Aufgabenstellung

Es ist die Strömung durch eine Lavaldüse mit Hilfe **eines** der folgenden drei zentralen numerischen Verfahren zu lösen:

1: MacCormack

2: Lax-Wendroff

3: Central Method

Das Ergebnis soll mit der analytischen Lösung aus dem beiliegenden Excel-File verglichen werden.

Folgende Ergebnis-Files werden erstellt:

Mach.out: Mach-Zahl entlang der Düse

Ptot.out: Total-Druck entlang der Düse

Press.out: Statischer Druck entlang der Düse

Cont.out: Dimensionsloser Massenstrom entlang der Düse

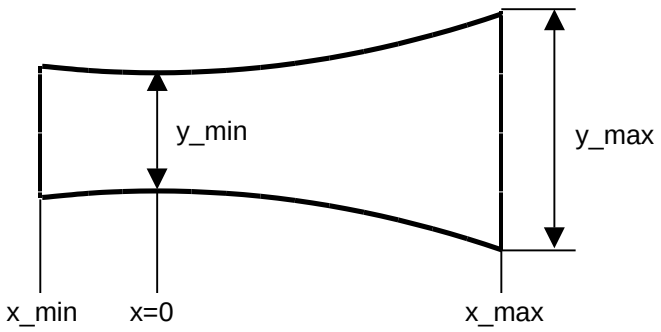
Nozzle.out: die 3 Größen des Zustandsvektors und am Ende die „vergangene“ Zeit

Die Randbedingungen und Steuerparameter sind im File nozzle_3.dat zu finden, der vom Programm eingelesen wird.

Geometriedefiniton:

Die Kontur der Lavaldüse (= Querschnittsfläche) konstanter Breite 1 ist durch folgende Beziehung gegeben, sodass der engste Querschnitt y_{min} sich bei x=0 befindet:

$$A(x) = (y_{\min} + (y_{\max} - y_{\min}) \frac{x^2}{x_{\max}^2}) * 1$$



Task

The flow in a Laval nozzle has to be calculated with **one** of the following central methods:

1: MacCormack

2: Lax-Wendroff

3: Central Method

The numerical result shall be compared with the analytical solutiont oft he attached Excel file.

The code will generate following result files:

Mach.out: Mach number along the nozzle axis

Ptot.out: Total pressure along the nozzle axis

Press.out: Static pressure along the nozzle axis

Cont.out: Non-dimensional mass flow along the nozzle axis

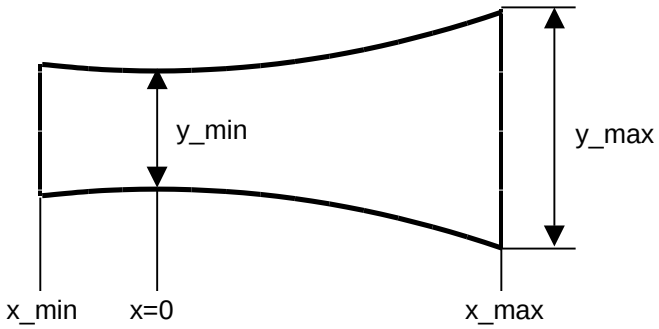
Nozzle.out: contains the three variables of the state vector and at the end to „past“ time span

The boundary conditions and the control parameters are listed in the input file nozzle_3.dat, which is read by the code.

Geometry:

The contour (=area) of the Laval nozzle of constant width 1 is given by following simple relation; the minimal flow cross section y_{min} is at x=0:

$$A(x) = (y_{\min} + (y_{\max} - y_{\min}) \frac{x^2}{x_{\max}^2}) * 1$$



Variablendefinition

area	Fläche
cfl	CFL-Zahl
convergence	Konvergenzlimit
da_dx	1. Ableitung der Fläche nach x
delta_u	ΔU-Vektor
dissip	Dissipations-Vektor
dt	Zeitschrittgröße
dx	Netzweite
end	=1: Konvergenzkriterium erreicht
f	Flussvektor
f_star	Stern-Flussvektor
gamma	Isentropenkoeffizient
cp	spez. Wärmekapazität
eps_s	Parameter für „einfache“ Dissipation
imax	Anzahl der Knotenpunkte
iread	=0: Neustart, =1: Fortsetzung
itr	Index des Iterationsschrittes
k2	2nd Order – Parameter für „komplexe“ Dissipation
k4	4th Order – Parameter für „komplexe“ Dissipation
max_iter	Anzahl der Iterationsschritte
method	Verfahrensauswahl (1=MCC, 2=LW, 3=Central)
nprint	Schreiben des Residuums bei jedem nprint-Schritt
p_exit	statischer Druck beim Austritt
p_tot	Totaldruck beim Eintritt
R	Gaskonstante
resid1	Gesamtresiduum der 1. Komponente beim 1. Zeitschritt
resid2	Gesamtresiduum der 2. Komponente beim 1. Zeitschritt
resid3	Gesamtresiduum der 3. Komponente beim 1. Zeitschritt
rho_tot	Totaldichte beim Eintritt
source	Quellterm
sub_exit	=1: Unterschallaustritt, sonst Überschallaustritt
time	Gesamtzeit
T_tot	Totaltemperatur beim Eintritt
u	Zustandsvektor
u_q	U_quer- Zustandsvektor

u_qq	U_quer_quer-Zustandsvektor
x	x-Koordinate
x_max	Endkoordinate des Netzes
x_min	Anfangskoordinate des Netzes
y_max	y-Koordinate (Querschnitt) bei x_max
y_min	kleinste y-Koordinate (engster Querschnitt) bei x=0

Definition of Variables

area	Area
cfl	CFL number
convergence	Convergence limit
da_dx	1st derivative of flow area along the x-axis
delta_u	ΔU, change in state ´vector
dissip	Dissipation vector
dt	time step size
dx	grid spacing
end	=1: convergence achieved
f	Flux vector
f_star	Star flux vector
gamma	Isentropic coefficient (ratio of specific heats)
cp	specific heat
eps_s	Parameter for „simple“ dissipation
imax	Number of grid points
iread	=0: new start, =1: continue with calculation
itr	Index of iteration step
k2	2nd Order parameter for „complex“ dissipation
k4	4th Order parameter for „complex“ dissipation
max_iter	Maximum number of iteration steps
method	Method parameter (1=MCC, 2=LW, 3=Central)
nprint	Write a residuum at every nprint step
p_exit	Static pressure at the exit
p_tot	Total pressure at the inlet
R	Gas constant
resid1	Total residuum of the 1^{st} state vector component at 1^{st} time step
resid2	Total residuum of the 2^{nd} state vector component at 1^{st} time step
resid3	Total residuum of the 3^{rd} state vector component at 1^{st} time step
rho_tot	Total density at the inlet
source	Source term (vector)
sub_exit	=1: subsonic outflow, else supersonic outflow
time	Total time
T_tot	Total temperature at the inlet
u	State vector
u_q	U_quer state vector (MCC)

u_qq	U_quer_quer state vector (MCC)
x	x coordinate (nozzle axis)
x_max	x coordinate of nozzle end
x_min	x coordinate of nozzle start
y_max	y coordinate (flow area) at x_max
y_min	Smallest y-coordinate (flow area) at x=0

```
Input file for 1d-nozzle solver
Laval nozzle flow
---------------------------------------------------------------
C.. IREAD   ITER     NPRINT    CONV
      0      100000         10    1.e-4
C.. imax    x_min[m]   x_max[m]  y_min[m]   y_max[m]
     101     -0.333333    1.        0.1       0.2
C..  R      kappa
    287.0    1.4
C.. p_tot[bar]  t_tot[K]   p_exit[bar]   subsonic exit
      1.0         273.15       0.9            1
C..  Method (MCC=1, LW=2, Central=3)      simple Dissip (yes=1)
        3                                       0
C..  CFL     eps_s       k4     k2
     0.35     10000.0    8.0    200.0
```

```
/*      program CENTRAL
c**********************************************************************
c
c                            1D CFD code for
cftt
c                              students of
c                       CFD in turbomachinery
c
c
c     solution of the 1-D unsteady Euler equations for nozzle flows
c     with a time-iterative central method
c
c     numerical algorithm:
c     - MacCormack algorithm
c         - Lax-Wendroff algorithm
c         - explicit central method with simple and "sophisticated" 4th order
c           artifical dissipation
c
c         Remarks to central method: CFL < 0.1
c         values of eps, k2, k4 Parameter are important
c         good results for eps=50, k4=0.04,k2=1
c         (if k2 is too small, Dissip4 is not switched off)
c
c**********************************************************************/

/*------------------ SUBROUTINES ------------------------*/
void init();
void grid();
void input();
void boundary();
void boundary_q();
void calc_uq();
void calc_uqq();
void calc_f();
void dissip_simple();
void dissip_complex();
void calc_f_star_LW();
void calc_f_star_central();
int conv(int itr);
void output();
void timestep();

/*------------------ DECLARATIONS ------------------------*/
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

#ifndef max
#define max(a,b) (((a) > (b)) ? (a) : (b))
#endif

#ifndef fabs
```

```
#define fabs(a) (((a) < (0.0)) ? (-(a)) : (a))
#endif

/* max. dimension of the arrays*/
#define Mat_dim 1001

/*------------------------ INPUT VARIABLES----------------------------*/
int     iread;                          /*0:std, 1:read from input file */
int     max_iter;                       /*max number of iterations*/
int     nprint;                         /*print at xxth iteration*/
double  convergence;            /*convergence limit*/
int     imax;                           /*number of cells*/
double  x_min;                          /*[m]*/
double  x_max;                          /*[m]*/
double  y_min;                          /*[m] for laval nozzle*/
double  y_max;                          /*[m] for laval nozzle*/
double  R;                                      /*R*/
double  gamma;                          /*kappa*/
double  p_tot;                          /*total inlet pressure [bar]*/
double  p_exit;                         /*static exit pressure [bar]*/
double  T_tot;                          /*total inlet temperature [K]*/
int             sub_exit;                       /*sub/supersonic conditions*/
int             method;                         /*=1: MCC, =2: LW, =3:central*/
int             simple_dissip;          /*1: simple dissipation, else
sophisticated dissipation*/
double  cfl;                            /*courant number   <0 => transient*/
double  eps_s;                          // Parameter of simple dissipation
double  k4;                                     // 4th order parameter for
complex dissipation
double  k2;                                     // 2nd order parameter for
complex dissipation
/*-------------------------   END INPUT VARIABLES  ----------------------*/

/*------------------------ STATIC VARIABLES----------------------------*/
/*Files*/
FILE *logfile;
/*control*/
double dt;
double dx ;
double time ;
/*geometry*/
double area[Mat_dim] = {0.0};
double da_dx[Mat_dim] = {0.0};
/*vektor*/
double u[Mat_dim][4] = {0.0};
double u_q[Mat_dim][4] = {0.0};
double u_qq[Mat_dim][4] = {0.0};
double dissip[Mat_dim][4] = {0.0};
double delta_u[Mat_dim][4] = {0.0};
double f[Mat_dim][4] = {0.0};
double f_star[Mat_dim][4] = {0.0};
double source[Mat_dim][4] = {0.0};
```

```
/*boundary*/
double rho_tot;
double h_tot;
/*x_coordinaten*/
double x[Mat_dim] = {0.0};
/*residuum*/
double resid1;
double resid2;
double resid3;
/*------------------------- END STATIC VARIABLES--------------------------*/


void main()
{
        int itr, end=0,i,k;

        input();

        grid();

        init();

        //----------------loop start---------------------------------------
        for (itr=1; itr<=max_iter; itr++)
        {
        timestep();

        switch (method)
                {
                        case 1: //MCC

                                calc_uq();

                                boundary_q();

                                calc_uqq();

                                for (i=2; i<=imax-1; i++)
                                {
                                        for (k=1;k<=3;k++)
                                        {
                                                delta_u[i][k] = 0.5*(u_q[i][k]
+ u_qq[i][k])-u[i][k];
                                                u[i][k] = u[i][k] +
delta_u[i][k];
                                        }
                                }

                                break;

                        case 2: //LW
```

```
                        calc_f();

                calc_f_star_LW();

                        for (i=2; i<=imax-1; i++)
                        {
                                for (k=1;k<=3;k++)
                                {
                                        delta_u[i][k] = ...  //
Calculation of delta_u using the conservative star fluxes
                                        u[i][k] = u[i][k] +
delta_u[i][k];
                                }
                        }

                        break;

                case 3: //central

                        calc_f();

                        if (simple_dissip == 1)
                                dissip_simple();
                        else
                                dissip_complex();

                calc_f_star_central();

                        for (i=2; i<=imax-1; i++)
                        {
                                for (k=1;k<=3;k++)
                                {
                                        delta_u[i][k] =
-dt*(f_star[i][k] - f_star[i-1][k])/dx + dt*source[i][k];
                                        u[i][k] = u[i][k] +
delta_u[i][k];
                                }
                        }

                        break;

                default:

                        printf("\n No numerical method was selected
(1-3)!!!\n");
                        exit(0);

        }


boundary();
```

```
        if((itr%nprint == 0) || (itr == max_iter))              end =
conv(itr);

        if (end == 1)
                {
                        printf("\n Convergence limit of %lf achieved!
\n",convergence);
                        printf("\n Number of iterations: %d \n",itr);
                        break;
                }
        }

        //---------------------loop
end---------------------------------------




        output();

        fclose(logfile);

        return;
}



/*--------------------------data input-----------------------------------*/
void input()
{
        FILE *input;
        char  dummy[81];

        input=fopen("nozzle_3.dat","rt");
        fgets(dummy,80,input);
        fgets(dummy,80,input);
        fgets(dummy,80,input);
        fgets(dummy,80,input);
        fscanf(input,"%d%d%d%lf\n",&iread,&max_iter,&nprint,&convergence);
        fgets(dummy,80,input);
        fscanf(input,"%d%lf%lf%lf%lf\n",&imax,&x_min,&x_max,&y_min,&y_max);
        fgets(dummy,80,input);
        fscanf(input,"%lf%lf\n",&R,&gamma);
        fgets(dummy,80,input);
        fscanf(input,"%lf%lf%lf%d\n",&p_tot,&T_tot,&p_exit,&sub_exit);
        fgets(dummy,80,input);
        fscanf(input,"%d%d\n",&method, &simple_dissip);
        fgets(dummy,80,input);
        fscanf(input,"%lf%lf%lf%lf\n",&cfl,&eps_s, &k4, &k2);

        if (imax-1 > Mat_dim)
```

```
        {
                printf("** Specified number of cells is higher than allocated
vector size ! **\n");
        }

        p_tot=p_tot*100000.0;
        p_exit=p_exit*100000.0;
        h_tot = gamma/(gamma-1)*R*T_tot;
}


/*-------------------------grid definition
-----------------------------------*/
void grid()
{
        int i;
        // Berechnen von dx, x[i], area[i], da_dx[i]
        // Calculation of dx, x[i], area[i], da_dx[i]

}


/*-------------------------initializing----------------------------------*/
void init()
{
    FILE *old_data;
        int i;

        // Berechnen von rho_tot, am Eintritt fuer die Randbedingungen
        // Calculaton of rho_tot at the inlet for the boundary condition
algorithm

        if (iread == 0)
        {
                // Initialisieren des Stroemungsfeldes (Zustandsvektor U) mit
den Ruhezustandswerten
                // Initialisation of the flow field (state vector U) with the
stagnation values (=total values)

        }
        else
        {
                old_data = fopen("nozzle.out","r");

                for(i=1; i<=imax; i++)

fscanf(old_data,"%lf%lf%lf\n",&u[i][1],&u[i][2],&u[i][3]);

                fclose(old_data);
        }
```

```
    logfile = fopen("nozzle.log","w");
        fprintf (logfile, "%s\n", " Iter cont_resid imp_resid energy_resid");
}


//-------------------------timestep
calculation-------------------------------------
void timestep()
{
        int i;
        double eigenmax,vel,p,c,eigen;

    /*
Bestimmen des maximalen Eigenwertes eigenmax fuer das gesamte Stroemungsfeld
         eigen = max(fabs(vel+c),fabs(vel-c))
Calculation of the maximum eigenvalue eigenmax for the total flow field
         eigen = max(fabs(vel+c),fabs(vel-c))

Bestimmen von dt als Funktion von cfl und max. Eigenwert
Find dt as function of cfl and maximium eigenvalue

*/
        time = time + dt;
}

//-----------------------flux and source
vector-----------------------------------------
void calc_f()
{
    double rho,vel,p, temp;
        int i;

        //Berechnung des des Flussvektors F und des Source-Vektors in allen
Punkten
        //Calculaton of flux vector F and source vector S in all grid points

}

//------------------------simple dissipation
vector-----------------------------------
void dissip_simple()
{
        int i,k;

        // dissipation vector at i+1/2


        // dissipation vector at i=1 and i=imax-1


}
```

```
//----------------------complex dissipation
vector-----------------------------------------
void dissip_complex()
{
        int i,k;
        double rho,vel,c,p,pm,pp;
        double eigen, eigenp;
        double eps2,eps4;
        double sensor[Mat_dim];

        // dissipation vector at i+1/2


        // dissipation vector for i=1 and i=imax-1


}


//---------------------------f_star
central----------------------------------------
void calc_f_star_central()
{
        int i,k;

        // calculation of f_star at i+1/2 for central method


}

//--------------------------f_star
Lax-Wendroff-----------------------------------------
void calc_f_star_LW()
{
        int i,k;

        // calculation of f_star at i+1/2 for Lax-Wendroff method

}

//--------------------------MCC U_q
vector-----------------------------------------
void calc_uq()
{
    double rho,vel,p;
        int i;

        /*
        Berechnung des Flussvektors F und des Source-Vektors in allen Punkten
fuer U
        Bestimmung von U_q (forward)
```

```
        Calculaton of flux vector F and source vector S in all grid points for
U
        Calculate U_q (forward)
        */
}

//----------------------------MCC_U_qq
vector-----------------------------------------
void calc_uqq()
{
    double rho,vel,p;
        int i;

        /*
        Berechnung des des Flussvektors F und des Source-Vektors in allen
Punkten fuer U_q
        Bestimmung von U_qq (backward)

        Calculaton of flux vector F and source vector S in all grid points for
U_q
        Calculate U_qq (backward)
        */
}

//-------------------------------U_q boundary
conditions----------------------------------
void boundary_q()
{
    double rho,p,vel;

//      Bestimmen der Randwerte fuer i=1 und i=imax fuer U_q-Vektor
//      Calculation of boundary values for i=1 and i=imax for U_q vector


        /*inlet i=1*/

        if (rho > rho_tot)      rho = rho_tot;


        /*outlet i=imax*/

}

//----------------------------U boundary
conditions-----------------------------------------
void boundary()
{
    double rho,p,vel,temp;

        //      Bestimmen der Randwerte fuer i=1 und i=imax fuer U-Vektor
        //      Calculation of boundary values for i=1 and i=imax for U vector
```

```
        /*inlet i=1*/

        if (rho > rho_tot)      rho = rho_tot;


        /*outlet i=imax*/
}

//---------------------------------------------------------------------
int conv(int itr)
{
        int i,k,end;
        double resid[4] = {0.0};

        if (itr == 1) return;

        printf("calc timestep = %d\t", itr);


        for(i=2; i<=imax-1; i++)
        {
        for (k=1;k<=3;k++)
                {
                        resid[k] = resid[k] + fabs(delta_u[i][k]);
                }
        }



        if ((itr == nprint) || (itr==2))
        {
                resid1 = resid[1];
                resid2 = resid[2];
                resid3 = resid[3];
        }

        fprintf(logfile,"%d %lf %lf %lf\n",itr, resid[1]/resid1,
resid[2]/resid2, resid[3]/resid3);

        if (resid[3]/resid3 < convergence) end = 1;

        printf("resid = %lf\n", resid[3]/resid3);

    return end;
}



//---------------------------------------------------------------------
void output()
```

```
{
    int i;
        double temp,cont0, rho,vel;
        double p[Mat_dim], mach[Mat_dim],p_tot_is[Mat_dim], cont[Mat_dim];
        FILE *result,*machzahl,*pressure,*continuity,*total_pressure;

    result = fopen("nozzle.out","wt");
        machzahl = fopen("mach.out","wt");
        pressure = fopen("press.out","wt");
        continuity = fopen("cont.out","wt");
        total_pressure = fopen("ptot.out","wt");

        for (i=1;i<=imax;i++)
fprintf(result,"%lf\t%lf\t%lf\n",u[i][1],u[i][2],u[i][3]);
        fprintf(result,"\n %lf \n", time);

        for (i=1; i<=imax; i++)
        {
        rho = u[i][1];
        vel = u[i][2]/rho;
        p[i] = (gamma-1)*(u[i][3]-rho*vel*vel/2);
        mach[i] = vel/pow((gamma*p[i]/rho),0.5);
        temp = 1+(gamma-1)/2*mach[i]*mach[i];
        p_tot_is[i] = p[i]*pow(temp,(gamma/(gamma-1)));
        if (i == 1)
                {
            cont0 = rho*vel*area[i];
            if(cont0<=0.) cont0 = 1.e-5;
                }
        cont[i] = rho*vel*area[i]/cont0;
        }


    for (i=1;i<=imax;i++) fprintf(pressure,"%lf\t%lf\n", x[i], p[i]/1.e5);
    for (i=1;i<=imax;i++) fprintf(machzahl,"%lf\t%lf\n", x[i], mach[i]);
    for (i=1;i<=imax;i++) fprintf(continuity,"%lf\t%lf\n", x[i], cont[i]);
    for (i=1;i<=imax;i++) fprintf(total_pressure,"%lf\t%lf\n", x[i],
p_tot_is[i]/p_tot);

    fclose (pressure);
    fclose (machzahl);
    fclose (continuity);
    fclose (total_pressure);
    fclose (result);
}
```