74
75 for i = 1:nGridX
76     for j = 1:nGridY
77
78         if (x(i)-xLength/2)^2 + (y(j)-yLength/2)^2 < 1
79             pN(i,j) = pN(i,j) + 10*exp(-1/(1-(x(i)-
                    xLength/2)^2))*exp(-1/(1-(y(j)-yLength/2)
                    ^2));
80         end
81
82         uN(i,j) = 0;
83         vN(i,j) = 0;
84
85         if strcmp(method,'wylie')
86             qN(i,j) = 0;
87         end
88     end
89 end
90
91 pO = pN;
```

```
92  pSol = pN;
93
94  uO = uN;
95  uSol = uN;
96
97  vO = vN;
98  vSol = vN;
99
100 %% Movie options
101
102 if makeMovie
103     writerObj = VideoWriter(movieName);
104     writerObj.FrameRate = frameRate;
105     open(writerObj);
106 end
107
108 %% Plot figures
109
110 subplot(2,1,1)
111 mesh(X',Y',pN);
112 xlabel('X')
113 xlim([0 xLength]);
114 ylabel('Y')
115 ylim([0 yLength]);
116 zlim([-1.5 1.5])
117 title(sprintf('Numerical, pressure at time %f',t))
118
119 subplot(2,1,2)
120 quiver(uN,vN);
121 xlabel('X')
122 ylabel('Y')
123 zlim([-2 2])
124 title(sprintf('Numerical, velocity vector at time %f',t))
125
126 pause(1)
127
128 %% Save to movie
129
130 if makeMovie
131     frame = getframe(gcf);
132     writeVideo(writerObj, frame);
133 end
134
135 %% Computational loop
136
137 while t <= T
```

```
138
139     %% Update time
140
141     t = t + dT
142
143
144     %% Compute interior points
145
146     for i = 2:nGridX-1
147         for j = 2:nGridY-1
148             if strcmp(method,'wylie')
149
150                 e1 = pO(i-1,j-1)/(rho*c) + uO(i-1,j-1) +
                        vO(i-1,j-1) + qO(i-1,j-1)*dX;
151                 e2 = pO(i-1,j+1)/(rho*c) + uO(i-1,j+1) -
                        vO(i-1,j+1) - qO(i-1,j+1)*dX;
152                 e3 = pO(i+1,j+1)/(rho*c) - uO(i+1,j+1) -
                        vO(i+1,j+1) + qO(i+1,j+1)*dX;
153                 e4 = pO(i+1,j-1)/(rho*c) - uO(i+1,j-1) +
                        vO(i+1,j-1) - qO(i+1,j-1)*dX;
154
155                 pN(i,j) = rho*c*(e1+e2+e3+e4)/4;
156                 uN(i,j) = (e1+e2-e3-e4)/4;
157                 vN(i,j) = (e1-e2-e3+e4)/4;
158                 qN(i,j) = (-e1+e2-e3+e4)/(4*dX);
159
160
161             elseif strcmp(method,'butler')
162
163                 R = c*dT/dX;
164
165                 A1 = [0, R*R/2, 0; R*R/2, 1-2*R*R, R*R/2;
                          0, R*R/2, 0];
166                 B1 = [-c*rho*R*R*R/8, 0, c*rho*R*R*R/8; -
                        c*rho*R*(2-R*R)/4, 0, c*rho*R*(2-R*R)
                        /4    ; -c*rho*R*R*R/8, 0, c*rho*R*R*R
                        /8];
167                 C1 = [c*rho*R*R*R/8, c*rho*R*(2-R*R)/4 ,
                        c*rho*R*R*R/8; 0, 0, 0; -c*rho*R*R*R
                        /8, -c*rho*R*(2-R*R)/4 , -c*rho*R*R*R
                        /8];
168
169                 A2 = [0, 0, 0; -R/(2*c*rho), 0, R/(2*c*
                        rho); 0, 0, 0];
170                 B2 = [0, 0, 0; R*R/2, 1-R*R, R*R/2; 0, 0,
                          0];
```

```
171                     C2 = [-R*R/8, 0, R*R/8; 0, 0, 0; R*R/8,
                            0, -R*R/8];
172
173                     A3 = [0, R/(2*c*rho), 0; 0, 0, 0; 0, -R
                            /(2*c*rho), 0];
174                     B3 = [-R*R/8, 0, R*R/8; 0, 0, 0; R*R/8,
                            0, -R*R/8];
175                     C3 = [0, R*R/2, 0; 0, 1-R*R, 0; 0, R*R/2,
                             0];
176
177                     pN(i,j) = 0;
178                     uN(i,j) = 0;
179                     vN(i,j) = 0;
180
181                     for k1 = 1:3
182                         for k2 = 1:3
183                             c1 = i + 2 - k1;
184                             c2 = j + 2 - k2;
185                             pN(i,j) = pN(i,j) + A1(4-k2,k1)*
                                    pO(c1,c2) + B1(4-k2,k1)*uO(c1,
                                    c2) + C1(4-k2,k1)*vO(c1,c2);
186                             uN(i,j) = uN(i,j) + A2(4-k2,k1)*
                                    pO(c1,c2) + B2(4-k2,k1)*uO(c1,
                                    c2) + C2(4-k2,k1)*vO(c1,c2);
187                             vN(i,j) = vN(i,j) + A3(4-k2,k1)*
                                    pO(c1,c2) + B3(4-k2,k1)*uO(c1,
                                    c2) + C3(4-k2,k1)*vO(c1,c2);
188                         end
189                     end
190
191
192                 end
193             end
194         end
195
196         %% Set boundary points
197
198         pN(1,1:nGridY) = pN(2,1:nGridY);
199         pN(nGridX,1:nGridY) = pN(nGridX-1,1:nGridY);
200         pN(1:nGridX,1) = pN(1:nGridX,2);
201         pN(1:nGridX,nGridY) = pN(1:nGridX,nGridY-1);
202
203         uN(1,1:nGridY) = -uN(2,1:nGridY);
204         uN(nGridX,1:nGridY) = -uN(nGridX-1,1:nGridY);
205         uN(1:nGridX,1) = uN(1:nGridX,2);
206         uN(1:nGridX,nGridY) = uN(1:nGridX,nGridY-1);
```

```
207
208     vN(1,1:nGridY) = vN(2,1:nGridY);
209     vN(nGridX,1:nGridY) = vN(nGridX-1,1:nGridY);
210     vN(1:nGridX,1) = -vN(1:nGridX,2);
211     vN(1:nGridX,nGridY) = -vN(1:nGridX,nGridY-1);
212
213     if strcmp(method,'wylie')
214         qN(1,1:nGridY) = -qN(2,1:nGridY);
215         qN(nGridX,1:nGridY) = -qN(nGridX-1,1:nGridY);
216         qN(1:nGridX,1) = -qN(1:nGridX,2);
217         qN(1:nGridX,nGridY) = -qN(1:nGridX,nGridY-1);
218     end
219
220     %% Update old values
221
222     pO = pN;
223     uO = uN;
224     vO = vN;
225
226     if strcmp(method,'wylie')
227         qO = qN;
228     end
229
230     %% Plot figures
231
232     if draw == drawLimit
233
234         % Show time
235         t
236
237
238         draw = 0;
239         subplot(2,1,1)
240         mesh(X',Y',pN);
241         xlabel('X')
242         xlim([0 xLength]);
243         ylabel('Y')
244         ylim([0 yLength]);
245         zlim([-1.5 1.5])
246         title(sprintf('Numerical, pressure at time %f',t)
                )
247
248         subplot(2,1,2)
249         quiver(uN',vN');
250         xlabel('X')
251         xlim([1 nGridX])
```

```
252             ylabel('Y')
253             ylim([1 nGridY])
254             title(sprintf('Numerical, velocity vector at time
                    %f',t))
255
256
257             drawnow
258
259             %% Save to movie
260
261             if makeMovie
262                 frame = getframe(gcf);
263                 writeVideo(writerObj, frame);
264             end
265         end
266
267         draw = draw + 1;
268
269
270     end
271
272
273     %% Plot figures
274
275     subplot(2,1,1)
276     mesh(X',Y',pN);
277     xlabel('X')
278     xlim([0 xLength]);
279     ylabel('Y')
280     ylim([0 yLength]);
281     zlim([-1.5 1.5])
282     % title(sprintf('Numerical, pressure at time %f',t))
283
284     subplot(2,1,2)
285     quiver(uN',vN');
286     xlabel('X')
287     xlim([1 nGridX])
288     ylabel('Y')
289     ylim([1 nGridY])
290     % title(sprintf('Numerical, velocity vector at time %f',t
            ))
291
292
293     drawnow
294
295     %% Movie steps
```

```
296
297 if makeMovie
298     frame = getframe(gcf);
299     writeVideo(writerObj, frame);
300
301     close(writerObj);
302 end
```

A.3 analytical-plot.m

Main code for generating plots of the analytical solution and the corresponding numerical solution.

```
1  % Solve the test solution of the wave equation, and plots
       the numerical
2  % solution and the analytical solution
3
4  clear
5
6  %% Global variables and constants
7
8  makeMovie = 0; % make a movie of the plots
9  movieName = 'analytical solution.avi';
10 frameRate = 25;
11
12 draw = 0; % if draw==drawLimit, update the the plots, and
       set draw=0
13 % every computational loop draw is incremented by 1
14 drawLimit = 1;
15
16 method = 'butler'; % which method to compute the solution
17 % available: 'wylie', 'butler'
18
19 courantNumber = 1/sqrt(2); % Only used in case of 'butler
       ' method
20
21 %% Define problem specific constants
22
23 rho = 1000; % density
24 c = 1000; % speed of sound in the fluid
25
26 %% Define grid variables
27
28 xLength = 1;
29 yLength = 1;
```

```
30
31 m = 9; % number of interior grid points in the x-
       direction
32 m2 = m + 2; % total number of grid points
33
34 dX = xLength/(m+1);
35 dY = dX;
36 dT = dX/c;
37
38 if strcmp(method,'butler')
39     dT = courantNumber*dT;
40 end
41
42 nGridX = m2;
43 nGridY = ceil(yLength/dY) + 1;
44
45 t = 0;
46 T = 1; % total computation time
47
48 %% Set computational grid
49
50 x = 0:dX:xLength;
51 y = 0:dY:yLength;
52 [X,Y] = meshgrid(x,y);
53
54 pN = zeros(nGridX,nGridY);
55 pO = pN;
56
57 uN = zeros(nGridX,nGridY);
58 uO = pN;
59
60 vN = zeros(nGridX,nGridY);
61 vO = pN;
62
63 if strcmp(method, 'wylie')
64     qN = zeros(nGridX,nGridY);
65     qO = pN;
66 end
67
68 % Set the analytical solution grid
69 pSol = pN;
70 uSol = uN;
71 vSol = vN;
72
73 %% Set initial conditions
74
```

```
75  for i = 1:nGridX
76      for j = 1:nGridY
77          [p1, u1, v1] = sol(x(i),y(j),t,xLength,yLength,c,
                rho);
78
79          pN(i,j) = p1;
80          uN(i,j) = u1;
81          vN(i,j) = v1;
82
83          if strcmp(method,'near-characteristic')
84              qN(i,j) = qSolution(x(i),y(j),t,xLength,
                    yLength,c,rho);
85          end
86      end
87  end
88
89  pO = pN;
90  pSol = pN;
91
92  uO = uN;
93  uSol = uN;
94
95  vO = vN;
96  vSol = vN;
97
98  %% Movie options
99
100 if makeMovie
101     writerObj = VideoWriter(movieName);
102     writerObj.FrameRate = frameRate;
103     open(writerObj);
104 end
105
106 %% Plot figures
107
108 subplot(2,2,1)
109 mesh(X',Y',pN);
110 xlabel('X')
111 xlim([0 xLength]);
112 ylabel('Y')
113 ylim([0 yLength]);
114 zlim([-1.5 1.5])
115 title(sprintf('Numerical, pressure at time %f',t))
116
117 subplot(2,2,2)
118 quiver(uN,vN);
```

```
119 xlabel('X')
120 ylabel('Y')
121 zlim([-2  2])
122 title(sprintf('Numerical, velocity vector at time %f',t))
123
124 subplot(2,2,3)
125 mesh(X',Y',pSol);
126 xlabel('X')
127 xlim([0  xLength]);
128 ylabel('Y')
129 ylim([0  yLength]);
130 zlim([-1.5  1.5])
131 title(sprintf('Analytical, pressure at time %f',t))
132
133 subplot(2,2,4)
134 quiver(uSol,vSol);
135 xlabel('X')
136 xlim([-2  nGridX+2]);
137 ylabel('Y')
138 ylim([-2  nGridY+2]);
139 zlim([-2  2])
140 title(sprintf('Analytical, velocity vector at time %f',t)
    )
141
142 pause(1)
143
144 %% Save to movie
145
146 if makeMovie
147     frame = getframe(gcf);
148     writeVideo(writerObj, frame);
149 end
150
151 %% Computational loop
152
153 while t <= T
154
155     %% Update time
156
157     t = t + dT
158
159     %% Obtain analytical solution
160
161     for i = 1:nGridX
162         for j = 1:nGridY
```

```
163             [p1, u1, v1] = sol(x(i),y(j),t,xLength,
                    yLength,c,rho);
164
165             pSol(i,j) = p1;
166             uSol(i,j) = u1;
167             vSol(i,j) = v1;
168
169             if strcmp(method,'near-characteristic')
170                 qSol(i,j) = qSolution(x(i),y(j),t,xLength
                        ,yLength,c,rho);
171             end
172         end
173     end
174
175     %% Compute interior points
176
177     for i = 2:nGridX-1
178         for j = 2:nGridY-1
179             if strcmp(method,'wylie')
180
181                 e1 = pO(i-1,j-1)/(rho*c) + uO(i-1,j-1) +
                        vO(i-1,j-1) + qO(i-1,j-1)*dX;
182                 e2 = pO(i-1,j+1)/(rho*c) + uO(i-1,j+1) -
                        vO(i-1,j+1) - qO(i-1,j+1)*dX;
183                 e3 = pO(i+1,j+1)/(rho*c) - uO(i+1,j+1) -
                        vO(i+1,j+1) + qO(i+1,j+1)*dX;
184                 e4 = pO(i+1,j-1)/(rho*c) - uO(i+1,j-1) +
                        vO(i+1,j-1) - qO(i+1,j-1)*dX;
185
186                 pN(i,j) = rho*c*(e1+e2+e3+e4)/4;
187                 uN(i,j) = (e1+e2-e3-e4)/4;
188                 vN(i,j) = (e1-e2-e3+e4)/4;
189                 qN(i,j) = (-e1+e2-e3+e4)/(4*dX);
190
191
192             elseif strcmp(method,'butler')
193
194                 R = c*dT/dX;
195
196                 A1 = [0, R*R/2, 0; R*R/2, 1-2*R*R, R*R/2;
                        0, R*R/2, 0];
197                 B1 = [-c*rho*R*R*R/8, 0, c*rho*R*R*R/8; -
                        c*rho*R*(2-R*R)/4, 0, c*rho*R*(2-R*R)
                        /4    ; -c*rho*R*R*R/8, 0, c*rho*R*R*R
                        /8];
```

```
198                     C1 = [c*rho*R*R*R/8, c*rho*R*(2-R*R)/4 ,
                            c*rho*R*R*R/8; 0, 0, 0; -c*rho*R*R*R
                            /8, -c*rho*R*(2-R*R)/4 , -c*rho*R*R*R
                            /8];
199
200                     A2 = [0, 0, 0; -R/(2*c*rho), 0, R/(2*c*
                            rho); 0, 0, 0];
201                     B2 = [0, 0, 0; R*R/2, 1-R*R, R*R/2; 0, 0,
                             0];
202                     C2 = [-R*R/8, 0, R*R/8; 0, 0, 0; R*R/8,
                            0, -R*R/8];
203
204                     A3 = [0, R/(2*c*rho), 0; 0, 0, 0; 0, -R
                            /(2*c*rho), 0];
205                     B3 = [-R*R/8, 0, R*R/8; 0, 0, 0; R*R/8,
                            0, -R*R/8];
206                     C3 = [0, R*R/2, 0; 0, 1-R*R, 0; 0, R*R/2,
                             0];
207
208                     pN(i,j) = 0;
209                     uN(i,j) = 0;
210                     vN(i,j) = 0;
211
212                     for k1 = 1:3
213                         for k2 = 1:3
214                             c1 = i + 2 - k1;
215                             c2 = j + 2 - k2;
216                             pN(i,j) = pN(i,j) + A1(4-k2,k1)*
                                    pO(c1,c2) + B1(4-k2,k1)*uO(c1,
                                    c2) + C1(4-k2,k1)*vO(c1,c2);
217                             uN(i,j) = uN(i,j) + A2(4-k2,k1)*
                                    pO(c1,c2) + B2(4-k2,k1)*uO(c1,
                                    c2) + C2(4-k2,k1)*vO(c1,c2);
218                             vN(i,j) = vN(i,j) + A3(4-k2,k1)*
                                    pO(c1,c2) + B3(4-k2,k1)*uO(c1,
                                    c2) + C3(4-k2,k1)*vO(c1,c2);
219                         end
220                     end
221
222
223                 end
224             end
225         end
226
227         %% Set boundary points
228
```

```
229     pN(1:nGridX,1) = pSol(1:nGridX,1);
230     pN(1:nGridX,nGridY) = pSol(1:nGridX,nGridY);
231
232     pN(1,1:nGridY) = pSol(1,1:nGridY);
233     pN(nGridX,1:nGridY) = pSol(nGridX,1:nGridY);
234
235     uN(1:nGridX,1) = uSol(1:nGridX,1);
236     uN(1:nGridX,nGridY) = uSol(1:nGridX,nGridY);
237
238     uN(1,1:nGridY) = uSol(1,1:nGridY);
239     uN(nGridX,1:nGridY) = uSol(nGridX,1:nGridY);
240
241     vN(1:nGridX,1) = vSol(1:nGridX,1);
242     vN(1:nGridX,nGridY) = vSol(1:nGridX,nGridY);
243
244     vN(1,1:nGridY) = vSol(1,1:nGridY);
245     vN(nGridX,1:nGridY) = vSol(nGridX,1:nGridY);
246
247     if strcmp(method,'wylie')
248         for i = 1:nGridX
249             qN(i,1) = qSol(x(i),0,t,xLength,yLength,c,rho
                    );
250             qN(i,nGridY) = qSol(x(i),yLength,t,xLength,
                    yLength,c,rho);
251         end
252
253         for j = 1:nGridY
254             qN(1,j) = qSol(0,y(j),t,xLength,yLength,c,rho
                    );
255             qN(nGridX,j) = qSol(xLength,y(j),t,xLength,
                    yLength,c,rho);
256         end
257     end
258
259     %% Update old values
260
261     pO = pN;
262     uO = uN;
263     vO = vN;
264
265     if strcmp(method,'wylie')
266         qO = qN;
267     end
268
269     %% Plot figures
270
```

```
271         subplot(2,2,1)
272         mesh(X',Y',pN);
273         xlabel('X')
274         xlim([0  xLength]);
275         ylabel('Y')
276         ylim([0  yLength]);
277         zlim([-1.5  1.5])
278         title(sprintf('Numerical, pressure at time %f',t))
279 
280         subplot(2,2,2)
281         quiver(uN',vN');
282         xlabel('X')
283         xlim([1  nGridX])
284         ylabel('Y')
285         ylim([1  nGridY])
286         title(sprintf('Numerical, velocity vector at time %f'
                ,t))
287 
288         subplot(2,2,3)
289         mesh(X',Y',pSol);
290         xlabel('X')
291         xlim([0  xLength]);
292         ylabel('Y')
293         ylim([0  yLength]);
294         zlim([-1.5  1.5])
295         title(sprintf('Analytical, pressure at time %f',t))
296 
297         subplot(2,2,4)
298         quiver(uSol',vSol');
299         xlabel('X')
300         xlim([1  nGridX])
301         ylabel('Y')
302         ylim([1  nGridY])
303         title(sprintf('Analytical, velocity vector at time %f
                ',t))
304 
305         drawnow
306 end
```

A.4 analytical-error.m

Main code for generating the EOC and computing the error values.

```
1 % Solve the test solution of the wave equation, and plots
      the numerical
```

```
% solution and the analytical solution

clear

%% Global variables and constants

method = 'butler'; % which method to compute the solution
% available: 'wylie', 'butler'

courantNumber = 1/sqrt(2); % Only used in case of 'butler
    ' method

% Set the cases for getting the errors.
nCases = 5;
testCases = [10 20 25 50 100];
h = 1./testCases;
E = zeros(nCases,1);

%% Define problem specific constants

rho = 1000; % density
c = 1000; % speed of sound in the fluid

for k = 1:nCases



    %% Define grid variables

    xLength = 1;
    yLength = 1;

    m = testCases(k)-1; % number of interior grid points
        in the x-direction
    m2 = m + 2; % total number of grid points

    dX = xLength/(m+1);
    dY = dX;
    dT = dX/c;

    disp(sprintf('Calculating error for h=%f',dX))

    if strcmp(method,'butler')
        dT = courantNumber*dT;
    end

```

```
46      nGridX = m2;
47      nGridY = ceil(yLength/dY) + 1;
48
49      t = 0;
50      T = 0.005; % total computation time
51
52      %% Set computational grid
53
54      x = 0:dX:xLength;
55      y = 0:dY:yLength;
56      [X,Y] = meshgrid(x,y);
57
58      pN = zeros(nGridX,nGridY);
59      pO = pN;
60
61      uN = zeros(nGridX,nGridY);
62      uO = pN;
63
64      vN = zeros(nGridX,nGridY);
65      vO = pN;
66
67      if strcmp(method, 'wylie')
68          qN = zeros(nGridX,nGridY);
69          qO = pN;
70      end
71
72      % Set the analytical solution grid
73      pSol = pN;
74      uSol = uN;
75      vSol = vN;
76
77      %% Set initial conditions
78
79      for i = 1:nGridX
80          for j = 1:nGridY
81              [p1, u1, v1] = sol(x(i),y(j),t,xLength,
                    yLength,c,rho);
82
83              pN(i,j) = p1;
84              uN(i,j) = u1;
85              vN(i,j) = v1;
86
87              if strcmp(method,'near-characteristic')
88                  qN(i,j) = qSolution(x(i),y(j),t,xLength,
                        yLength,c,rho);
89              end
```

```
90          end
91      end
92
93      pO = pN;
94      pSol = pN;
95
96      uO = uN;
97      uSol = uN;
98
99      vO = vN;
100     vSol = vN;
101
102     %% Computational loop
103
104     while t <= T
105
106         %% Update time
107
108         t = t + dT;
109
110         %% Obtain analytical solution
111
112         for i = 1:nGridX
113             for j = 1:nGridY
114                 [p1, u1, v1] = sol(x(i),y(j),t,xLength,
                        yLength,c,rho);
115
116                 pSol(i,j) = p1;
117                 uSol(i,j) = u1;
118                 vSol(i,j) = v1;
119
120                 if strcmp(method,'near-characteristic')
121                     qSol(i,j) = qSolution(x(i),y(j),t,
                            xLength,yLength,c,rho);
122                 end
123             end
124         end
125
126         %% Compute interior points
127
128         for i = 2:nGridX-1
129             for j = 2:nGridY-1
130                 if strcmp(method,'wylie')
131
132                     e1 = pO(i-1,j-1)/(rho*c) + uO(i-1,j
                            -1) + vO(i-1,j-1) + qO(i-1,j-1)*dX
```

```
        ;
133     e2 = pO(i-1,j+1)/(rho*c) + uO(i-1,j
            +1) - vO(i-1,j+1) - qO(i-1,j+1)*dX
            ;
134     e3 = pO(i+1,j+1)/(rho*c) - uO(i+1,j
            +1) - vO(i+1,j+1) + qO(i+1,j+1)*dX
            ;
135     e4 = pO(i+1,j-1)/(rho*c) - uO(i+1,j
            -1) + vO(i+1,j-1) - qO(i+1,j-1)*dX
            ;
136
137     pN(i,j) = rho*c*(e1+e2+e3+e4)/4;
138     uN(i,j) = (e1+e2-e3-e4)/4;
139     vN(i,j) = (e1-e2-e3+e4)/4;
140     qN(i,j) = (-e1+e2-e3+e4)/(4*dX);
141
142
143 elseif strcmp(method,'butler')
144
145     R = c*dT/dX;
146
147     A1 = [0, R*R/2, 0; R*R/2, 1-2*R*R, R*
            R/2; 0, R*R/2, 0];
148     B1 = [-c*rho*R*R*R/8, 0, c*rho*R*R*R
            /8; -c*rho*R*(2-R*R)/4, 0, c*rho*R
            *(2-R*R)/4    ; -c*rho*R*R*R/8, 0,
            c*rho*R*R*R/8];
149     C1 = [c*rho*R*R*R/8, c*rho*R*(2-R*R)
            /4 , c*rho*R*R*R/8; 0, 0, 0; -c*
            rho*R*R*R/8, -c*rho*R*(2-R*R)/4 ,
            -c*rho*R*R*R/8];
150
151     A2 = [0, 0, 0; -R/(2*c*rho), 0, R/(2*
            c*rho); 0, 0, 0];
152     B2 = [0, 0, 0; R*R/2, 1-R*R, R*R/2;
            0, 0, 0];
153     C2 = [-R*R/8, 0, R*R/8; 0, 0, 0; R*R
            /8, 0, -R*R/8];
154
155     A3 = [0, R/(2*c*rho), 0; 0, 0, 0; 0,
            -R/(2*c*rho), 0];
156     B3 = [-R*R/8, 0, R*R/8; 0, 0, 0; R*R
            /8, 0, -R*R/8];
157     C3 = [0, R*R/2, 0; 0, 1-R*R, 0; 0, R*
            R/2, 0];
158
```

```
159                     pN(i,j) = 0;
160                     uN(i,j) = 0;
161                     vN(i,j) = 0;
162
163                     for k1 = 1:3
164                         for k2 = 1:3
165                             c1 = i + 2 - k1;
166                             c2 = j + 2 - k2;
167                             pN(i,j) = pN(i,j) + A1(4-k2,
                                    k1)*pO(c1,c2) + B1(4-k2,k1
                                    )*uO(c1,c2) + C1(4-k2,k1)*
                                    vO(c1,c2);
168                             uN(i,j) = uN(i,j) + A2(4-k2,
                                    k1)*pO(c1,c2) + B2(4-k2,k1
                                    )*uO(c1,c2) + C2(4-k2,k1)*
                                    vO(c1,c2);
169                             vN(i,j) = vN(i,j) + A3(4-k2,
                                    k1)*pO(c1,c2) + B3(4-k2,k1
                                    )*uO(c1,c2) + C3(4-k2,k1)*
                                    vO(c1,c2);
170                         end
171                     end
172
173
174                 end
175             end
176         end
177
178         %% Set boundary points
179
180         pN(1:nGridX,1) = pSol(1:nGridX,1);
181         pN(1:nGridX,nGridY) = pSol(1:nGridX,nGridY);
182
183         pN(1,1:nGridY) = pSol(1,1:nGridY);
184         pN(nGridX,1:nGridY) = pSol(nGridX,1:nGridY);
185
186         uN(1:nGridX,1) = uSol(1:nGridX,1);
187         uN(1:nGridX,nGridY) = uSol(1:nGridX,nGridY);
188
189         uN(1,1:nGridY) = uSol(1,1:nGridY);
190         uN(nGridX,1:nGridY) = uSol(nGridX,1:nGridY);
191
192         vN(1:nGridX,1) = vSol(1:nGridX,1);
193         vN(1:nGridX,nGridY) = vSol(1:nGridX,nGridY);
194
195         vN(1,1:nGridY) = vSol(1,1:nGridY);
```

```
196             vN(nGridX,1:nGridY) = vSol(nGridX,1:nGridY);
197
198             if strcmp(method,'wylie')
199                 for i = 1:nGridX
200                     qN(i,1) = qSol(x(i),0,t,xLength,yLength,c
                            ,rho);
201                     qN(i,nGridY) = qSol(x(i),yLength,t,
                            xLength,yLength,c,rho);
202                 end
203
204                 for j = 1:nGridY
205                     qN(1,j) = qSol(0,y(j),t,xLength,yLength,c
                            ,rho);
206                     qN(nGridX,j) = qSol(xLength,y(j),t,
                            xLength,yLength,c,rho);
207                 end
208             end
209
210             %% Update old values
211
212             pO = pN;
213             uO = uN;
214             vO = vN;
215
216             if strcmp(method,'wylie')
217                 qO = qN;
218             end
219
220             %% Store error values
221
222             E(k) = norm2(pN,pSol,dX,dY);
223
224         end
225     end
226
227     %% Compute and plot the log-log errors
228     error_loglog(h,E)
229     error_table(h,E)
```

A.5 'water-hammer-two-dimensional.m'

Main code for simulating the two dimensional water hammer equations.

```
1 % Solve the 2d water hammer equation, with instantaneous
      valve closure
```

```
2
3  clear
4
5  %% Global variables and constants
6
7  makeMovie = 0; % make a movie of the plots
8  movieName = 'water hammer 2d.avi';
9  frameRate = 25;
10
11 draw = 0; % if draw==drawLimit, update the the plots, and
       set draw=0
12 % every computational loop draw is incremented by 1
13 drawLimit = 10;
14
15 method = 'butler'; % which method to compute the solution
16 % available: 'wylie', 'butler'
17
18 courantNumber = 1/sqrt(2); % Only used in case of 'butler
       ' method
19
20 viewpoint = 'side'; % Viewpoint of pressure distribution
21 % 'side' for side view
22 % 'top' for top view
23
24 %% Define problem specific constants
25
26 rho = 1000; % density
27 c = 1000; % speed of sound in the fluid
28
29 p0 = 3e5;
30 v0 = 0.1;
31 u0 = 0;
32
33 %% Define grid variables
34
35 xLength = 1;
36 yLength = 20;
37
38 m = 9; % number of interior grid points in the x-
       direction
39 m2 = m + 2; % total number of grid points
40
41 dX = xLength/(m+1);
42 dY = dX;
43 dT = dX/c;
44
```

```
45 if strcmp(method,'butler')
46     dT = courantNumber*dT;
47 end
48
49 nGridX = m2;
50 nGridY = ceil(yLength/dY) + 1;
51
52 t = 0;
53 T = 0.03; % total computation time
54
55 %% Set computational grid
56
57 x = 0:dX:xLength;
58 y = 0:dY:yLength;
59 [X,Y] = meshgrid(x,y);
60
61 pN = zeros(nGridX,nGridY);
62 pO = pN;
63
64 uN = zeros(nGridX,nGridY);
65 uO = pN;
66
67 vN = zeros(nGridX,nGridY);
68 vO = pN;
69
70 if strcmp(method, 'wylie')
71     qN = zeros(nGridX,nGridY);
72     qO = pN;
73 end
74
75 % Set the analytical solution grid
76 pSol = pN;
77 uSol = uN;
78 vSol = vN;
79
80 %% Set initial conditions
81
82 for i = 1:nGridX
83     for j = 1:nGridY
84
85         pN(i,j) = p0;
86         uN(i,j) = u0;
87         vN(i,j) = v0;
88
89         if strcmp(method,'wylie')
90             qN(i,j) = 0;
```

```
91              end
92          end
93      end
94
95      pO = pN;
96      pSol = pN;
97
98      uO = uN;
99      uSol = uN;
100
101     vO = vN;
102     vSol = vN;
103
104     %% Movie options
105
106     if makeMovie
107         writerObj = VideoWriter(movieName);
108         writerObj.FrameRate = frameRate;
109         open(writerObj);
110     end
111
112     %% Plot figures
113
114     subplot(2,2,1)
115     mesh(X',Y',pN);
116     xlabel('X')
117     xlim([0 xLength]);
118     ylabel('Y')
119     ylim([0 yLength]);
120     zlim([0 1e6])
121     title(sprintf('Numerical, pressure at time %f',t))
122     if strcmp(viewpoint,'top')
123         view([90 90])
124     elseif strcmp(viewpoint,'side')
125         view([95 30])
126     end
127
128     subplot(2,2,2)
129     quiver(vN(1:nGridX,1:floor(nGridY/10):nGridY),uN(1:nGridX
            ,1:floor(nGridY/10):nGridY));
130     xlabel('Y')
131     ylabel('X')
132     zlim([-2 2])
133     title(sprintf('Numerical, velocity vector at time %f',t))
134
135     subplot(2,2,3)
```

```
136 avPPlot = plot(mean(pN,1));
137 title('Averaged pressure')
138 xlim([0 nGridY])
139 ylim([0 6e5])
140
141 subplot(2,2,4)
142 avVPlot = plot(mean(vN,1));
143 title('Averaged velocity')
144 xlim([0 nGridY])
145 ylim([-.5 .5])
146
147 pause(1)
148
149 %% Save to movie
150
151 if makeMovie
152     frame = getframe(gcf);
153     writeVideo(writerObj, frame);
154 end
155
156 %% Computational loop
157
158 while t < T
159
160     %% Update time
161
162     t = t + dT;
163
164
165     %% Compute interior points
166
167     for i = 2:nGridX-1
168         for j = 2:nGridY-1
169             if strcmp(method,'wylie')
170
171                 e1 = pO(i-1,j-1)/(rho*c) + uO(i-1,j-1) +
                        vO(i-1,j-1) + qO(i-1,j-1)*dX;
172                 e2 = pO(i-1,j+1)/(rho*c) + uO(i-1,j+1) -
                        vO(i-1,j+1) - qO(i-1,j+1)*dX;
173                 e3 = pO(i+1,j+1)/(rho*c) - uO(i+1,j+1) -
                        vO(i+1,j+1) + qO(i+1,j+1)*dX;
174                 e4 = pO(i+1,j-1)/(rho*c) - uO(i+1,j-1) +
                        vO(i+1,j-1) - qO(i+1,j-1)*dX;
175
176                 pN(i,j) = rho*c*(e1+e2+e3+e4)/4;
177                 uN(i,j) = (e1+e2-e3-e4)/4;
```

```
178             vN(i,j) = (e1-e2-e3+e4)/4;
179             qN(i,j) = (-e1+e2-e3+e4)/(4*dX);
180
181
182         elseif strcmp(method,'butler')
183
184             R = c*dT/dX;
185
186             A1 = [0, R*R/2, 0; R*R/2, 1-2*R*R, R*R/2;
                      0, R*R/2, 0];
187             B1 = [-c*rho*R*R*R/8, 0, c*rho*R*R*R/8; -
                    c*rho*R*(2-R*R)/4, 0, c*rho*R*(2-R*R)
                    /4    ; -c*rho*R*R*R/8, 0, c*rho*R*R*R
                    /8];
188             C1 = [c*rho*R*R*R/8, c*rho*R*(2-R*R)/4 ,
                    c*rho*R*R*R/8; 0, 0, 0; -c*rho*R*R*R
                    /8, -c*rho*R*(2-R*R)/4 , -c*rho*R*R*R
                    /8];
189
190             A2 = [0, 0, 0; -R/(2*c*rho), 0, R/(2*c*
                    rho); 0, 0, 0];
191             B2 = [0, 0, 0; R*R/2, 1-R*R, R*R/2; 0, 0,
                      0];
192             C2 = [-R*R/8, 0, R*R/8; 0, 0, 0; R*R/8,
                    0, -R*R/8];
193
194             A3 = [0, R/(2*c*rho), 0; 0, 0, 0; 0, -R
                    /(2*c*rho), 0];
195             B3 = [-R*R/8, 0, R*R/8; 0, 0, 0; R*R/8,
                    0, -R*R/8];
196             C3 = [0, R*R/2, 0; 0, 1-R*R, 0; 0, R*R/2,
                      0];
197
198             pN(i,j) = 0;
199             uN(i,j) = 0;
200             vN(i,j) = 0;
201
202             for k1 = 1:3
203                 for k2 = 1:3
204                     c1 = i + 2 - k1;
205                     c2 = j + 2 - k2;
206                     pN(i,j) = pN(i,j) + A1(4-k2,k1)*
                            pO(c1,c2) + B1(4-k2,k1)*uO(c1,
                            c2) + C1(4-k2,k1)*vO(c1,c2);
207                     uN(i,j) = uN(i,j) + A2(4-k2,k1)*
                            pO(c1,c2) + B2(4-k2,k1)*uO(c1,
```

```
                                    c2) + C2(4-k2,k1)*vO(c1,c2);
208                             vN(i,j) = vN(i,j) + A3(4-k2,k1)*
                                    pO(c1,c2) + B3(4-k2,k1)*uO(c1,
                                    c2) + C3(4-k2,k1)*vO(c1,c2);
209                         end
210                     end
211
212
213             end
214         end
215     end
216
217     %% Set boundary points
218
219     pN(1,1:nGridY) = pN(2,1:nGridY);
220     pN(nGridX,1:nGridY) = pN(nGridX-1,1:nGridY);
221     pN(1:nGridX,1) = p0;
222     pN(1:nGridX,nGridY) = pN(1:nGridX,nGridY-1);
223
224     uN(1,1:nGridY) = -uN(2,1:nGridY);
225     uN(nGridX,1:nGridY) = -uN(nGridX-1,1:nGridY);
226     uN(1:nGridX,1) = uN(1:nGridX,2);
227     uN(1:nGridX,nGridY) = uN(1:nGridX,nGridY-1);
228
229     vN(1,1:nGridY) = vN(2,1:nGridY);
230     vN(nGridX,1:nGridY) = vN(nGridX-1,1:nGridY);
231     vN(1:nGridX,1) = vN(1:nGridX,2);
232     vN(1:nGridX,nGridY) = -vN(1:nGridX,nGridY-1);
233
234     if strcmp(method,'wylie')
235         qN(1,1:nGridY) = -qN(2,1:nGridY);
236         qN(nGridX,1:nGridY) = -qN(nGridX-1,1:nGridY);
237         qN(1:nGridX,1) = -qN(1:nGridX,2);
238         qN(1:nGridX,nGridY) = -qN(1:nGridX,nGridY-1);
239     end
240
241     %% Update old values
242
243     pO = pN;
244     uO = uN;
245     vO = vN;
246
247     if strcmp(method,'wylie')
248         qO = qN;
249     end
250
```

```
251     %% Plot figures
252     draw = draw + 1;
253
254     if draw == drawLimit
255
256         draw = 0;
257
258         subplot(2,2,1)
259         mesh(X',Y',pN);
260         xlabel('X')
261         xlim([0 xLength]);
262         ylabel('Y')
263         ylim([0 yLength]);
264         zlim([0 1e6])
265         title(sprintf('Numerical, pressure at time %f',t)
                )
266         if strcmp(viewpoint,'top')
267             view([90 90])
268         elseif strcmp(viewpoint,'side')
269             view([95 30])
270         end
271
272         subplot(2,2,2)
273         quiver(vN(1:nGridX,1:floor(nGridY/10):nGridY),uN
                (1:nGridX,1:floor(nGridY/10):nGridY));
274         xlabel('Y')
275         ylabel('X')
276         zlim([-2 2])
277         title(sprintf('Numerical, velocity vector at time
                %f',t))
278
279         subplot(2,2,3)
280         avPPlot = plot(mean(pN,1));
281         title('Averaged pressure')
282         xlim([0 nGridY])
283         ylim([0 6e5])
284
285         subplot(2,2,4)
286         avVPlot = plot(mean(vN,1));
287         title('Averaged velocity')
288         xlim([0 nGridY])
289         ylim([-.5 .5])
290
291
292         drawnow
293
```

```
294             %% Save to movie
295
296             if makeMovie
297                 frame = getframe(gcf);
298                 writeVideo(writerObj, frame);
299             end
300         end
301     end
302
303     %% Draw graphs
304     subplot(2,2,1)
305     mesh(X',Y',pN);
306     xlabel('X')
307     xlim([0 xLength]);
308     ylabel('Y')
309     ylim([0 yLength]);
310     zlim([0 1e6])
311     title(sprintf('Numerical, pressure at time %f',t))
312     if strcmp(viewpoint,'top')
313         view([90 90])
314     elseif strcmp(viewpoint,'side')
315         view([95 30])
316     end
317
318     subplot(2,2,2)
319     quiver(vN(1:nGridX,1:floor(nGridY/10):nGridY),uN(1:nGridX
        ,1:floor(nGridY/10):nGridY));
320     xlabel('Y')
321     ylabel('X')
322     zlim([-2 2])
323     title(sprintf('Numerical, velocity vector at time %f',t))
324
325     subplot(2,2,3)
326     avPPlot = plot(mean(pN,1));
327     title('Averaged pressure')
328     xlim([0 nGridY])
329     ylim([0 6e5])
330
331     subplot(2,2,4)
332     avVPlot = plot(mean(vN,1));
333     title('Averaged velocity')
334     xlim([0 nGridY])
335     ylim([-.5 .5])
336
337
338     drawnow
```

```
339
340 %% Movie steps
341
342 if makeMovie
343     frame = getframe(gcf);
344     writeVideo(writerObj, frame);
345
346     close(writerObj);
347 end
```

A.6 Bicharacteristic.m

Main code for checking if the bicharacteristic method is truly unstable.

```
1  % Use the old method of characteristics to solve the
       linearized euler
2  % equations
3
4  clear;
5
6  %% Global variables
7  p0 = 1;
8  v0 = 0;
9
10 makeMovie = 0; % make a movie of the plots
11 movieName = 'drop.avi';
12 frameRate = 25;
13
14 draw = 0; % on each computation this is incremented by 1,
       by reaching drawLimit graph is plotted and set to
       zero
15 drawLimit = 1;
16
17 %% movie options
18 if makeMovie
19     writerObj = VideoWriter('bichar error.avi');
20     writerObj.FrameRate = 2;
21     open(writerObj);
22 end
23
24 %% Set constants and variables
25 xLength = 8;
26 yLength = 8;
27 totalTime = 0.0006;
28
29 dX = .1;
```

```
30 dY = dX;
31 t = 0;
32
33 rho = 1000; % Density
34 c = 1000; % Speed of sound
35 dT = .5*dX/c;
36
37 nGridX = ceil(xLength/dX)+1 + 2;
38 nGridY = ceil(yLength/dY)+1 + 2;
39
40 %% Set computational grid
41 x = -dX:dX:xLength+dX;
42 y = -dX:dY:yLength+dX;
43 [X,Y] = meshgrid(x,y);
44 X = X';
45 Y = Y';
46
47 pGrid = zeros(nGridX,nGridY);
48 pN = pGrid;
49
50 uGrid = zeros(nGridX,nGridY);
51 uN = uGrid;
52
53 vGrid = zeros(nGridX,nGridY);
54 vN = vGrid;
55
56 uxGrid = zeros(nGridX,nGridY);
57 uxN = uxGrid;
58
59 vyGrid = zeros(nGridX,nGridY);
60 vyN = vyGrid;
61
62 %% Set initial conditions
63
64 for i = 1:nGridX
65     for j = 1:nGridY
66         pGrid(i,j) = 0.5;
67         if (x(i)-xLength/2)^2 + (y(j)-yLength/2)^2 < 1
68             pGrid(i,j) = pGrid(i,j) + 0.5*exp(-1/(1-(x(i)
                -xLength/2)^2))*exp(-1/(1-(y(j)-yLength/2)
                ^2));
69         end
70     end
71 end
72
73 pN = pGrid;
```

```
74
75 %% Plot figures
76
77 % figure('units','normalized','outerposition',[0 0 1 1])
78 subplot(2,1,1)
79 pPlot = mesh(X,Y,pGrid);
80 xlabel('X')
81 xlim([0,xLength])
82 ylim([0,yLength])
83 ylabel('Y')
84 title(sprintf('Numerical, pressure at time %f',t))
85 zlim([0,1])
86
87 subplot(2,1,2)
88 velPlot = quiver(uGrid,vGrid);
89 xlabel('X')
90 ylabel('Y')
91 title(sprintf('Numerical, velocity vector at time %f',t))
92 xlim([-50 (nGridY + 50)])
93 ylim([-5 (nGridX + 5)])
94
95 if makeMovie
96     frame = getframe(gcf);
97     writeVideo(writerObj, frame);
98 end
99
100 pause(1)
101
102 %% Computational loop
103
104 xL = 2:nGridX-1;
105 yL = 2:nGridY-1;
106
107 x = -dX:dX:xLength+dX;
108 y = -dX:dY:yLength+dX;
109
110 while t < totalTime
111     t = t + dT
112
113     %% Compute analytical solution
114     for i = 1:nGridX
115         for j = 1:nGridY
116             [p1,u1,v1] = sol(x(i),y(j),t,xLength,yLength,
                    c,rho);
117
118             pSolution(i,j) = p1;
```

```
119             uSolution(i,j) = u1;
120             vSolution(i,j) = v1;
121         end
122     end
123
124     %% Compute interior points
125
126     for i = 2:nGridX-1
127         for j = 2:nGridX-1
128             x1 = x(i) - c*dT;
129             y1 = y(j);
130             e1 = interpol(x1,y1,pGrid((i-1:i+1),(j-1:j+1)
                    ),x(i-1),y(j-1),dX,dY) + c*rho*interpol(x1
                    ,y1,uGrid((i-1:i+1),(j-1:j+1)),x(i-1),y(j
                    -1),dX,dY) - .5*rho*c^2*dT*interpol(x1,y1,
                    vyGrid((i-1:i+1),(j-1:j+1)),x(i-1),y(j-1),
                    dX,dY);
131
132             x1 = x(i);
133             y1 = y(j) - c*dT;
134             e2 = interpol(x1,y1,pGrid((i-1:i+1),(j-1:j+1)
                    ),x(i-1),y(j-1),dX,dY) + c*rho*interpol(x1
                    ,y1,uGrid((i-1:i+1),(j-1:j+1)),x(i-1),y(j
                    -1),dX,dY) - .5*rho*c^2*dT*interpol(x1,y1,
                    uxGrid((i-1:i+1),(j-1:j+1)),x(i-1),y(j-1),
                    dX,dY);
135
136             x1 = x(i) + c*dT;
137             y1 = y(j);
138             e3 = interpol(x1,y1,pGrid((i-1:i+1),(j-1:j+1)
                    ),x(i-1),y(j-1),dX,dY) - c*rho*interpol(x1
                    ,y1,uGrid((i-1:i+1),(j-1:j+1)),x(i-1),y(j
                    -1),dX,dY) - .5*rho*c^2*dT*interpol(x1,y1,
                    vyGrid((i-1:i+1),(j-1:j+1)),x(i-1),y(j-1),
                    dX,dY);
139
140             x1 = x(i);
141             y1 = y(j) + c*dT;
142             e4 = interpol(x1,y1,pGrid((i-1:i+1),(j-1:j+1)
                    ),x(i-1),y(j-1),dX,dY) - c*rho*interpol(x1
                    ,y1,uGrid((i-1:i+1),(j-1:j+1)),x(i-1),y(j
                    -1),dX,dY) - .5*rho*c^2*dT*interpol(x1,y1,
                    uxGrid((i-1:i+1),(j-1:j+1)),x(i-1),y(j-1),
                    dX,dY);
143
144             x1 = x(i);
```

```
145             y1 = y(j);
146             e5 = interpol(x1,y1,pGrid((i-1:i+1),(j-1:j+1)
                    ),x(i-1),y(j-1),dX,dY) - .5*rho*c^2*dT*(
                    interpol(x1,y1,uxGrid((i-1:i+1),(j-1:j+1))
                    ,x(i-1),y(j-1),dX,dY) + interpol(x1,y1,
                    vyGrid((i-1:i+1),(j-1:j+1)),x(i-1),y(j-1),
                    dX,dY));
147
148             pN(i,j) = (e1+e2+e3+e4-2*e5)/2;
149             uN(i,j) = (e3-e1)/(2*c*rho);
150             vN(i,j) = (e4-e2)/(2*c*rho);
151             uxN(i,j) = -(e1+e3-2*e5)/(dT*rho*c^2);
152             vyN(i,j) = -(e2+e4-2*e5)/(dT*rho*c^2);
153
154         end
155     end
156
157     %% Compute boundary points
158     %% Set boundary points
159     pN(1:nGridX,1) = pN(1:nGridX,2);
160     pN(1:nGridX,nGridY) = pN(1:nGridX,nGridY-1);
161
162     pN(1,1:nGridY) = pN(2,1:nGridY);
163     pN(nGridX,1:nGridY) = pN(nGridX-1,1:nGridY);
164
165     uN(1:nGridX,1) = uN(1:nGridX,2);
166     uN(1:nGridX,nGridY) = uN(1:nGridX,nGridY-1);
167
168     uN(1,1:nGridY) = -uN(2,1:nGridY);
169     uN(nGridX,1:nGridY) = -uN(nGridX-1,1:nGridY);
170
171     vN(1:nGridX,1) = -vN(1:nGridX,2);
172     vN(1:nGridX,nGridY) = -vN(1:nGridX,nGridY-1);
173
174     vN(1,1:nGridY) = vN(2,1:nGridY);
175     vN(nGridX,1:nGridY) = vN(nGridX-1,1:nGridY);
176
177     pGrid = pN;
178     uGrid = uN;
179     vGrid = vN;
180     uxGrid = uxN;
181     vyGrid = vyN;
182
183     %% Plot graph
184     draw = draw + 1;
185
```

```
186         if draw == drawLimit
187             % figure('units','normalized','outerposition',[0
                    0 1 1])
188             subplot(2,1,1)
189             pPlot = mesh(X,Y,pGrid);
190             xlabel('X')
191             xlim([0,xLength])
192             ylim([0,yLength])
193             ylabel('Y')
194             title(sprintf('Numerical, pressure at time %f',t)
                    )
195             zlim([0,1])
196 
197             subplot(2,1,2)
198             velPlot = quiver(uGrid,vGrid);
199             xlabel('X')
200             ylabel('Y')
201             title(sprintf('Numerical, velocity vector at time
                    %f',t))
202             xlim([-50 (nGridY + 50)])
203             ylim([-5 (nGridX + 5)])
204 
205             if makeMovie
206                 frame = getframe(gcf);
207                 writeVideo(writerObj, frame);
208             end
209 
210             drawnow
211             draw = 0;
212         end
213     end
214 
215     % figure('units','normalized','outerposition',[0 0 1 1])
216     subplot(2,1,1)
217     pPlot = mesh(X,Y,pGrid);
218     xlabel('X')
219     xlim([0,xLength])
220     ylim([0,yLength])
221     ylabel('Y')
222     title(sprintf('Numerical, pressure at time %f',t))
223     zlim([0,1])
224 
225     subplot(2,1,2)
226     velPlot = quiver(uGrid,vGrid);
227     xlabel('X')
228     ylabel('Y')
```

```
229 title(sprintf('Numerical, velocity vector at time %f',t))
230 xlim([-50 (nGridY + 50)])
231 ylim([-5 (nGridX + 5)])
232
233 if makeMovie
234     frame = getframe(gcf);
235     writeVideo(writerObj, frame);
236
237     close(writerObj);
238 end
239
240 %% Functions used
241
242 % Bi parabolic interpolation
243 % Interpolates 9 points in 2D space
244 % A is a 3x3 matrix
245 function result = interpol(x,y,A,x0,y0,dX,dY)
246     a00 = A(1,1);
247     a10 = .5*(-3*A(1,1) + 4*A(2,1)- A(3,1));
248     a01 = .5*(-3*A(1,1) + 4*A(1,2) - A(1,3));
249     a11 = .25*(9*A(1,1) - 12*A(1,2) + 3*A(1,3) - 12*A
          (2,1) + 16*A(2,2) - 4*A(2,3) + 3*A(3,1) - 4*A(3,2)
           + A(3,3));
250     a20 = .5*(A(1,1) - 2*A(2,1) + A(3,1));
251     a02 = .5*(A(1,1) - 2*A(1,2) + A(1,3));
252     a21 = .25*(-3*A(1,1) + 4*A(1,2) - A(1,3) + 6*A(2,1) -
           8*A(2,2) + 2*A(2,3) - 3*A(3,1) + 4*A(3,2) - A
          (3,3));
253     a12 = .25*(-3*A(1,1) + 6*A(1,2) - 3*A(1,3) + 4*A(2,1)
           - 8*A(2,2) + 4*A(2,3) - A(3,1) + 2*A(3,2) - A
          (3,3));
254     a22 = .25*(A(1,1) - 2*A(1,2) + A(1,3) - 2*A(2,1) + 4*
         A(2,2) - 2*A(2,3) + A(3,1) - 2*A(3,2) + A(3,3));
255
256     xC = (x-x0)/dX;
257     yC = (y-y0)/dY;
258
259     result = a00 + a10*xC + a01*yC + a11*xC*yC + a20*xC^2
           + a02*yC^2 + a12*xC*yC^2 + a21*yC*xC^2 + a22*(xC*
         yC)^2;
260 end
```

A.7 Other Matlab files

In this subsection the Matlab files used in the main files are stated.
error-loglog.m

```
function error_loglog(h,E)

% Produce log-log plot of E vs h
% Estimate order of accuracy by doing a linear least
    squares fit

h = h(:);               % make sure it's a column vector
E = E(:);               % make sure it's a column vector
ntest = length(h);
clf
loglog(h,E,'o-')
axis([.5*min(h) 1.5*max(h)  .5*min(E) 1.5*max(E)])
title('log-log plot of errors vs. h')

% Estimate order of accuracy from least squares fit:
Ap = ones(ntest,2);
Ap(:,2) = log(h);
bp = log(E);
Kp = Ap\bp;
K = Kp(1);
p = Kp(2);
disp(' ')
disp(sprintf('Least squares fit gives E(h) = %g * h^%g',
    exp(K),p))
disp(' ')

% add graph of this line to loglog plot:
hold on
err1 = exp(K)*h.^p;
loglog(h,err1,'r')
legend('errors', 'least squares fit','Location','
    SouthEast')
xlabel('h')
ylabel('Error')
hold off
```

error-table.m

```
function error_table(h,E)
%
```

```
% Print out table of errors, ratios, and observed order
    of accuracy.
%
% From  http://www.amath.washington.edu/~rjl/fdmbook/
    (2007)

ntest = length(h);
ratio = nan(size(h));   % initialize storage
order = nan(size(h));   % initialize storage

for j=2:ntest
    ratio(j) = E(j-1)/E(j);
    order(j) = log(abs(ratio(j))) / log(abs(h(j-1)/h(j)));
    end


% print out table:

disp(' ')
disp('      h         error        ratio        observed
    order')
for j=1:ntest
    disp(sprintf(' %9.5f  %12.5e %9.5f %15.5f',h(j),E(j),
        ratio(j),order(j)));
    end
disp(' ')
```

norm2.m

```
% 2 norm, compute the 1 norm |A-B|, where A and B are the
    same size
% matrices
function n = norm2(A,B,dX,dY)
    [s1, s2] = size(A); % get the size of A

    total = 0;
    for i = 1:s1
        for j = 1:s2
            total = total + dX*dY*abs(A(i,j)-B(i,j))^2;
        end
    end

    n = sqrt(total);
end
```

qSolution.m

```
% Analytical solution to q:
function q = qSolution(x,y,t,a,b,c,rho)

    mu = pi/a;
    nu = pi/b;

    lambda = c*sqrt(mu^2+nu^2);

    q = -mu*nu*cos(mu*x)*cos(nu*y)*sin(lambda*t)/(lambda*
        rho);
end
```

sol.m

```
% Analytical solution
% Initial condition p = sin(pi x/a) * sin(pi y/b)
function [p,u,v] = sol(x,y,t,a,b,c,rho)
    mu = pi/a;
    nu = pi/b;
    lambda = c*sqrt(mu^2+nu^2);

    p = sin(mu*x)*sin(nu*y)*cos(lambda*t);
    u = -mu*cos(mu*x)*sin(nu*y)*sin(lambda*t)/(lambda*rho
        );
    v = -nu*sin(mu*x)*cos(nu*y)*sin(lambda*t)/(lambda*rho
        );
end
```