Υπολογιστική ρευστομηχανική
7^o εξάμηνο

Προαιρετικό θέμα 2

Επίλυση συμπιεστής ροής εντός αγωγού μεταβλητής διατομής

Γεώργιος Νιδριώτης - mc18054

6 Φεβρουαρίου 2024

Περιεχόμενα

Εισαγωγή

Η παρούσα εργασία στοχεύει στην ανάπτυξη λογισμικού που λύνει τις μονοδιάστατες εξισώσεις Euler για μοντελοποίηση της ροής συμπιεστής ροής εντός αγωγού μεταβλητής διατομής. Θεωρούμε πως στο αριστερό μέρος του αγωγού έχουμε συνδεδεμένο αεροφυλάκιο σταθερών και γνωστών συνθηκων. Η ροή μελετάται ως μη μόνιμη και παρουσιάζεται το μεταβατικό της στάδιο απο την εκκίνησή της ανοίγωντας το αεροφυλάκιο έως την αποκατάσταση της μόνιμης ροής.

Οι εξισώσεις Euler παρουσιάζονται στην πρωταρχική τους μορφή (1) και την συντηρητική τους μορφή (2) που βοηθά στην επίλυσή τους. Η πρωταρχική μορφή μας παρέχει τις ιδιοτιμές της Ιακωβιανής οι οποίες είναι αναγκαίες για να αναγνωρίσουμε τη ροή της πληροφορίας και τον υπολογισμό των παροχών όπως θα δούμε παρακάτω.

$$\frac{\partial}{\partial t}\begin{pmatrix}\rho\\ u\\ p\end{pmatrix} + \begin{pmatrix}u & \rho & 0\\ 0 & u & 1/\rho\\ 0 & \rho c^2 & u\end{pmatrix}\cdot\frac{\partial}{\partial x}\begin{pmatrix}\rho\\ u\\ p\end{pmatrix} = \begin{pmatrix}-\dfrac{\rho u}{S}\dfrac{dS}{dx}\\ 0\\ -\dfrac{\rho u c^2}{S}\dfrac{dS}{dx}\end{pmatrix} \qquad \textbf{Πρωταρχική μορφή} \tag{1}$$

$$\frac{\partial}{\partial t}\begin{pmatrix}\tilde{\rho}\\ \tilde{\rho}u\\ \tilde{\rho}E\end{pmatrix} + \frac{\partial}{\partial x}\begin{pmatrix}\tilde{\rho}u\\ \tilde{\rho}u^2+\tilde{p}\\ \tilde{\rho}Hu\end{pmatrix} = \begin{pmatrix}0\\ \dfrac{\tilde{p}}{S}\dfrac{dS}{dx}\\ 0\end{pmatrix} \qquad \textbf{Συντηρητική μορφή} \tag{2}$$

Όπου,

- $\widetilde{\rho} = \rho\mathbf{S}$
- $\mathbf{\widetilde{p} = pS}$

Όπως αναφέραμε και παραπάνω, θα λύσουμε την συντηρητική μορφή της εξίσωσης του Euler που επιτρέπει την άμεση ολοκλήρωσή της, και η μεθοδολογία που ακολουθήσαμε για την επίλυση αναλύεται στην παρακάτω παράγραφο.

1 Μεθοδολογία επίλυσης

Ολοκληρώνωντας την εξίσωση 2 και γράφοντάς τη σε διακριτή μορφή (προσαρμόζοντας στο πλέγμα) αποκτάμε την παρακάτω μορφή (εξ 1.1).

$$\Delta x\frac{\partial \overline{U}_i}{\partial t} + (F_{i+1/2} - F_{i-1/2}) = \Delta x \cdot \overline{Q}_i \tag{1.1}$$

Όπου $\overline{U}_i$ και $\overline{Q}_i$ είναι η μέση τιμή των συντηρητικών μεταβλητών και των όρων πηγής αντίστοιχα σε ένα κελλί. Επομένως, η κεντρική ιδέα της μεθόδου είναι πως χρησιμοποιώντας την διακριτοποιημένη έκφραση της 1.1, εκτελούμε σχήμα ρητής ολοκλήρωσης Runge Kutta 4ης τάξης για τον υπολογισμό της χρονικής εξέλιξης της ροής (παράγραφος 1.2).

Οι τιμές των παροχών στα σύνορα (faces) των κελλιών ($F_{i+1/2}, F_{i-1/2}$), υπολογίζονται με τη βοήθεια του σχήματος ανακατασκευής του Roe (παράγραφος 1.1).

1.1 Σχήμα ανακατασκευής του Roe

Το σχήμα του Roe υπολογίζει τις παροχές στις επιφάνειες των κελλιών μέσω της σχέσης 1.2. Πρακτικά υπολογίζει τον μέσο όρο των παροχών που προκύπτουν απο το αριστερό και το δεξί γειτονικό κελλι, και διορθώνει την τιμή λαμβάνωντας υπόψη, μέσω των ιδιοτιμών, την κατεύθυνση που μεταφέρεται η πληροφορία.

$$F_f = \frac{1}{2}(F_L + F_R) - \frac{1}{2}|A|\cdot(U_R - U_L) \tag{1.2}$$

Όπου,

- $\mathbf{F_L}, \mathbf{F_R}$, οι τιμές της παροχής ($\tilde{\rho}u, \tilde{\rho}u^2+\tilde{p}, \tilde{\rho}Hu$) χρησιμοποιώντας τη μέση τιμή του αντίστοιχου συντηρητικού μεγέθους (σχήμα 1ης τάξης) για το αντίστοιχο κελί (αριστερό ή δεξί)

- $|\mathbf{A}| = \mathbf{R}|\mathbf{\Lambda}|\mathbf{R^{-1}}$, με $|\Lambda|$ τον διαγώνιο πίνακα που έχει τις απόλυτες τιμές των ιδιοτιμών στη διαγώνιό του
- $\mathbf{U_R} - \mathbf{U_L}$, το διάνυσμα της διαφοράς των συντηρητικών μεταβλητών του δεξιού μείον του αριστερού κελλιού

Για τον υπολογισμό του μητρώου $|A|$ χρησιμοποιούμε μια έκφραση των τιμών που αποτελεί ένα σταθμισμένο μέσο όρο των τιμών του αριστερού και του δεξιού κελλιού (εξ 1.3).

$$\begin{aligned}
\tilde{\rho} &= \sqrt{\rho_L} \cdot \sqrt{\rho_R} \\
\tilde{u} &= \frac{\sqrt{\rho_L} \cdot u_L + \sqrt{\rho_R} \cdot u_R}{\sqrt{\rho_L} + \sqrt{\rho_R}} \\
\tilde{H} &= \frac{\sqrt{\rho_L} \cdot H_L + \sqrt{\rho_R} \cdot H_R}{\sqrt{\rho_L} + \sqrt{\rho_R}} \\
\tilde{c} &= \sqrt{(\gamma - 1)(\tilde{H} - 0.5\tilde{u}^2)}
\end{aligned} \tag{1.3}$$

Με $H = E + pv = E + \dfrac{p}{\rho} = \dfrac{\frac{\tilde{\rho}E}{S} + p}{\rho}$

Οπότε, με βάση τα μεγέθη των σχέσεων 1.3, οι ιδιοτιμές και το μητρώο R είναι:

$$\begin{aligned}
|\lambda| &= \begin{bmatrix} |\tilde{u}| & 0 & 0 \\ 0 & |\tilde{u} + \tilde{c}| & 0 \\ 0 & 0 & |\tilde{u} - \tilde{c}| \end{bmatrix} \\
R &= \begin{bmatrix} 1 & 0.5\tilde{\rho}/\tilde{c} & -0.5\tilde{\rho}/\tilde{c} \\ \tilde{u} & 0.5(\tilde{u} + \tilde{c})\tilde{\rho}/\tilde{c} & -0.5(\tilde{u} - \tilde{c})\tilde{\rho}/\tilde{c} \\ 0.5\tilde{u}^2 & 0.5 \cdot (0.5\tilde{u}^2 + \tilde{u}\tilde{c} + \dfrac{\tilde{c}^2}{\gamma - 1})\tilde{\rho}/\tilde{c} & -0.5 \cdot (0.5\tilde{u}^2 - \tilde{u}\tilde{c} + \dfrac{\tilde{c}^2}{\gamma - 1})\tilde{\rho}/\tilde{c} \end{bmatrix}
\end{aligned} \tag{1.4}$$

Τέλος, για την αντιμετώπιση ασταθειών στην περίπτωση κρουστικού κύματος, διορθώνουμε τις ιδιοτιμές ως εξής:

$$\begin{aligned}
\delta &= 0.05 \cdot \tilde{c}^2 \\
|\lambda| &= \frac{|\lambda|^2 + \delta^2}{2\delta} \text{ εαν } |\lambda| \leq \delta
\end{aligned}$$

Έτσι, μέσω των παραπάνω, υπολογίζουμε τις παροχές σε κάθε επιφάνεια του πλέγματός μας. Το αντίστοιχο τμήμα του κώδικα που πραγματοποιεί τον υπολογισμό των παροχών παρατίθεται στο παράρτημα.

1.2 Σχήμα ολοκλήρωσης Runge Kutta

Η αριθμητική επίλυση της συντηρητικής δ.ε λύνεται με το σχήμα Runge Kutta 4ης τάξης. Αρχικά, η εξίσωση 1.1 γράφεται ως:

$$f(U_i, t) = \frac{\partial \overline{U}_i}{\partial t} = \overline{Q}_i - \frac{F_{i+1/2} - F_{i-1/2}}{\Delta x}$$

Και το διάνυσμα των συντηρητικών μεταβλητών για τον κόμβο i και για κάθε βήμα του αλγορίθμου k είναι:

$$U_i^k = U_{t-1} + dt \cdot C_k \cdot f(U_i^{k-1}, t - 1) \tag{1.5}$$

Η τιμή της επόμενης χρονικής στιγμής t είναι η U_4 δηλαδή για k=4. Οι τιμές του συντελεστή C_k είναι: $C_k = [0.1084, 0.2602, 0.5052, 1]$. Οι τιμές που υπολογίζονται στα ενδιάμεσα βήματα συνεισφέρουν αποκλειστικά στη νέα τιμή της $f(U_i, t)$ παρά μόνο την περίπτωση του τελευταίου βήματος, όπου ανανεώνεται η τιμή του διανύσματος για το επόμενο χρονικό βήμα.

1.3 Ορισμός οριακών συνθηκών

Αρχικά, αναφέρεται πως έχουμε 2 εικονικά κελλιά (ghost cells) τα οποία βρίσκονται μπροστά απο το πρώτο κελλί του χωρίου και μετά το τελευταίο. Σε αυτά τα κελλιά ορίζουμε τις οριακές συνθήκες εισόδου και εξόδου και το σχήμα ανακατασκευής του Roe είναι υπεύθυνο να ανανεώσει τις τιμές των παροχών στο πρώτο και το τελευταίο face με βάση τις τιμές στα ghost cells.

Εκτός απο τις τιμές στα ghost cells που καθορίζονται απο τα δεδομένα στο αεροφυλάκιο, αρχικοποιήσαμε τις τιμές των κελλιών ως εξής: Θεωρώντας πως το αεροφυλάκιο είναι απομονωμένο απο τον αγωγό πρίν απο τη στιγμή t=0, όλα τα κελλιά πλήν του πρώτου ghost cell, αρχικοποιούνται με τις τιμές του περιβάλλοντος (εξόδου) για την πυκνότητα και την πίεση και με μηδενική ταχύτητα.

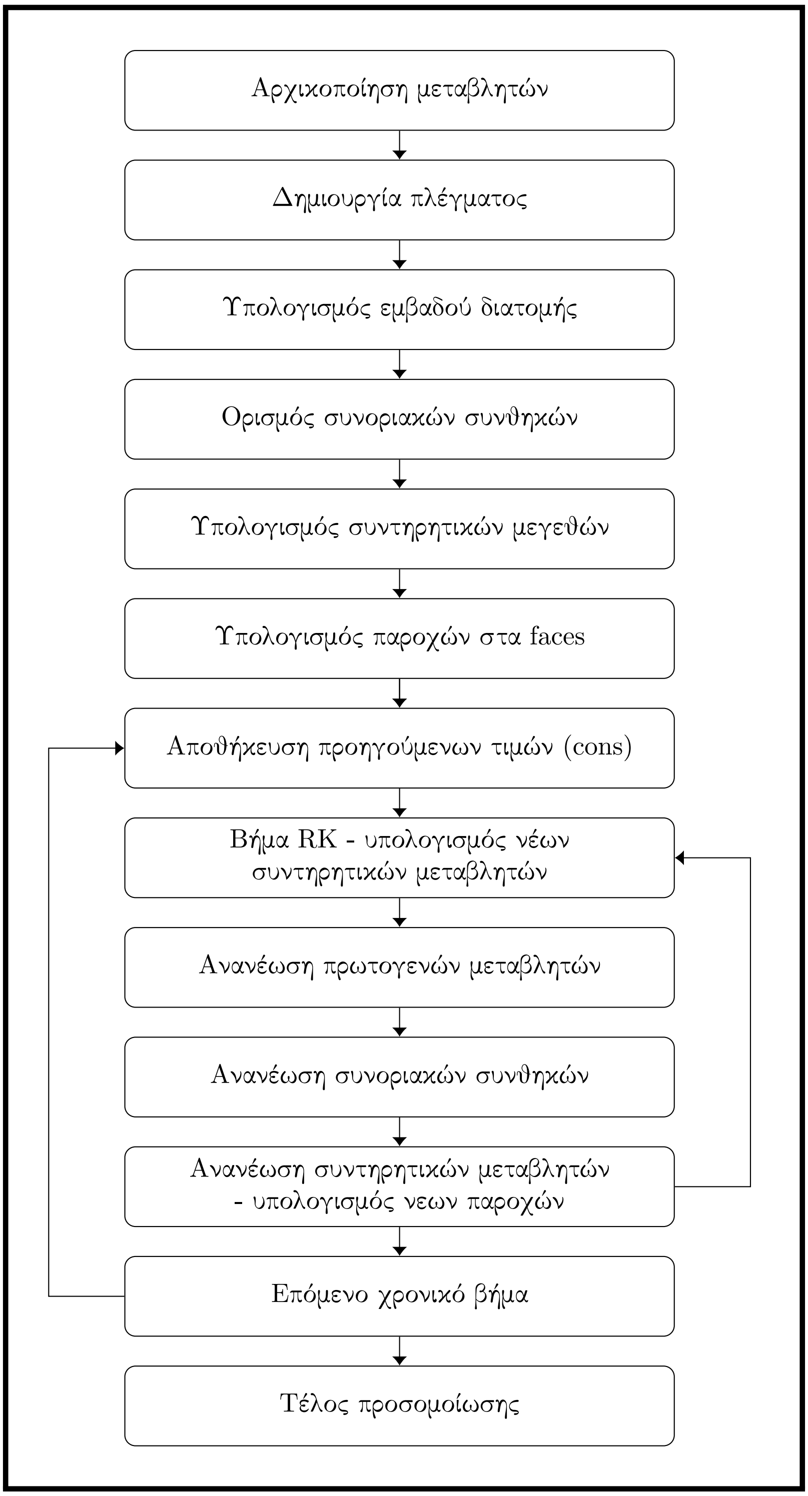
Επιπλέον, για τις τιμές που δεν έχουμε δεδομένες σε είσοδο και έξοδο, (ταχύτητα σε είσοδο έξοδο, πυκνότητα στην έξοδο), οι τιμές στα ghost cells ανανεώνονται ωστε να είναι ίσες με τις τιμές του γειτονικού κελλιού (μηδενική κλίση).

Τέλος, αναφέρεται πως προφανώς όλες οι οριακές συνθήκες ορίσθηκαν για τις πρωτογενείς μεταβλητές.

Οι ακριβείς τιμές ακολουθώντας τα προσωπικά δεδομένα παρατίθενται παρακάτω στον πίνακα 1.

1.4 Αλγόριθμος επίλυσης

Ο αλγόριθμος επίλυσης υλοποιώντας τις μεθόδους που περιγράφηκαν παραπάνω φαίνεται σχηματικά στο σχήμα 1.



Σχήμα 1: Αλγόριθμος επίλυσης μοντέλου

2 Παρουσίαση προσωπικών δεδομένων εργασίας

Τα δεδομένα που αφορούν γεωμετρικές παραμέτρους και φυσικά μεγέθη παρατίθενται στον πίνακα 1.

k	0.09
a	0.216
b	1.03
c	0.97
$p_{αεροφ}$	4 bar
$p_{εξ}$	3.3 bar
$T_{αεροφ}$	286K
Μήκος αγωγού	2m

Πίνακας 1: Δεδομένα εργασίας

Η τιμή της πυκνότητας για τις συνοριακές συνθήκες και για την αρχικοποίηση υπολογίζονται ως $\rho = \dfrac{p}{RT}$. Αντίστοιχα, οι αριθμητικές παράμετροι που χρησιμοποιήθηκαν παρατίθενται στον πίνακα 2.

Αριθμός κόμβων	801
Χρονικό βήμα	$10^{-6}s$
Αριθμός βημάτων	45000
Τελικός χρόνος	0.045s

Πίνακας 2: Αριθμητικές παράμετροι

Η τάξη μεγέθους του χρονικού βήματος επιλέχθηκε ακολουθώντας το όριο του Courant. Ο αριθμός του Courant είναι:

$$C = \frac{Udt}{dx}$$

Ο αριθμός Courant θέλουμε να είναι τουλάχιστον μικρότερος της μονάδας, αλλά συνήθως θέλουμε να είναι αρκετά μικρότερος. Λύνωντας ως προς dt έχουμε:

$$dt = \frac{Cdx}{U}$$

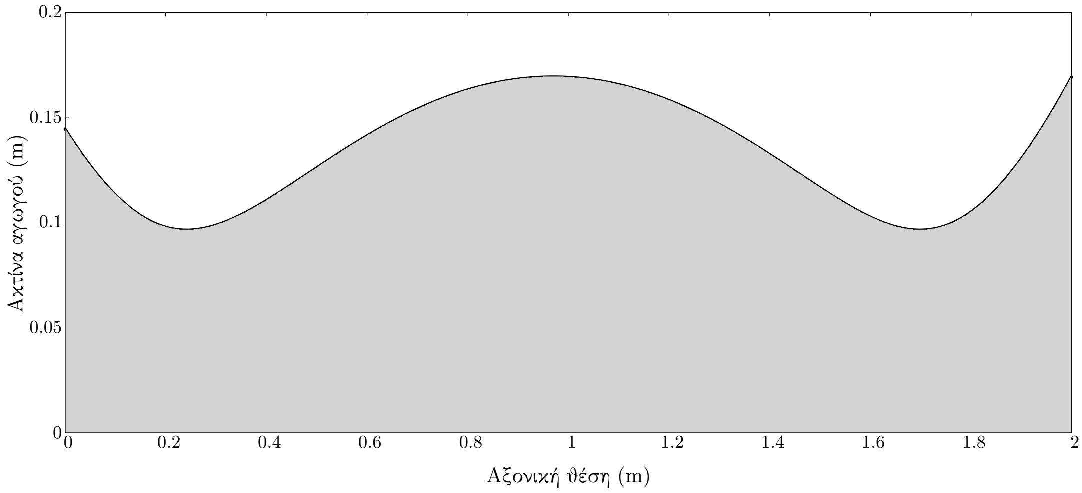
Αν θεωρήσουμε πως η ταχύτητα μπορεί να φτάσει περίπου 2 φορές την ταχύτητα του ήχου ($\approx 600m$) και αριθμό Courant C $= 0.5$, τότε προκύπτει οτι θέλουμε το χρονικό βήμα να είναι περίπου τρεις τάξεις μεγέθους μικρότερο απο το χωρικό βήμα. Έτσι ακολουθήσαμε αυτό τον κανόνα για την επιλογή του χρονικού βήματος.

Ο αριθμός των κόμβων (αντίστοιχα μήκος κελλιού) επιλέχθηκε με δοκιμές, ξεκινώντας απο μικρό αριθμό αυξάνοντας σταδιακά έως να ευσταθήσει η σύγκλιση του κώδικα.

2.1 Γεωμετρία αγωγού

[h!]

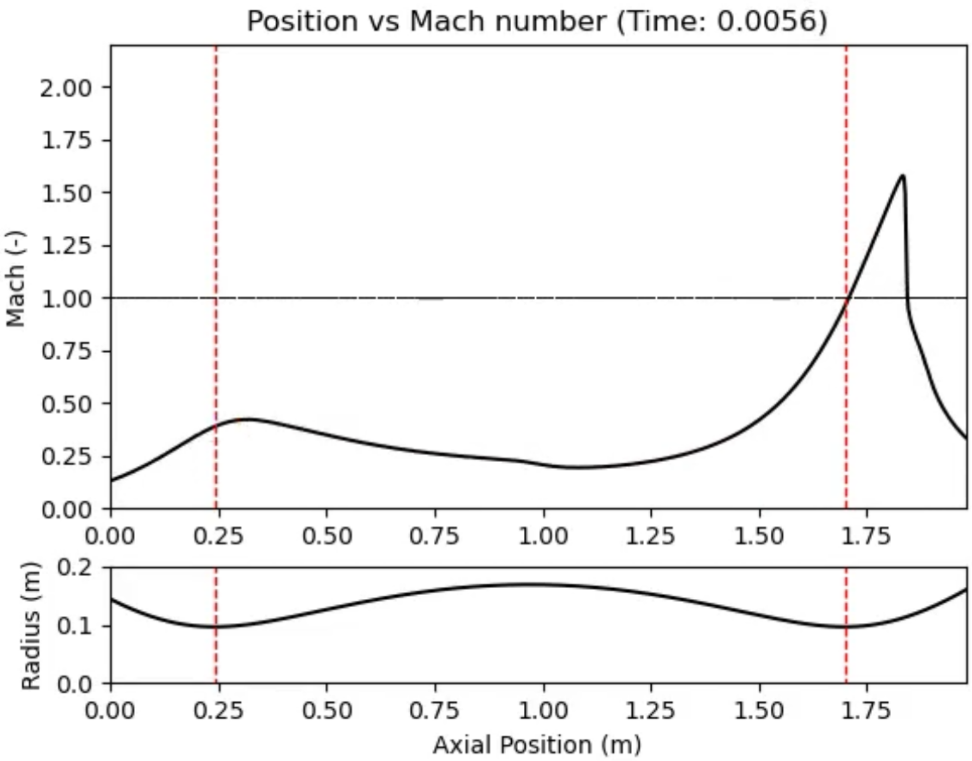
Με βάση τα αριθμητικά δεδομένα που παρουσιάστηκαν στον πίνακα 1, η γεωμετρία που προκύπτει φαίνεται στο σχήμα 2.



Σχήμα 2: Διάγραμμα μεταβολής ακτίνας αγωγού

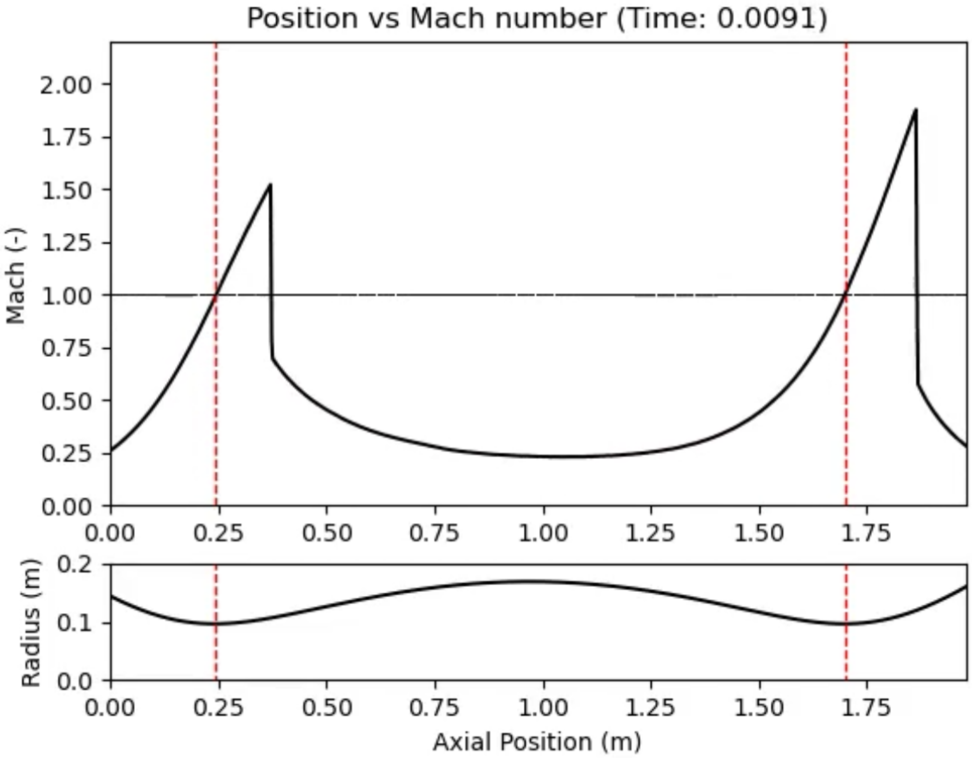
3 Αποτελέσματα και σχολιασμός

Στα παρακάτω σχήματα φαίνονται χαρακτηριστικές στιγμές στην εξέλιξη της ροής. Αρχικά, παραθέτουμε τη στιγμή που σχηματίζεται το πρώτο κύμα κρούσης, έπειτα το δεύτερο και τέλος την αποκατάσταση της ροής.



Σχήμα 3: Σχηματισμός πρώτου κύματος κρούσης

Απο τα στιγμιότυπα και το βίντεο της εξέλιξης της ροής παρατηρούμε πως όταν ανοίγει το αεροφυλάκιο η ροή ξεκινά απο το αεροφυλάκιο και προχωρά προς την έξοδο σαν ενα κύμα. Φαίνεται επίσης σα να αποκαθίσταται η ροή απο την έξοδο προς την είσοδο, και σχηματίζεται πρώτα το κύμα κρούσης στον δεύτερο λαιμό (κοντά στην έξοδο)

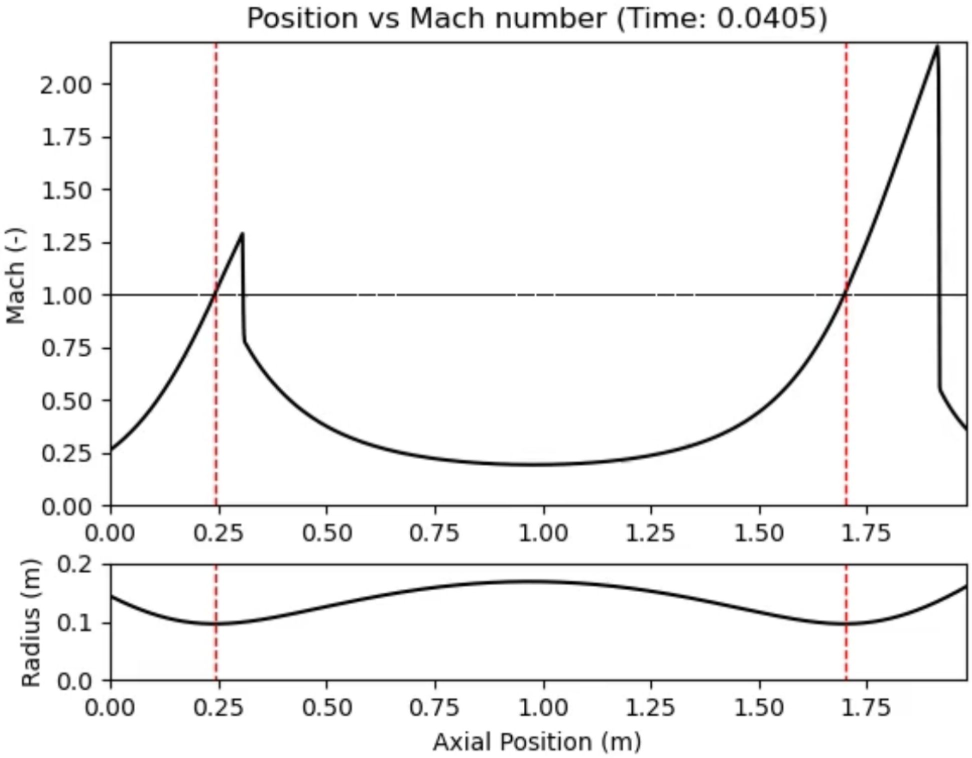


Σχήμα 4: Σχηματισμός δεύτερου κύματος κρούσης

όπως φαίνεται και στο σχήμα 3 και όταν σταθεροποιείται η ροή κοντά στον δεύτερο λαιμό, αρχίζει να αποκαθίσταται και στα πίσω τμήματα. Πράγματι, στη συνέχεια εμφανίζεται και δεύτερο κύμα κρούσης μετά τον πρώτο λαιμό και η ροή αρχίζει αργά να αποκαθίσταται.

Επιπλέον, απο όλες τις στιγμές των στιγμιότυπων στα διαγράμματα του αριθμού Mach, παρατηρούμε πως στα σημεία των λαιμών έχουμε αριθμό Mach ίσο με τη μονάδα, που συμφωνεί με τα αναμενόμενα βάσει της θεωρίας.

Τέλος, βλέπουμε πως στην αποκατεστημένη ροή μετά τον δεύτερο λαιμό έχουμε πολύ μεγαλύτερο κύμα κρούσης όπου ο αριθμός Ma ξεπερνά το 2, και απο υπερηχητική η ροή μεταβαίνει σε υποηχητική για να μεταβεί στις συνθήκες εξόδου, όπως φαίνεται και στα αντίστοιχα βίντεο με τα διαγράμματα πίεσης και πυκνότητας.



Σχήμα 5: Αποκατάσταση μόνιμης ροής

I Πηγαίος κώδικας

Το λογισμικό που λύνει το μοντέλο υλοποιήθηκε σε γλώσσα C++. Η δομή του κώδικα αποτελείται απο τα εξής. Ενα κύριο αρχείο (main.cpp) με την κύρια συνάρτηση του προγράμματος που περιλαμβάνει την κεντρική δομή του αλγορίθμου, και ένα δευτερεύον βοηθητικό αρχείο (utils.cpp) που περιέχει τους ορισμούς των συναρτήσεων που καλούνται στην κεντρική δομή του αλγορίθμου. Τέλος, έχουμε ένα αρχείο που περιέχει τις γεωμετρικές παραμέτρους του προβλήματος και τις αριθμητικές παραμέτρους του κώδικα. Η δομή του main.cpp ακολουθεί τη ροή του αλγορίθμου που παρουσιάζεται στο σχήμα 1. Έτσι, παρακάτω παρατίθενται τα αρχεία με την σειρά που αναφέρθηκαν.

I.i Κύρια συνάρτηση αλγορίθμου main.cpp

```
#include "utils.h"
#include "data.h"

using namespace std;

// Define global variables
std::vector<double> x;
std::vector<double> S;
std::vector<double> U1_p;
std::vector<double> U2_p;
std::vector<double> U3_p;
std::vector<double> U1_c;
std::vector<double> U2_c;
std::vector<double> U3_c;
std::vector<double> F1;
std::vector<double> F2;
std::vector<double> F3;
int cells = nodes+1;
std::vector<double> dS;
double dx;


int main()
{
    initialize(); // Initialize variables
    discretize(); // Create grid points
    calcS(); // Calculate values of section area for each node position

    updateBC(); // Assign primitive values to ghost cells
    calcCons(); // Calculate conservative variables for all cells

    roeFlux(); // Calculate face fluxes using Roe's scheme

    std::vector<double> rkConst(4);
    rkConst[0] = 0.1084; rkConst[1] = 0.2602; rkConst[2] = 0.5052; rkConst[3] = 1;

    // Initialize vectors to hold variables of
    // previous timestep
    std::vector<double> Uold1(cells);
    std::vector<double> Uold2(cells);
    std::vector<double> Uold3(cells);

    for (int it = 1; it <= maxIter; it ++)
    {
        Uold1 = U1_c;
        Uold2 = U2_c;
        Uold3 = U3_c;

        for (int rk = 0; rk<4; rk++)
        {
            // Calculate conservative variables for
            // the next RK step for all (non-ghost) cells
            for (int cl=1; cl<cells-1; cl++)
            {
                U1_c[cl] = Uold1[cl] - dt*rkConst[rk]* (F1[cl]-F1[cl-1])/dx;
                U2_c[cl] = Uold2[cl] + dt*rkConst[rk]* (U3_p[cl]*(dS[cl-1]+dS[cl])/2 -(F2[cl]-F2[cl-1])/dx) ;
                U3_c[cl] = Uold3[cl] - dt*rkConst[rk]* (F3[cl]-F3[cl-1])/dx;
```

```
            }
            // Re-calculate values for next RK step
            calcPrim();
            updateBC();
            calcCons();
            roeFlux();

        }
        // Handle result writing
        std::cout << it << " " << it*dt << std::endl;
        if (it%writeEvery == 0)
            writeResults(it);
    }

}
```

I.ii Βοηθητικό αρχείο συναρτήσεων utils.cpp

```
#include "utils.h"
#include "data.h"
#include <fstream>
#include <string>

// Declare global variables
extern std::vector<double> x;
extern std::vector<double> S;
extern std::vector<double> U1_p;
extern std::vector<double> U2_p;
extern std::vector<double> U3_p;
extern std::vector<double> U1_c;
extern std::vector<double> U2_c;
extern std::vector<double> U3_c;
extern std::vector<double> F1;
extern std::vector<double> F2;
extern std::vector<double> F3;
extern int cells;
extern std::vector<double> dS;
extern double dx;

void initialize()
{
    // Allocate memory for all vectors
    x.resize(nodes);
    S.resize(nodes);
    dS.resize(nodes);

    F1.resize(nodes);
    F2.resize(nodes);
    F3.resize(nodes);

    U1_p.resize(cells);
    U2_p.resize(cells);
    U3_p.resize(cells);
    U1_c.resize(cells);
    U2_c.resize(cells);
    U3_c.resize(cells);

```

```
    // Initialize all vectors with their respective values
    std::fill(U1_p.begin(), U1_p.end(), densOut);
    std::fill(U2_p.begin(), U2_p.end(), 0);
    std::fill(U3_p.begin(), U3_p.end(), presOut);

}
void discretize() {
    dx = tubeLength/(nodes-1);
    for (int i =0; i<nodes; i++)
    {
        x[i] = i*dx;
    }
}

void calcS()
{
    double d = 0;
    double x_s = 0;
    int size = (int) S.size();
    std::ofstream sOut("results/S.dat");

    for (int i = 0; i<size; i ++)
    {
        x_s = x[i] - cc;
        // Calculate section area
        S[i] = k + a*pow(x_s,2)*(pow(x_s,2)-pow(b,2));
        // Calculate diameter
        d = sqrt(4*S[i]/pi);
        // Save area derivative
        dS[i] = 2 * a * x_s *( pow(x_s,2)-pow(b,2) ) + 2 * a * pow(x_s,3);
        // File output
        sOut << x[i] << " " << d << " " << S[i] << std::endl;
    }
    sOut.close();
}

void calcCons()
{

    double Sm = 1; // Mean area of cell

    for (int i = 0; i < cells; i ++)
    {
        // Interpolate area of adjacent nodes
        // as the mean area of the cell
        if (i!=0 && i!= cells-1)
        {
            Sm = 0.5 * (S[i-1]+S[i]);
        }else if (i==0) // If first ghost cell take the area of the first node
        {
            Sm = S[0];
        }else // If last ghost -> area of last node
        {
            Sm = S[nodes];
        }

        U1_c[i] = U1_p[i]*Sm;
        U2_c[i] = U2_p[i]*U1_p[i]*Sm;
        U3_c[i] = ( U3_p[i]/Gamma1 + 0.5 * U1_p[i] * pow( U2_p[i], 2) ) * Sm;
    }

}

void calcPrim()
{
    double Sm = 1; // Mean area of cell

    for (int i = 0; i < cells; i ++)
    {
        // Interpolate area of adjacent nodes
        // as the mean area of the cell
        if (i!=0 && i!= cells-1)
        {
```

```
            Sm = 0.5 * (S[i-1]+S[i]);
        }else if (i==0)
        {
            Sm = S[0];
        }else
        {
            Sm = S[nodes];
        }

        U1_p[i] = U1_c[i]/Sm;
        U2_p[i] = U2_c[i]/U1_c[i];
        U3_p[i] = Gamma1 * ( U3_c[i] - 0.5*pow(U2_c[i] ,2) / U1_c[i] )/Sm;
    }

}

void calcFlux()
{
    std::vector<double> FL(3), FR(3);
    double VN_L, VN_R, NORM_L, NORM_R;
    NORM_L = 1;
    NORM_R = 1;
    for (int i=0; i<nodes; i++)
    {
        VN_L = NORM_L*U2_p[i];
        VN_R = NORM_R*U2_p[i+1];
        FL[0] = U1_p[i]*VN_L*S[i];
        FL[1] = (U1_p[i] * U2_p[i] * VN_L + NORM_L*U3_p[i])*S[i];
        FL[2] = (U3_c[i]/S[i] + U3_p[i]) * VN_L * S[i];

        FR[0] = U1_p[i+1]*VN_R*S[i];
        FR[1] = (U1_p[i+1] * U2_p[i+1] * VN_R + NORM_R*U3_p[i+1])*S[i];
        FR[2] = (U3_c[i+1]/S[i] + U3_p[i+1]) * VN_R * S[i];

        F1[i] = 0.5*(FL[0]+FR[0]);
        F2[i] = 0.5*(FL[1]+FR[1]);
        F3[i] = 0.5*(FL[2]+FR[2]);
    }
}

// Assemble left matrix
Eigen::Matrix<double, 3, 3> asmLeft(double U, double dens, double C)
{
    Eigen::Matrix<double, 3, 3> left;

    left(0,0) = 1 - 0.5 * Gamma1 * pow(U,2)/pow(C,2);
    left(0,1) = Gamma1 * U / pow(C,2);
    left(0,2) = - Gamma1 / pow(C,2);

    left(1,0) = (0.5 * Gamma1 * pow(U,2) - U*C) * 1/(dens*C);
    left(1,1) = (C - Gamma1 * U)/(dens*C);
    left(1,2) = Gamma1/(dens*C);

    left(2,0) = - (0.5 * Gamma1 * pow(U,2) + U*C)/(dens*C);
    left(2,1) = (C+Gamma1*U)/(dens*C);
    left(2,2) = - Gamma1/(dens*C);

    return left;
}

// Assemble right matrix
Eigen::Matrix<double, 3, 3> asmRight(double U, double dens, double C)
{

    Eigen::Matrix<double, 3, 3> right;

    right(0,0) = 1;
    right(0,1) = 0.5*dens/C;
    right(0,2) = -0.5*dens/C;

    right(1,0) = U;
    right(1,1) = 0.5*(U+C)*dens/C;
    right(1,2) = -0.5*(U-C)*dens/C;
```

```
    right(2,0) = 0.5*pow(U,2);
    right(2,1) = (0.5 * pow(U,2) + U*C + pow(C,2)/Gamma1)*0.5*dens/C;
    right(2,2) = -(0.5 * pow(U,2) - U*C + pow(C,2)/Gamma1)*0.5*dens/C;

    return right;
}

void roeFlux()
{
    calcFlux();
    double r,u,HL,HR,H,c,delta;
    Eigen::Matrix<double, 3, 3> right, left, lambda, alpha;
    Eigen::Matrix<double, 3, 1> ULR;
    Eigen::Matrix<double, 3, 1> roeCor;
    right.setZero(); left.setZero(); lambda.setZero(); ULR.setZero(); alpha.setZero();

    for (int i = 0; i<nodes; i++)
    {
        r = pow(U1_p[i],0.5)*pow(U1_p[i+1],0.5);
        u = ( pow(U1_p[i],0.5)*U2_p[i] + pow(U1_p[i+1],0.5)*U2_p[i+1] )
            /(pow(U1_p[i],0.5) + pow(U1_p[i+1],0.5));
        HL = (U3_c[i]/S[i]+U3_p[i])/U1_p[i];
        HR = (U3_c[i+1]/S[i]+U3_p[i+1])/U1_p[i+1];
        H = (pow(U1_p[i],0.5)*HL + pow(U1_p[i+1],0.5)*HR )
            /(pow(U1_p[i],0.5) + pow(U1_p[i+1],0.5));
        c = sqrt(Gamma1*(H-0.5*pow(u,2)));


        delta = 0.05*c;

        // Assemble right and left matrices
        left = asmLeft(u, r, c);
        right = asmRight(u, r, c);

        // Perform corrections to handle shock waves
        lambda(0,0) = abs(u);
        if (lambda(0,0)<=delta)
            lambda(0,0) = (pow(lambda(0,0),2)+pow(delta,2))/(2*delta);

        lambda(1,1) = abs(u+c);
        if (lambda(1,1)<=delta)
            lambda(1,1) = (pow(lambda(1,1),2)+pow(delta,2))/(2*delta);

        lambda(2,2) = abs(u-c);
        if (lambda(2,2)<=delta)
            lambda(2,2) = (pow(lambda(2,2),2)+pow(delta,2))/(2*delta);

        // Difference of right - left conservative vars vector
        ULR(0,0) = U1_c[i+1]-U1_c[i];
        ULR(1,0) = U2_c[i+1]-U2_c[i];
        ULR(2,0) = U3_c[i+1]-U3_c[i];

        // Calc A matrix
        alpha = right*lambda*left;
        roeCor = alpha*ULR;

        // Calculate final fluxes for face
        F1[i] = F1[i] - 0.5*roeCor(0,0);
        F2[i] = F2[i] - 0.5*roeCor(1,0);
        F3[i] = F3[i] - 0.5*roeCor(2,0);

    }

}

void updateBC()
{
    // Inflow BC
    U1_p[0] = densIn;
    U2_p[0] = U2_p[1];
    U3_p[0] = presIn;
```

```
    // Outflow BC
    U1_p.back() = U1_p[cells-2];
    U2_p.back() = U2_p[cells-2];
    U3_p.back() = presOut;
}


void writeResults(int iter)
{
    std::string primName = "primitiveVars";
    std::string extension = ".dat";
    std::string filepath = "results/";
    std::string prFileName = filepath.append(primName).append(std::to_string(iter)).append(
    extension);
    std::ofstream outVar(prFileName);


    outVar << "Time Iteration: " << iter <<std::endl;
    outVar << "Time: " << (iter)*dt <<std::endl;
    outVar << std::setw(25) << "Position"
            << std::setw(30) << "Density"
            << std::setw(30) << "Velocity"
            << std::setw(30) << "Speed of sound"
            << std::setw(30) << "Mach"
            << std::setw(30) << "Pressure" << std::endl;

    double xm;
    for (int j=0; j<cells; j++)
    {
        if (j!=0 && j!=cells-1)
        {
            xm = (x[j-1]+x[j])/2;
        }else if (j==0)
        {
            xm = -(x[0]+x[1])/2;
        }else
            xm = -(x[nodes-1]+x[nodes-1])/2 + x[nodes-1];

        double u_c = sqrt(Gamma*U3_p[j]/(U1_p[j]));
        outVar << std::setw(25) << xm
            << std::setw(30) << U1_p[j]
            << std::setw(30) << U2_p[j]
            << std::setw(30) << u_c // Sound speed
            << std::setw(30) << U2_p[j]/u_c // Mach number
            << std::setw(30) << U3_p[j] << std::endl;
    }
    outVar.close();
}
```

I.iii Αρχείο παραμέτρων data.h

```
#pragma once

// Physical properties
#define Gamma 1.4
#define Gamma1 0.4
#define R 287 // J/(kgK)
#define presIn 400000.0 //4 bar to Pa
#define presOut 330000.0 // Pa
#define tempIn 286.0 // K
#define densIn presIn/(R*tempIn) // kg/m3
#define densOut presOut/(R*tempIn) // kg/m3
```

```c

// Simulation parameters
#define nodes 801
#define dt 0.000001
#define writeEvery 100
#define maxIter 45000

// Geometry parameters
#define tubeLength 2.0
#define a 0.216
#define b 1.03
#define cc 0.97
#define k 0.09
```