Averaging theory for heat transfer in circular hydraulic jumps with a separation bubble – Software documentation

R. Solana Gómez[1]†, T. Bohr[2], S. Nielsen[2], W. Rohlfs[3], R. Kneer[1] and H. Askarizadeh[1]

[1]Institute of Heat and Mass Transfer, RWTH Aachen University, Augustinerbach 6, 52056 Aachen, Germany

[2]Department of Physics and Center for Fluid Dynamics, Technical University of Denmark, DK-2800 Kgs. Lyngby, Denmark

[3]Department of Thermal and Fluid Engineering, University of Twente, Drienerlolaan 5, 7522 NB Enschede, The Netherlands

1. Preface

This document describes the code for the solution of the averaging theory for heat transfer (ATHT) in circular hydraulic jumps with a separation bubble proposed by Solana Gómez *et al.* (2024) and instructions on how to use it. It is a program consisting of multiple Matlab scripts. The code was developed by me, Rafael Solana Gómez, and is partially based on a previous code developed by Simon Møller Nielsen for the solution of the averaging theory for the flow (AT). The code was developed during my work on my master's thesis at the Institute of Heat and Mass Transfer of the RWTH Aachen University and also during my PhD studies at the same institute. The master's thesis was supervised by Hossein Askarizadeh Ravizi, with whom I developed the extension of the AT to the ATHT. Special thanks go to Thomas Bohr for providing the code developed by Simon Møller Nielsen and his precious advice during our work on the hydraulic jump.

2. Description and instructions

Please make sure to add all folders and subfolders to your path for all features to work properly. If you use the code in your work, please reference the code and the theoretical investigations it is based on (Bohr *et al.* 1997; Watanabe *et al.* 2003; Solana Gómez *et al.* 2024).

In addition to the scripts for solving the ATHT, the script **Testcase.m** is provided. This script can be used as a basis for investigations of the hydraulic jump using the ATHT. Therein, parameters and boundary conditions for the flow field and the temperature field are set. Then the ATHT is solved. This means that the variables describing the flow are determined in a first step. Based on these, the variables describing the temperature field are determined.

† Email address for correspondence: solana.gomez@wsa.rwth-aachen.de

Finally, the different scripts for the postprocessing, such as the generation of a graph with the streamlines, are used. Furthermore, the code which can be used to solve only the heat transfer problem with an externally pre-determined velocity field is used. As input for the velocity field, the velocity field determined by the AT is used. Therefore, the solution is identical to the one determined using the original ATHT. The given velocity field can be adapted by the user to achieve different solutions.

Section 3 briefly sums up the folder structure. A more detailed description of the different scripts is given in section 4. Note that Solana Gómez *et al.* (2024) chose a fourth-order polynomial for the modelling of the temperature field, as it leads to better results then the usage of a third-order polynomial. However, the ATHT using a third-order polynomial for the modelling of the temperature field was developed as well and is, therefore, implemented in the present code. The procedure for the construction of the ATHT based on a third-order polynomial is the same as the one presented in (Solana Gómez *et al.* 2024). All the equations resulting for that case, which are not given in (Solana Gómez *et al.* 2024), can be found in this description (see section 4.2). Furthermore, all equations relevant for the numerical solution of the ATHT with an externally given velocity field are given as well (see section 4.2).

3. Basic structure

The scripts are contained in four different folders:

- **01_AveragingTheory_Flow**: This folder contains all scripts for flow calculations in the framework of the AT. The script with the main algorithm to solve the flow is **FlowSolver.m** and the one for determining an initial guess of the flow variables is **GuessFlowVariables.m**.
- **02_AveragingTheory_TemperatureField**: This folder contains all scripts for the calculation of the temperature field in the framework of the ATHT. Within the folder, the scripts are further subdivided depending on the applied boundary condition at the plate ((a) a constant heat flux or (b) a constant plate temperature). For each boundary condition, equations for the usage of a third-order and fourth-order polynomial to model the temperature are included. For the constant heat flux condition, scripts to numerically solve the temperature field in the framework of the ATHT for an externally given velocity field are provided. Note that the velocity field does not need to be determined by the AT. The temperature profile in this case is a fourth-order polynomial. The main scripts for solving the ATHT are **TemperatureFieldSolver.m**, for a constant heat flux, **TemperatureFieldSolverCWT.m** for a constant plate temperature and **TemperatureFieldSolverNum.m** for the numerical solution of the ATHT for a given velocity field.
- **03_MatlabFunctions**: This folder contains scripts which are not directly related to the ATHT, such as the numerical methods used to solve the different ODEs and some other auxiliary methods.
- **04_Postprocessing**: This folder contains all scripts for the postprocessing of the data. This includes scripts for the determination of the different fields, for the determination and visualisation of streamlines, for further investigations of the heat transfer model and for the determination of different parameters describing the recirculation region.

4. Detailed description of the scripts

In the following, all the scripts, i.e. their working principle and the input and output variables, are explained.

4.1. *01_AveragingTheory_Flow*

This folder contains the scripts required for the solution of the hydrodynamic problem.

FlowSolver.m

This is the main script for the solution of the AT. It takes the specifications of the case and returns the variables describing the flow.

Working principle:

(i) Generation of an evenly spaced grid between the specified initial and final radii with the set resolution
(ii) Generation of an initial Guess using **GuessFlowVariables.m**
(iii) Solution of h and λ using **RelaxationMethod.m**

Input:
- **BoundaryConditions**: The boundary conditions consisting of the initial radius, initial height, final radius and final height of the flow. Those have to be given in the form [rInit, hInit; rEnd, hEnd]
- **resolution**: Number of evenly spaced radial points
- **LambdaThreshold**: Threshold value for λ used in the method for the determination of the initial guess. More informations can be found in the description of the script **GuessFlowVariables.m**

Output:
- **grid**: Evenly spaced grid from the initial radius to the final radius
- **height**: Height of the flow
- **lambda**: Shape parameter for the determination of the velocity profile

Command: [grid,height,lambda] = FlowSolver(BoundaryConditions,resolution,LambdaThreshold)

GuessFlowVariables

This script determines the initial guess for the variables describing the flow field, h and λ.

Working principle:

(i) Determine h and λ according to the asymptotic solution for the boundary layer theory before the jump:

$$h = \frac{4}{5G(-3/5)} \frac{r^3 + l^3}{r}$$
$$\lambda = -\frac{3}{5} + \frac{h^4}{5} - \frac{105}{272} r^3 h^3,$$

where l can be determined based on the height prescribed at the initial radius

$$l^3 = \frac{5G(-3/5)}{4} r_\mathrm{i} h_\mathrm{i} - r_\mathrm{i}^3.$$

(ii) Determine h and λ according to the asymptotic solution for the boundary layer theory after the jump:

$$h = \left(12 \ln\left(\frac{r}{r_{\text{end}}}\right) + h_{\text{end}}^4\right)^{\frac{1}{4}}$$
$$\lambda = \frac{3}{10r^2}\left(\frac{3}{h^3} - h\right).$$

(iii) Generate the initial guess by combining the two solutions. The changing point is the point where the outer solution for λ is equal to a previously set threshold value.

Input:
- **grid**: Radial grid of the treated problem
- **BoundaryConditions**: Boundary conditions for the given problem in the form [rInit, hInit; rEnd, hEnd].
- **LambdaThreshold**: Threshold value for λ which determines where the asymptotic solutions for the inner and outer regions are connected

---

Output:
- **Guess**: Initial guess for h and λ given as a matrix which has the values of h along the grid as its first column and the values of λ along the grid as its second column

Command: Guess = GuessFlowVariables(grid,BoundaryConditions,LambdaThreshold).

G, Gprime, Gprimeprime

Scripts for the determination of G as well as the first a second derivatives with respect to λ.

Commands: G = G(lambda) Gprime = Gprime(lambda) Gprimeprime = Gprimeprime(~)

fFlowVariables

Script to determine the right-hand side of the differential equations for the flow variables h and λ. Those are

$$\frac{\mathrm{d}h}{\mathrm{d}r} = -\frac{5\lambda + 3}{rh^3}$$
$$\frac{\mathrm{d}\lambda}{\mathrm{d}r} = \frac{1}{\frac{\mathrm{d}G}{\mathrm{d}\lambda}}\left[\frac{4\lambda r}{h} + \frac{h^4 - (5\lambda + 3)}{rh^4}\right].$$

Input:

- **y**: Values for h and λ saved as

$$\mathrm{y} = \begin{pmatrix} h \\ \lambda \end{pmatrix}$$

- **r**: Radial position

Output:

- **F**: Values for $\mathrm{d}h/\mathrm{d}r$ and $\mathrm{d}\lambda/\mathrm{d}r$ given as

$$\mathrm{F} = \begin{pmatrix} \frac{\mathrm{d}h}{\mathrm{d}r} \\ \frac{\mathrm{d}\lambda}{\mathrm{d}r} \end{pmatrix}$$

Command: F = fFlowVariables(y,r,~)

DfFlowVariables.m

Determines the matrix containing the derivatives of $\mathrm{d}h/\mathrm{d}r$ and $\mathrm{d}\lambda/\mathrm{d}r$ with respect to h and λ. The structure is

$$\begin{pmatrix} \frac{\partial}{\partial h}\left(\frac{\mathrm{d}h}{\mathrm{d}r}\right) & \frac{\partial}{\partial h}\left(\frac{\mathrm{d}\lambda}{\mathrm{d}r}\right) \\ \frac{\partial}{\partial \lambda}\left(\frac{\mathrm{d}h}{\mathrm{d}r}\right) & \frac{\partial}{\partial \lambda}\left(\frac{\mathrm{d}\lambda}{\mathrm{d}r}\right) \end{pmatrix}.$$

Input:

- **y**: Values for h and λ saved as

$$\mathrm{y} = \begin{pmatrix} h \\ \lambda \end{pmatrix}$$

- **r**: Radial position

Output:

- **Df**: Matrix containing the different partial derivatives

Command: Df = DfFlowVariables(y,r,~)

DetReferenceValues.m

Determines the reference values used to nondimensionalise the radial and vertical lengths (r and z) and velocities (u and w). These were introduced in Bohr *et al.* (1997) and are defined as follows:

$$r_* = \left(\frac{q^5}{\nu^3 g}\right)^{\frac{1}{8}},\ z_* = \left(\frac{q\nu}{g}\right)^{\frac{1}{4}},\ u_* = \left(q\nu g^3\right)^{\frac{1}{8}},\ w_* = \left(\frac{\nu^3 g}{q}\right)^{\frac{1}{4}},\ p_* = \rho\left(q\nu g^3\right)^{\frac{1}{4}},$$

where $Q = 2\pi q$ is the volume flow, ν the kinematic viscosity and g the gravitational acceleration.

Input:
- **Q**,ν,**g**: Physical constants appearing in the reference values. Q and not q is given.

Output:
- **ReferenceValues**: Reference values given in the form $[r_*, z_*, u_*, w_*]$.

Command:
ReferenceValues = DetReferenceValues(Q,nu,g)

4.2. *02_AveragingTheory_TemperatureField*

This folder contains the scripts required to solve the temperature field. Its content is further subdivided into two folders which contain the scripts for a constant heat flux condition (**ConstantHeatFlux**) and a constant plate temperature condition (**ConstantWallTemperature**), respectively. The folder for a constant heat flux condition contains the main script for the solution of the problem and three additional folders. One for the use of a third-order polynomial to approximate the vertical temperature profile (**ThirdOrderPolynomial**), one for the use of a fourth-order polynomial (**FourthOrderPolynomial**) and one for the use of an externally given flow field (**NumericalIntegration**). The folder for the constant plate temperature condition contains the main script for the solution of the heat transfer problem with this boundary condition to together with a folder with the scripts required to use a third-order polynomial to describe the temperature profile (**ThirdOrderPolynomial**) and another folder with the scripts required if a fourth-order polynomial is used to model the temperature profile. The basic structure of the algorithm for the determination of the temperature field is shown in Fig. 1. More details about the working principle can be found in the descriptions of the different scripts (**TemperatureFieldSolver.m**,**TemperatureFieldSolverCWT.m** and **TemperatureFieldSolverNum.m**). A more detailed description of how the initial condition for θ_s is reconstructed if $r^1_{\Phi>1} = r_i$ is given in the description of the script **DetInitThetaSurface.m**.

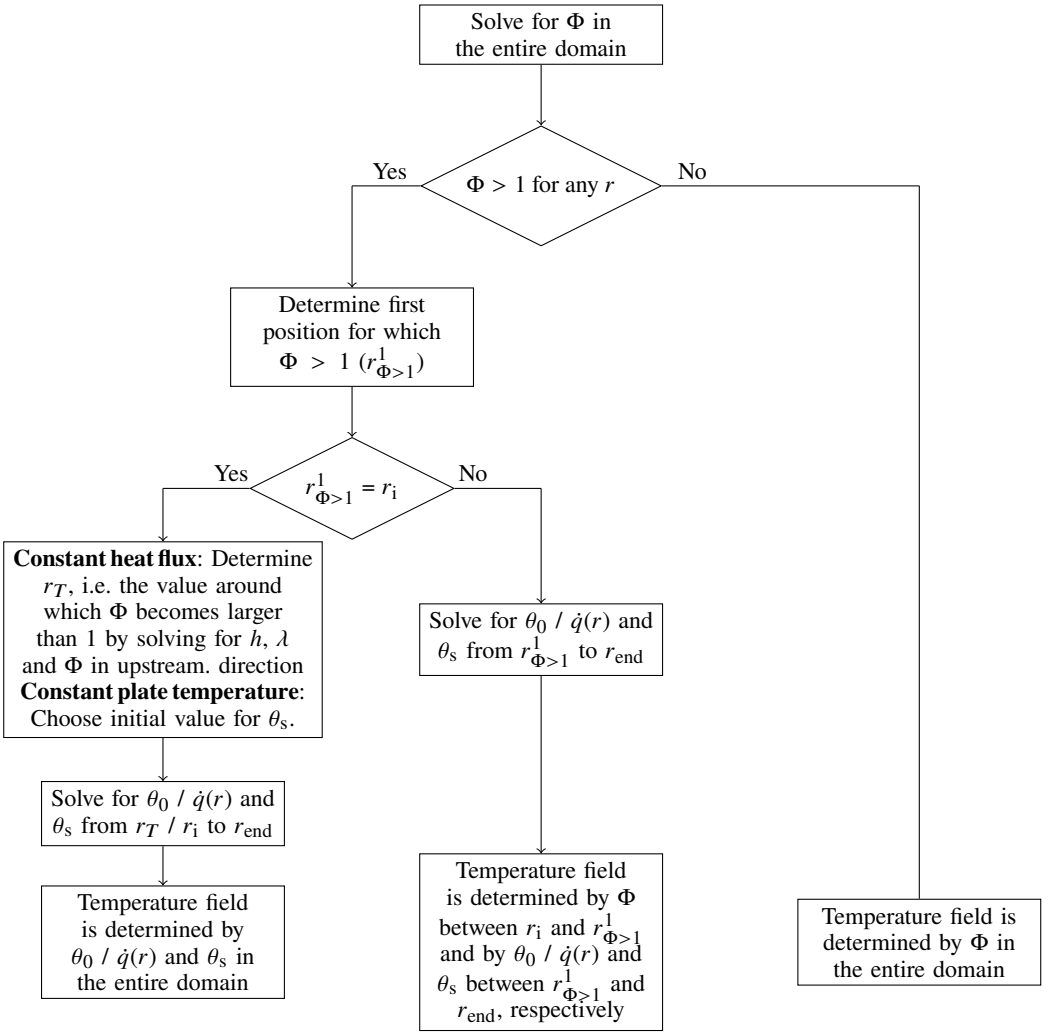


Figure 1: Flowchart of the solution method for the heat transfer problem as presented in (Solana Gómez *et al.* 2024). The flow chart assumes the usage of a fourth-order polynomial for the modelling of the temperature. If a third-order polynomial is used, θ_s is not a relevant magnitude and there is no difference in the procedure between cases for which $r^1_{\Phi>1} = r_i$ holds or does not hold.

4.2.1. *ConstantHeatFlux*

TemperatureFieldSolver.m

This is the main script for solving the ATHT. It takes the specifications of the case together with the variables describing the flow field, i.e the radial grid and the profiles of h and λ and returns the variables describing the temperature field.

Working principle:

For all the following steps either a third-order or a fourth-order polynomial can be used to model the temperature field.

(i) Solve the equation for $\Phi = \delta_T/h$ on the entire radial domain using a Newton method to solve for the root of the respective polynomial.

(ii) Determine the first position for which $\Phi > 1$ holds. Up to the point before this one, the temperature is described by Φ. If $\Phi < 1$ holds for the entire radial domain, Φ describes the flow on the entire domain.

(iii) From the point where $\Phi > 1$ holds for the first time onwards, the equations for plate temperature, θ_0, and, if a fourth-order polynomial is used, the temperature at the free surface, θ_S, are solved. If a third-order polynomial is used, this results in a simple algebraic equation, which can be solved analytically. If a fourth-order polynomial is used, an algebraic equation for θ_0 and a first-order ordinary differential equation for θ_S need to be solved. The initial condition for θ_S is 0, as the solution of the equations starts where the thermal boundary layer hits the free surface. From this point on, θ_0 and, if a fourth-order polynomial is used, θ_S describe the temperature field.A simple explicit Euler method is good enough for the solution of the differential equation.

(iv) If $\Phi > 1$ holds already for the first radius contained in the radial grid and a fourth-order polynomial is used to model the vertical temperature profile, an initial condition is determined for θ_S using the method **DetInitThetaSurface.m**.

Input:

- **grid**: Radial grid generated when solving the hydrodynamic problem
- **height**: Height profile determined when solving the hydrodynamic problem
- **lambda**: Profile of the shape parameter λ determined when solving the hydrodynamic problem
- **HeatTransferParameter**: Parameter required for the solution of the temperature field given as a row vector. The parameters are:
  (i) Pr: Prandtl number of the treated case
  (ii) rStartHeating: Radius at which the heat transfer between the plate and the fluid starts. This value must be smaller or equal to the initial radius of the radial grid. This option has for now only been implemented if a fourth-order polynomial is used
- **OrderPoly**: Order of the polynomials used to model the temperature field in the region described by Φ and the region described by θ_0, and possibly θ_S, given as a row vector

---

Output:

- **delta_t**: Radial profile of the thermal boundary layer thickness
- **Theta0**: Radial profile of the plate temperature
- **ThetaSurface**: Radial profile of the free surface temperature
- **ChangeIndex**: Index of the first radial position for which $\Phi > 1$ holds

```
Command:
[ delta_t,Theta0,ThetaSurface,ChangeIndex ] =
TemperatureFieldSolver(grid,height,lambda,HeatTransferParameter,OrderPoly)
```

Scripts contained in ThirdOrderPolynomial

PhiPoly3.m

Gives result of the left-hand side of the equation which needs to be solved to determine Φ if a third-order polynomial is used the model the vertical temperature profile. This equation is

$$\frac{\lambda+3}{15}\Phi^3 - \frac{5\lambda+3}{72}\Phi^4 + \frac{4\lambda}{210}\Phi^5 - \frac{r^2}{2\mathrm{Pr}h} = 0 \tag{4.1}$$

Command:
result = PhiPoly3(Φ,[r, h, λ],Pr)

DPhiPoly3.m

Gives value of the derivative with respect to Φ of the left-hand side of Eq. (4.1).

Command:
result = DPhiPoly3(Φ,[r, h, λ],~)

HeatFluxPhi3.m

Gives value of the heat flux transported with the flow in the region described by Φ for a given Φ. This can be determined from the left-hand side of Eq. (4.1) by omitting the term independent of Φ.

Command:
result = HeatFluxPhi3(Φ,λ)

DetTheta03.m

Evaluates the equation for θ_0 if a third-order polynomial is used to describe the vertical temperature profile. This equation is

$$\theta_0 = \frac{r^2}{2\mathrm{Pr}} + \left(\frac{61}{120} - \frac{41}{2520}\lambda\right)h$$

Command:
Theta0 = DetTheta03(r,h,λ,Pr)

Scripts contained in FourthOrderPolynomial

PhiPoly4.m

Gives result of the left-hand side of the equation which needs to be solved to determine Φ if a fourth-order polynomial is used the model the vertical temperature profile. This equation is

$$\frac{\lambda+3}{30}\Phi^3 - \frac{5\lambda+3}{168}\Phi^4 + \frac{\lambda}{140}\Phi^5 - \frac{r^2-r_0^2}{2\mathrm{Pr}h} = 0, \tag{4.2}$$

where r_0 is the value at which the heat transfer between the plate and the fluid starts.

Command: result = PhiPoly4(Φ,[r, h, λ],[Pr, r_0])

DPhiPoly4.m

Gives value of the derivative with respect to Φ of left-hand side of Eq. (4.2).

Command: result = DPhiPoly4(Φ,[r, h, λ],~)

HeatFluxPhi4.m

Gives value of the heat flux transported with the flow in the region described by Φ for a given Φ. This can be determined from the left-hand side of Eq. (4.2) by omitting the term independent of Φ.

Command: result = HeatFluxPhi4(Φ,λ)

DetTheta04.m

Evaluates the equation for θ_0 if a fourth-order polynomial is used to describe the vertical temperature profile. This equation is

$$\theta_0 = \frac{1}{1+\mathrm{M}(\lambda)}\left(\frac{1}{2}\frac{r^2}{\mathrm{Pr}} + h\mathrm{N}(\lambda) + \theta_s\mathrm{M}(\lambda)\right),$$

where

$$\mathrm{M}(\lambda) = \int_0^1 f(r,\eta)\left(3\eta^4 - 4\eta^3\right)\mathrm{d}\eta = \frac{1}{30}\lambda - \frac{19}{35} \qquad \mathrm{N}(\lambda) = \int_0^1 f(r,\eta)\left(2\eta^4 - 3\eta^3 + \eta\right)\mathrm{d}\eta = \frac{1}{168}\lambda + \frac{41}{280}.$$

Command:
Theta0 = DetTheta04(r,h,λ,[Pr, r_0],θ_s)

dThetaSdr.m

Evaluates the right-hand side of the differential equation describing θ_s. This differential equation is

$$\frac{\mathrm{d}\theta_S}{\mathrm{d}r} = \frac{6r}{\mathrm{Pr}hf\,(r,\eta=1)}\left[2\,(\theta_0-\theta_S)-h\right],$$

where $f(r,\eta)$ is the function describing the velocity profile.

Command:
FSurfTemp = dThetaSdr(θ_s,[r, h, λ],[Pr, r_0])

DetInitThetaSurface.m

Determines a initial condition for θ_s for the case of a constant heat flux condition if the thermal boundary layer reached the surface before the initial radius. This is only necessary if the vertical temperature profile is modelled using a fourth-order polynomial. There are three different options to choose:

(i) **ThirdOrderPoly**: Uses the equations derived assuming a third-order polynomial to determine θ_s at the initial radius. This is possible because these equations are simple algebraic equations.

(ii) **Analyt**: Uses an analytical relation based on the asymptotic relation for h, $\lambda = -3/5$ and the equation for the determination of Φ to determine the radial position r_T at which the thermal boundary layer hits the free surface. This analytical relation is:

$$r_T = \left(\frac{K}{1-K}\right)^{\frac{1}{3}} l^3,$$

where K and l are defined as

$$K = \frac{8\mathrm{Pr}f\,(\eta=1,\lambda=-3/5)}{5G\,(-3/5)} \quad ; \quad l^3 = \frac{5G\,(\lambda=-3/5)}{4} r_\mathrm{i} h_\mathrm{i} - r_\mathrm{i}^3.$$

r_i and h_i are the initial radius and the initial height. Between r_T and r_i the equations for θ_0 and θ_s are solved starting with the initial condition $\theta_S = 0$. Thus determining the value to be set for θ_s at r_i. For h and λ the asymptotic relations are used.

(iii) **AnalytNum**: A minimal radius is set and h and λ between this radius radius and the initial radius r_i are determined using the asymptotic relations. If in this region a radius exists where Φ changes from a value smaller than 1 to a value larger than 1, the equations for θ_0 and θ_s are solved starting with the initial condition $\theta_s = 0$. Thus, determining the value to be set for θ_s at r_i.

Input:

- **Polynom**:Handle of the function used to describe the temperature field. Up to now, only the use of a fourth-order polynomial is possible.
- **BoundaryConditions**: The boundary conditions consisting of the initial radius, initial

height, final radius and final height of the flow. Those have to be given in the form [rInit, hInit; rEnd, hEnd].

• **lambda**: Value of λ at the initial radius r_i. Only relevant if the option **ThirdOrderPoly** is used.

• **HeatTransferParameter**: Parameter required for the solution of the temperature field given as a row vector. The parameters are:

(i) Pr: Prandtl number of the treated case.

(ii) rStartHeating: Radius at which the heat transfer between the plate and the fluid starts. This value must be smaller or equal to the initial radius of the radial grid. This option has for now only been implemented if a fourth-order polynomial is used.

• **Method**: One of the three methods described above to determine the initial value of θ_S if needed.

• **Param**: Additional values required for the determination of the initial value given in a row vector. Those values are:

(i) Resolution: Resolution chosen for the region before r_i. Only needed if **Analyt** or **AnalytNum** are chosen.

(ii) rMin: Minimal radius needed if **AnalytNum** is chosen.

Command: InitialValue = DetInitThetaSurface(Polynom,BoundaryConditions,lambda, HeatTransferParameter,Method,Param)

Scripts contained in NumericalIntegration

TemperatureFieldSolverNum.m

This is the main script for solving the ATHT using an externally given velocity field. It takes the specifications of the case together with the variables describing the flow field, i.e the radial grid, the profile of h and the velocity field and returns the variables describing the temperature field. For this method only the usage of a fourth-order polynomial for the description of the temperature profile is possible.

Working principle:

(i) Determine $\Phi = \delta_T/h$ on the entire radial domain. The averaged heat transfer equation is evaluated using trapezoidal quadrature and the root is found using a bisection method. More information on the evaluation of the heat transfer equation is given in the description of the script **ThermalEnergyConservation**.

(ii) Determines the first position for which $\Phi > 1$. Up to the point before this one, the temperature is described by Φ. If $\Phi < 1$ holds for the entire radial domain, Φ describes the flow on the entire domain.

(iii) From the point where $\Phi > 1$ holds for the first time onwards, the equations for plate temperature, θ_0, and the temperature at the free surface, θ_S, are solved. The initial condition for θ_S is 0, as the solution of the equations starts where the thermal boundary layer hits the free surface. From this point on, θ_0 and θ_S describe the temperature field.

(iv) If $\Phi > 1$ already holds for the first radius contained in the radial grid, an initial condition is determined for θ_S using the method **DetInitThetaSurface**, which is described in the section about the scripts contained in the folder **FourthOrderPolynomial**. It has to

be kept in mind, that the given flow does not need to be like the one predicted by the AT. Therefore, in this case the methods for the determination of the initial value of θ_S are not necessarily suitable.

Input:

- **grid**: Radial grid
- **height**: Height profile
- **uField**: Velocity field
- **MethodInitCond**: Method chosen for the determination of the initial condition
- **HeatTransferParameter**: Parameter required for the solution of the temperature field given as a row vector. The parameters are:
  (i) Pr: Prandtl number of the treated case
  (ii) rStartHeating: Radius at which the heat transfer between the plate and the fluid starts. This value must be smaller or equal to the initial radius of the radial grid. This option has for now only been implemented if a fourth-order polynomial is used. Up to now the usage of this functionality for is not enabled if an external velocity field is used.
- **lambdaInit**: Value for λ at r_i, which is required if a third-order polynomial is used to determine the initial condition of θ_s in case $\Phi > 1$ holds at r_i

---

Output:

- **delta_t**: Radial profile of the thermal boundary layer thickness
- **Theta0**: Radial profile of the plate temperature
- **ThetaS**: Radial profile of the free surface temperature
- **ChangeIndex**: Index of the first radial position for which $\Phi > 1$ holds.

```
Command:
[ delta_t,Theta0,ThetaS,ChangeIndex ] =
= TemperatureFieldSolverNum(grid,height,uField,MethodInitCond
,HeatTransferParameter,lambdaInit)
```

ThermalEnergyConservation.m

The averaged heat transfer equation reads

$$\vartheta - \frac{r^2}{2\mathrm{Pr}} = 0 \quad ; \quad \vartheta = rh^2 \int_0^1 u\theta d\eta.$$

This relation is evaluated using trapezoidal quadrature for the evaluation of the integral describing ϑ.

Input:

- **Phi**: Φ at the given radius r
- **u**: Radial velocity profile at the given radius r
- **Paramters**: Rest of the relevant magnitudes given as $[r,h,Pr]$

```
Command:
Unconserved = ThermalEnergyConservation(Phi,u,Parameters)
```

DetTheta0Num.m

Evaluates the value of θ_0. As the determination of the equation is performed using an unknown function for the radial velocity profile. The solved equation differs from the one presented by Solana Gómez *et al.* (2024) and is

$$\theta_0 = \frac{1}{\chi_1}\left(\frac{r}{2\mathrm{Pr}h} + \chi_2\right),$$

where

$$\chi_1 = \int_0^1 u\left(1 - 4\eta^3 + 3\eta^4\right) d\eta \qquad \chi_2 = \int_0^1 u\left(h\left(\eta - 3\eta^3 + 2\eta^4\right) - \theta_S\left(4\eta^3 - 3\eta^4\right)\right) d\eta.$$

Both integrals are evaluated using trapezoidal quadrature. If the function used for u in the AT is inserted into the previous equation, the function of the ATHT presented by Solana Gómez *et al.* (2024) is recovered.

Command: Theta0 = DetTheta0Num(r,height,u,ThetaS,Pr)

dThetaSdrNum.m

Evaluates the right-hand side of the differential equation describing θ_S. This differential equation is

$$\frac{\mathrm{d}\theta_S}{\mathrm{d}r} = \frac{6r}{\mathrm{Pr}hf\left(r, \eta = 1\right)}\left[2\left(\theta_0 - \theta_S\right) - h\right],$$

where $f(r, \eta)$ is the function describing the velocity profile. θ_0 is evaluated using **DetTheta0Num.m**.

Command: FSurfTemp = dThetaSdr(ThetaSurface,[r, h, λ],Pr)

4.2.2. *ConstantWallTemperature*

TemperatureFieldSolverCWT.m

This is the main script for solving the ATHT assuming a constant plate temperature. It takes the specifications of the case together with the variables describing the flow field, i.e the radial grid and the profiles of h and λ, and returns the variables describing the temperature field.

Working principle:

For all the following steps either a third-order or a fourth-order polynomial can be used to model the temperature field.

(i) Solve the differential equation for $\Phi = \delta_T/h$ on the entire radial domain. Contrary

to the case of a constant heat flux an initial value for Φ is needed, as Φ is described by a first-order differential equation. As the initial condition the Nusselt number Nu_i at the initial radius r_i is fixed. For Φ this means $\Phi_i = 3/(2\mathrm{Nu}_i)$, if a third-order polynomial is used, and $\Phi_i = 2/(\mathrm{Nu}_i)$, if a fourth-order polynomial is used. For the solution of the differential equation an explicit Euler method is sufficient.

(ii) Determines the first position for which $\Phi > 1$. Up to the point before this one, the temperature is described by Φ. If $\Phi < 1$ holds for the entire radial domain, Φ describes the flow on the entire domain.

(iii) From the point where $\Phi > 1$ holds for the first time onwards, the equations for plate heat flux, $\dot{q}$, and, if a fourth-order polynomial is used, the temperature at the free surface, θ_s, are solved. If a third-order polynomial is used, this results in single first-order differential equation. If a fourth-order polynomial is used a system of two first-order differential equations needs to be solved. The initial condition for θ_s is 0, as the solution of the equations starts where the thermal boundary layer hits the free surface. In order to match the profiles described by Φ and those described by $\dot{q}$ and θ_s at the point where both regions connect, qh is set. If a third-order polynomial is used this means $\dot{q}h = 3/2$ and if a fourth-order polynomial is used this means $\dot{q}h = 2$.

(iv) If $\Phi > 1$ holds already for the first radius contained in the radial grid initial conditions for $\dot{q}$ and, if a fourth-order polynomial is used, θ_S are needed. $\dot{q}$ is determined using the initial condition for the Nusselt number, which results in $qh = \mathrm{Nu}$. For θ_S an additional condition needs to be chosen. In the current method $\theta_\mathrm{s} = 0$ is chosen at r_i.

Input:

- **grid**: Radial grid generated when solving the hydrodynamic problem
- **height**: Height profile determined when solving the hydrodynamic problem
- **lambda**: Profile of the shape parameter λ determined when solving the hydrodynamic problem
- **HeatTransferParameter**: Parameter required for the solution of the temperature field given as a row vector. The parameters are:
  (i) Pr: Prandtl number of the treated case
  (ii) NuInit: Value of the Nusselt number at the initial radius
- **OrderPoly**: Order of the polynomials used to model the temperature field in the region described by Φ and the region described by θ_0 and possibly θ_S given as a row vector.

Output:

- **delta_t**: Radial profile of the thermal boundary layer thickness
- **qh**: Value of the plate heat flux multiplied by the height.
- **ThetaSurface**: Radial profile of the free surface temperature
- **ChangeIndex**: Index of the first radial position for which $\Phi > 1$ holds.

```
Command:
[ delta_t,Theta0,ThetaSurface,ChangeIndex ] =
TemperatureFieldSolver(grid,height,lambda,HeatTransferParameter,OrderPoly)
```

Scripts contained in ThirdOrderPolynomial

dPhidr3CWT.m

Evaluates the right-hand side of the differential equation describing Φ if a third-order polynomial is used to describe the temperature profile. This differential equation is

$$\frac{\mathrm{d}\Phi}{\mathrm{d}r} = \frac{1}{\dfrac{\partial\xi(\Phi,\lambda)}{\partial\Phi}}\left[\frac{3r}{2\Phi h\mathrm{Pr}} - \frac{\partial\xi(\Phi,\lambda)}{\partial\lambda}\frac{\mathrm{d}\lambda}{\mathrm{d}r}\right] \text{ with } \xi(\Phi,\lambda) = \frac{\lambda+3}{10}\Phi^2 - \frac{5\lambda+3}{48}\Phi^3 + \frac{\lambda}{35}\Phi^4.$$

Command:
dPhi = dPhidr3CWT(Phi,[r,height,lambda],Pr)

dqhdr3.m

Evaluates the right-hand side of the differential equation describing $\dot{q}h$ if a third-order polynomial is used to describe the temperature profile. This differential equation is

$$\frac{\mathrm{d}\dot{q}h}{\mathrm{d}r} = \left(\frac{41}{2520}\lambda - \frac{61}{120}\right)^{-1}\left(\frac{r}{\mathrm{Pr}h} - \frac{41}{2520}\frac{\mathrm{d}\lambda}{\mathrm{d}r}\right)\dot{q}h.$$

Command:
dqh = dqhdr3(qh,Curves,Pr)

Scripts contained in FourthOrderPolynomial

dPhidr4CWT.m

Evaluates the right-hand side of the differential equation describing Φ if a fourth-order polynomial is used to describe the temperature profile. This equation is

$$\frac{\mathrm{d}\Phi}{\mathrm{d}r} = \frac{1}{\dfrac{\partial\xi(\Phi,\lambda)}{\partial\Phi}}\left[\frac{2r}{\Phi h\mathrm{Pr}} - \frac{\partial\xi(\Phi,\lambda)}{\partial\lambda}\frac{\mathrm{d}\lambda}{\mathrm{d}r}\right] \text{ with } \xi(\Phi,\lambda) = \frac{\lambda+3}{15}\Phi^2 - \frac{5\lambda+3}{84}\Phi^3 + \frac{\lambda}{70}\Phi^4.$$

Command:
dPhi = dPhidr4(Phi,Curves,Pr)

dqhdr4.m

Evaluates the right-hand side of the differential equation describing $\dot{q}h$ if a fourth-order polynomial is used to describe the temperature profile. In this case the output consists of a

row vector of the form $[\mathrm{d}\,(\dot{q}h)\,/\mathrm{d}r,\ \mathrm{d}\theta_S/\mathrm{d}r]$. The differential equations are

$$\frac{\mathrm{d}\theta_\mathrm{s}}{\mathrm{d}r} = \frac{6\,r}{\mathrm{Pr}\,h\,f\,(r,\eta=1)}\left[2\,(1-\theta_\mathrm{s}) - \dot{q}h\right].$$

$$\frac{\mathrm{d}\,(\dot{q}(r)h)}{\mathrm{d}r} = \frac{1}{\mathrm{N}(\lambda)}\left[\zeta_1\dot{q}(r)h + \zeta_2\,(1-\theta_\mathrm{s})\right],$$

where

$$\zeta_1 = \frac{6\,r\,\mathrm{M}(\lambda)}{\mathrm{Pr}\,h\,f(r,\eta=1)}\left[2 - \dot{q}(r)h - 2\theta_\mathrm{s}\right] \qquad \zeta_2 = \frac{\mathrm{dM}(\lambda)}{\mathrm{d}r} - \frac{12\,r\,\mathrm{M}(\lambda)}{\mathrm{Pr}\,h\,f(r,\eta=1)}$$

and the functions $N\,(\lambda)$ and $M\,(\lambda)$ are the same which are presented in the description of the script **DetTheta04.m**

Command: dq = dqhdr4(FuncValues,Curves,Parameters)

4.3. *03_MatlabFunctions*

This folder includes different functions required for the solution algorithms of the ATHT. These range from basic operations to methods for the solution of integrals and differential equations.

RelaxationMethod.m

Relaxation method for the solution of a stiff differential equation given in the form $\mathrm{d}\boldsymbol{y}/\mathrm{d}x = f(x,\boldsymbol{y})$. $\boldsymbol{y}$ is a vector including multiple variables of the form

$$\boldsymbol{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}$$

Working principle:

(i) An error is defined as

$$E_i = \boldsymbol{y}_{i+1} - \boldsymbol{y}_i - \Delta x_i f\left(x_{i+\frac{1}{2}}, \boldsymbol{y}_{i+\frac{1}{2}}\right),$$

where the index denotes the point in the discrete representation of x for which a certain expression is evaluated, Δx_i is the distance between the i-th point and the subsequent point and values with the index $i + \frac{1}{2}$ are determined by interpolation, using the two surrounding points.

(ii) The solution can be found iteratively starting from a given initial guess by modifying y using the relation

$$\boldsymbol{y}^{j+1} = \boldsymbol{y}^j + \Delta\boldsymbol{y}^j$$

where the superscript denotes the number of the iteration. $\Delta\mathrm{d}\boldsymbol{y}^j$ is determined solving the

following set of equations:

$$E_i^j + \left(\sum_{n=1}^{N} \frac{\partial E_i^j}{\partial y_{n,i}} \Delta y_{n,i}^j \right) + \left(\sum_{n=1}^{N} \frac{\partial E_i^j}{\partial y_{n,i+1}} \Delta y_{n,i+1}^j \right) = 0,$$

together with the boundary conditions. In this last equation the first index of y denotes the n-th variable contained in $\boldsymbol{y}$ and the second index represents the the point in the discrete representation of x for which a certain expression is evaluated.

The code has been developed in such a way that the user can choose which variable contained in $\boldsymbol{y}$ is fixed at which radial point to serve as boundary condition.

Input:

- **Function**: Handle containing the function $f(x, \mathrm{d}\boldsymbol{y})$
- **Derivative**: Handle containing a function which determines the derivative of the function $f(x, \mathrm{d}\boldsymbol{y})$ with respect to the different variables contained in $\mathrm{d}\boldsymbol{y}$ in the form:

$$\begin{pmatrix} \frac{\mathrm{d}f_1}{\mathrm{d}y_1} & \frac{\mathrm{d}f_2}{\mathrm{d}y_1} & \cdots & \frac{\mathrm{d}f_{N-1}}{\mathrm{d}y_1} & \frac{\mathrm{d}f_N}{\mathrm{d}y_1} \\ \frac{\mathrm{d}f_1}{\mathrm{d}y_2} & & & & \frac{\mathrm{d}f_N}{\mathrm{d}y_2} \\ \vdots & & \ddots & & \vdots \\ \frac{\mathrm{d}f_1}{\mathrm{d}y_{N-1}} & & & & \frac{\mathrm{d}f_N}{\mathrm{d}y_{N-1}} \\ \frac{\mathrm{d}f_1}{\mathrm{d}y_N} & \frac{\mathrm{d}f_2}{\mathrm{d}y_n} & \cdots & \frac{\mathrm{d}f_{n-1}}{\mathrm{d}y_N} & \frac{\mathrm{d}f_N}{\mathrm{d}y_N} \end{pmatrix}.$$

- **BoundaryConditions**: Boundary conditions given as a matrix. Each row denotes a different boundary conditions. The given informations are the point at which the condition holds, the given value and the variable of y for which it holds.
- **InitialGuess**: Initial guess for $\boldsymbol{y}$ given in the form

$$\begin{pmatrix} y_{1,1} & y_{2,1} & \cdots & y_{N-1,1} & y_{N,1} \\ y_{1,2} & & & & y_{N,2} \\ \vdots & & \ddots & & \vdots \\ y_{1,M-1} & & & & y_{N,M-1} \\ y_{1,M} & y_{2,M} & \cdots & y_{N-1,M} & y_{N,M} \end{pmatrix},$$

where M is the number of points in the discrete representation of x.

- **Curves**: Other function given for the evaluation of $f(x, \boldsymbol{y})$
- **Parameters**: Constant parameters given for the evaluation of $f(x, \boldsymbol{y})$
- **MaxChange**: If the change of all variable of $\mathrm{d}\boldsymbol{y}$ at all point is smaller than this value for an iteration, the method ends
- **MaxCounter**: Maximum number of iterations performed until the algorithm is stopped

---

Output:

- **Result**: The vector containing the result is given in the form

$$\boldsymbol{y} = \begin{pmatrix} \boldsymbol{y}_1 \\ \boldsymbol{y}_2 \\ \vdots \\ \boldsymbol{y}_M \end{pmatrix}.$$

The indices denote the points of the discrete representation of x where $\boldsymbol{y}$ holds.

Command:
Result =
RelaxationMethod(Function,Derivative,BoundaryConditions,InitialGuess,
Curves,Parameters,MaxChange,MaxCounter)

GenerateSubmatrixForRelaxationMethod.m

The working principle of the relaxation method leads to a system of linear differential equations, which can be constructed by putting together matrices of the size $N \times 2N$ to a matrix of the size $((M-1)\,N) \times (MN)$. The missing equations are the boundary conditions. Here N is the number of variables contained in y and M is the number of grid points for x. This script evaluates a single of those $Nx2N$ matrices, which is

$$\begin{pmatrix} \frac{\partial E_i^j}{\partial y_{1,i}} & \frac{\partial E_i^j}{\partial y_{2,i}} & \cdots & \frac{\partial E_i^j}{\partial y_{N-1,i}} & \frac{\partial E_i^j}{\partial y_{N,i}} & \frac{\partial E_i^j}{\partial y_{1,i+1}} & \frac{\partial E_i^j}{\partial y_{2,i+1}} & \cdots & \frac{\partial E_i^j}{\partial y_{N-1,i+1}} & \frac{\partial E_i^j}{\partial y_{N,i+1}} \\ \frac{\partial E_i^j}{\partial y_{1,i}} & & & & \frac{\partial E_i^j}{\partial y_{N,i}} & \frac{\partial E_i^j}{\partial y_{1,i+1}} & & & & \frac{\partial E_i^j}{\partial y_{N,i+1}} \\ \vdots & & \ddots & & \vdots & \vdots & & \ddots & & \vdots \\ \frac{\partial E_i^j}{\partial y_{1,i}} & & & & \frac{\partial E_i^j}{\partial y_{N,i}} & \frac{\partial E_i^j}{\partial y_{1,i+1}} & & & & \frac{\partial E_i^j}{\partial y_{N,i+1}} \\ \frac{\partial E_i^j}{\partial y_{1,i}} & \frac{\partial E_i^j}{\partial y_{2,i}} & \cdots & \frac{\partial E_i^j}{\partial y_{N-1,i}} & \frac{\partial E_i^j}{\partial y_{N,i}} & \frac{\partial E_i^j}{\partial y_{1,i+1}} & \frac{\partial E_i^j}{\partial y_{2,i+1}} & \cdots & \frac{\partial E_i^j}{\partial y_{N-1,i+1}} & \frac{\partial E_i^j}{\partial y_{N,i+1}} \end{pmatrix},$$

for pair of consecutive points i and $i+1$.

Input:

- **dx**: Distance between the consecutive points
- **yHalf**: Value of ***y*** interpolated between the two points
- **Derivative**: Handle containing a function which determines the derivative of the function $f(x, y)$ with respect to the different variables contained in y in the form:

$$\begin{pmatrix} \frac{\mathrm{d}f_1}{\mathrm{d}y_1} & \frac{\mathrm{d}f_2}{\mathrm{d}y_1} & \cdots & \frac{\mathrm{d}f_{N-1}}{\mathrm{d}y_1} & \frac{\mathrm{d}f_N}{\mathrm{d}y_1} \\ \frac{\mathrm{d}f_1}{\mathrm{d}y_2} & & & & \frac{\mathrm{d}f_N}{\mathrm{d}y_2} \\ \vdots & & \ddots & & \vdots \\ \frac{\mathrm{d}f_1}{\mathrm{d}y_{N-1}} & & & & \frac{\mathrm{d}f_N}{\mathrm{d}y_{N-1}} \\ \frac{\mathrm{d}f_1}{\mathrm{d}y_N} & \frac{\mathrm{d}f_2}{\mathrm{d}y_n} & \cdots & \frac{\mathrm{d}f_{n-1}}{\mathrm{d}y_N} & \frac{\mathrm{d}f_N}{\mathrm{d}y_N} \end{pmatrix}$$

- **CurvesHalf**: Values of other functions, including the radial position, relevant for the evaluation of the differential equation.
- **Parameters**: Parameters relevant for the evaluation of the differential equation.

Command:
Matrix=
GenerateSubmatrixForRelaxationMethod(dx,yHalf,Derivative,CurvesHalf,Parameters)

NewtonMethod.m

Newton method for the determination of the root of a given function

---

Input:
- **Function**: Handle of the function for which a root shall be found
- **Derivative**: Handle of the derivative function for which a root shall be found
- **Guess**: Initial guess for the root
- **Curves**: Values of other functions going into the function
- **Parameters**: Constant parameters going into the function
- **MaxError**: Maximum value of the final error
- **MaxCounter**: Maximum number of iterations

Output:
- **Root**: Determined root
- **Error**: Value of the function at the determined root
- **Counter**: Number of required iterations.

```
Command:
[Root,Error,Counter] = NewtonMethod(Function,Derivative,Guess,Curves,
Parameters,MaxError,MaxCounter)
```

ExplicitEulerMethod.m

Explicit Euler method for the solution of a differential equation given in the form $\mathrm{d}\boldsymbol{y}/\mathrm{d}x = f(x, \boldsymbol{y})$. $\boldsymbol{y}$ is a vector including multiple variables of the form

$$\boldsymbol{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}$$

Input:
- **Function**: Handle of the function given in the form $\mathrm{d}\boldsymbol{y}/\mathrm{d}x = f\,(x, \boldsymbol{y})$.
- **InitialConditions**: Initial conditions for all functions contained in $\boldsymbol{y}$
- **Curves**: Values of the used grid and values of additional functions Each of those entrances is a row vector within a matrix of the form [grid; function1; function2;...;functionN]
- **Parameters**: Constant parameters
- **ConstantStepSize**: If the distance between all adjacent grid points is equal set this to 1, if not, set it to 0.

Output:
- **y**: Solution given in the form

$$\begin{pmatrix} y_{1,1} & y_{2,1} & \cdots & y_{N-1,1} & y_{N,1} \\ y_{1,2} & & & & y_{N,2} \\ \vdots & & \ddots & & \vdots \\ y_{1,M-1} & & & & y_{N,M-1} \\ y_{1,M} & y_{2,M} & \cdots & y_{N-1,M} & y_{N,M} \end{pmatrix},$$

where the first index of y denotes the number of the variable contained in $\boldsymbol{y}$ and the second index represents the point of the discretized form of x where the respective value of y holds. M is the number of points in the discrete representation of x and N the number of variables contained in $\boldsymbol{y}$.

Command: y = ExplicitEulerMethod(Function,InitialConditions,Curves,Parameters, ConstantStepSize)

ExplicitEulerMethodField.m

Explicit Euler method for the solution of a differential equation given in the form $\mathrm{d}\boldsymbol{y}/\mathrm{d}x = f(x, \boldsymbol{y})$, like the function **ExplicitEulerMethod.m** described above. However, additionally a field of the form $u = F(x, \eta)$ is given as input function as well. This function is required in the method **TemperatureFieldSolverNum.m** as for each radial integration step, integrals of the form

$$\chi_1 = \int_0^1 u\left(1 - 4\eta^3 + 3\eta^4\right) d\eta \quad ; \quad \chi_2 = \int_0^1 u\left(h\left(\eta - 3\eta^3 + 2\eta^4\right) - \theta_S\left(4\eta^3 - 3\eta^4\right)\right) d\eta$$

need to be solved.

Input:

- **Function**: Handle of the function given in the form $\mathrm{d}\boldsymbol{y}/\mathrm{d}x = f\,(x, \boldsymbol{y})$. $\boldsymbol{y}$ can be a vector containing multiple variables
- **InitialConditions**: Initial conditions for all functions contained in $\boldsymbol{y}$
- **Field**: Discrete field of the form $u = F(x, \eta)$ given as a matrix of the form

$$\begin{pmatrix} u\,(x_1, \eta_1) & u\,(x_1, \eta_2) & \dots & u\,(x_1, \eta_{n-1}) & u\,(x_2, \eta_m) \\ u\,(x_2, \eta_1) & & & & u\,(x_1, \eta_m) \\ \vdots & & \ddots & & \vdots \\ u\,(x_{n-1}, \eta_1) & & & & u\,(x_{n-1}, \eta_n) \\ u\,(x_n, \eta_2) & u\,(x_n, \eta_1) & \dots & u\,(x_n, \eta_{m-1}) & u\,(x_n, \eta_m) \end{pmatrix}$$

- **Curves**: Values of the used grid and values of additional functions. Each of those entrances is a row vector within a matrix of the form [grid; function1; function2;...;functionN]
- **Parameters**: Constant parameters
- **ConstantStepSize**: If the distance between all adjacent grid points is equal set this to 1, if not, set it to 0.

Output:

- **y**: Solution given in the form

$$\begin{pmatrix} y_{1,1} & y_{2,1} & \dots & y_{N-1,1} & y_{N,1} \\ y_{1,2} & & & & y_{N,2} \\ \vdots & & \ddots & & \vdots \\ y_{1,M-1} & & & & y_{N,M-1} \\ y_{1,M} & y_{2,M} & \dots & y_{N-1,M} & y_{N,M} \end{pmatrix},$$

where the first index of y denotes the number of the variable contained in $\boldsymbol{y}$ and the second index represents the point of the discretized form of x where the respective value of y holds. M is the number of discrete points in the discrete representation of x and N the number of variables contained in $\boldsymbol{y}$.

```
Command:
y = ExplicitEulerMethodField(Function,InitialConditions,Field,Curves,Parameters,
ConstantStepSize)
```

BisectionMethod.m

Bisection method for the determination of the roots of a function

Working principle:

(i) Evaluates the function at a given point and second point defined by a distance to the first point δ.
(ii) Checks if the sign of the two evaluated values is different. If not, δ is added to the both points and the function is evaluated once more at the new points. This is repeated until the evaluated values are of different signs or a maximum number of displacements is reached.
(iii) If two points at which the evaluations of the function give values of different sign are found within the maximum number of displacements, the algorithm proceeds like a standard bisection method.

Input:
- **Function**: Handle of the function for which a root shall be found
- **StartingPoint**: First guess of a root.
- **Delta**: Distance from the first guess at which another computation of the function is performed.
- **Curves**: Values of other functions going into the function.
- **Parameters**: Constant parameters going into the function
- **MaxError**: Maximum value of the final error
- **MaxCounter**: Maximum number of iterations
- **MaxDisplacements**: Maximum number of displacements

Output:
- **Root**: Determined root
- **Error**: Value of the function at the determined root
- **Counter**: Number of required iterations.

```
Command:
[Root,Error,Counter] =
BisectionMethod(Function,StartingPoint,Delta,Curves,Parameters,
MaxError,MaxCounter,MaxDeplacements)
```

TrapezoidRule.m

Algorithm for trapezoid quadrature

Input:
- **grid**: Partition of the domain of integration given as a row vector
- **Function**: Values of the function to be integrated at the points of the partition given as a column vector
- **InitialValue**: Initial value of the integrated function
- **ConstantStepSize**: If the distances between adjacent points in the partition are all equal set this value to 1. If they are not, set it to 0.

```
Command:
Integral = TrapezoidRule(grid,Function,InitialValue,ConstantStepSize)
```

ExcludeValues.m

Excludes specified entries of a vector

Input:
- **OriginalVector**: Vector from which entries shall be excluded
- **Positions**: Vector including the positions of the entries which shall be excluded

```
Command:
Vector = ExcludeValues(OriginalVector,Positions)
```

ExcludeColumns.m

Excludes columns from a matrix.

Input:
- **OriginalMatrix**: Matrix from which columns shall be excluded
- **ErasedColumns**: Column vector including the numbers of the columns which shall be excluded

```
Command:
Matrix = ExcludeColumns(OriginalMatrix,ErasedColumns)
```

IncludeValues.m

Includes values into a vector

Input:
- **OriginalVector**: Vector into which the new values shall be included
- **NewValues**: Values which shall be included into the vector given as [Values, Positions].

Here Values is a column vector including the values which shall be added. Positions is a column vector including the positions where the new values shall be added.

```
Command:
Vector = IncludeValues(OriginalVector,NewValues)
```

InterpolateValue.m

Algorithm to interpolate values of discrete given functions

Input:
- **grid**: Partition of the domain of integration given as a row vector
- **Data**:Values at the points of the partition given as a column vector
- **Position**: Position of the point for which a value shall be interpolated.

```
Command:
Value = InterpolateValue(grid,Data,Position)
```

Order.m

Orders a matrix according to the size of the entries in a specified column and from the smallest value to the largest value

---

Input:
- **Matrix**: Matrix which shall be ordered
- **Column**: Column according to which the matrix shall be ordered

```
Command:
OrderedMatrix = Order(Matrix,Column)
```

4.4. *04_Postprocessing*

This folder includes several scripts used for the postprocessing of the data. It is itself subdivided into four folders. **1_Computation_of_fields_and_profiles** contains the methods for the construction of the different fields and profiles (velocity and temperature) from the variables describing the different phenomena (h,λ,Φ,θ_0,θ_S).
2_Analysis_of_streamlines includes the methods for the determination of open and closed streamlines. **3_Analysis_of_heat_transfer** contains methods for the evaluation of the heat transfer. **4_Recirculation_region** contains two methods for the characterization of the recirculation region.

Scripts contained in 1_Computation_of_fields_and_profiles

ComputefField.m

Computes the field of the function $f\,(r,\eta)$ used in the description of the radial velocity

Input:
- **eta**: Chosen grid for η
- **lambda**: λ as determined by the AT

Command: f = ComputefField(eta,lambda)

ComputeUField.m

Computes the radial velocity field

Input:
- **eta**: Chosen grid for η
- **grid**: Radial grid given by the AT
- **height**: h as determined by the AT
- **lambda**: λ as determined by the AT

Command: Vel = ComputeUField(eta,grid,height,lambda)

ComputeVField.m

Computes the vertical velocity field

Input:
- **eta**: Chosen grid for η
- **grid**: Radial grid given by the AT
- **height**: h as determined by the AT
- **lambda**: λ as determined by the AT

Command: V = ComputeVField(eta,grid,height,lambda)

Intdflambda.m

Evaluates the expression

$$\int_0^{\eta} \frac{\mathrm{d}f}{\mathrm{d}\lambda} \mathrm{d}\hat{\eta}$$

Input:
- **eta**: Values of η for which the expression shall be evaluated

Command: Intdf = Intdfdlambda(eta)

ComputeThetaField.m

Computes the temperature field.

Input:

- **eta**: Chosen grid for η
- **height**: h as determined by the AT
- **delta_t**: δ_T as determined by the ATHT
- **ParametersOnPlate**: Parameter on the plate as determined by the ATHT. θ_0 if a constant heat flux is used and $\dot{q}h$ if a constant plate temperature is used.
- **ThetaSurface**: θ_S as determined by the ATHT
- **CaseSpecifications**: Specifications of the case given as a row vector with the following entries

  (i) **OrderOfPolynomial**: Order of the polynomial used for the modelling of the temperature

  (ii) **BoundaryCondition**: Set to 1 if a constant heat flux is used and to 2 if a constant plate temperature is used.

```
Command:
TProfile = ComputeThetaField(eta,height,delta_t,ParameterOnPlate,
ThetaSurface,CaseSpecifications)
```

Scripts contained in 2_Analysis_of_streamlines

DrawStreamlines.m

Plots specified open and closed streamlines.

Input:

- **grid**: Radial grid as given by the AT
- **height**: h as determined by the AT
- **lambda**: λ as determined by the AT
- **HStart**: Starting height values for the open streamlines given as a row vector. For more information see the description of the function **GetOpenStreamlines**.
- **RelativeStreamFunctionValue**: Relative values of the stream function of the closed streamlines with respect to the minimal stream function value. For the treated problem, the value of the stream function is equal to the flow under the vertical position of the point. Therefore, all points within the recirculation bubble have negative the stream function values. The minimal stream function value is found for the point in the recirculation bubble where $u = 0$ and $w = 0$.
- **RescalingFactors**: Factors for the rescaling of the radius and the height given as a row vector ([RescalingFactorR, RescalingFactorH])

```
Command:
DrawStreamlines(grid,height,lambda,HStart,RelativeStreamFunctionValue,
RescalingFactors)
```

GetOpenStreamline.m

Determines an open streamline which is fixed by choosing its height at the initial radius.

Input:

- **height**: h as determined by the AT
- **lambda**: λ as determined by the AT
- **HStart**: Height of the streamline at the initial radius

```
Command:
OpenStreamline = GetOpenStreamline(height,lambda,HStart)
```

GetClosedStreamlines.m

Determines a closed streamline which is fixed by choosing its stream function value. The closed streamline is given as two line, a lower one and an upper one. Both start and end on the line for which the condition $u = 0$ holds without being the plate surface.

Working principle:

(i) Determines the stream function on the line where $u = 0$ holds without being the plate surface.
(ii) Determines between which radial points the closed streamline intersects the line where $u = 0$, i.e the points stream function changes sign are found. The interior points are chosen as start and end points of the upper and lower lines. Here interior points means the point after the intersection for the start and the point before the intersection for the end.
(iii) Uses a bisection method to determine the heights of the closed streamline (upper and lower line) for the points laying between the start and end point.

Input:

- **height**: h as determined by the AT
- **lambda**: λ as determined by the AT
- **EtaFZero**: Value of η on the line on which the radial velocity is 0, which is not the plate surface.
- **RecirculationIndices**: Indices of the radial points which delimit the recirculation region. The function to determine them (**GetIndicesRecirculation.m**) is in the folder **4_Recirculation_region**.
- **StreamFunctionValue**: Value of the stream function on the closed streamline which shall be determined.

Output:

- **LowerLine**: Lower line of the closed streamline
- **UpperLine**: Upper line of the closed streamline
- **IndexBeginning**: Index of the radial point at which the specific closed streamline starts
- **IndexEnd**: Index of the radial point at which the specific closed streamline ends

```
Command:
[LowerLine,UpperLine,IndexBeginning,IndexEnd] =
GetClosedStreamline(height,lambda,EtaFZero,RecirculationIndices,
StreamFunctionValue)
```

FindEta.m

Finds the value of η at which a certain stream function value holds considering a given value of λ using a Newton method.

Input:
- **lambda**: Value of λ
- **Integral**: Value of the stream function
- **InitialGuess**: Initial guess for the Newton method.

```
Command:
Eta = FindEta(lambda,Integral,InitialGuess)
```

FindEta2.m

Finds the value of η at which a certain stream function value holds considering a given value of λ using a bisection method.

Input:
- **lambda**: Value of λ
- **Integral**: Value of the stream function
- **StartingPoint**: Specified in more detail in the description of **BisectionMethod.m**
- **Delta**: Specified in more detail in the description of **BisectionMethod.m**

```
Command:
Eta = FindEta2(lambda,Integral,StartingPoint,Delta)
```

IntegrateF.m

Evaluates the expression

$$\int_0^{\eta} f\,(r, \hat{\eta})\,\mathrm{d}\hat{\eta}$$

.

Input:
- **lambda**: Value of λ
- **eta**: Value of η

```
Command:
FInt = IntegrateF(lambda,eta)
```

IntegrateF2.m

Evaluates the expression

$$\int_0^{\eta} f\,(r, \hat{\eta})\,\mathrm{d}\hat{\eta} - C,$$

where C is a constant.

Input:
- **lambda**: Value of λ
- **eta**: Value of η
- **IntKonst**: Value of C

```
Command:
FInt = IntegrateF2(eta,lambda,IntKonst)
```

Scripts contained in 3_Analysis_of_heat_transfer

ComputeHeatFlux.m

Computes the heat flux transferred between the plate and the flow and the heat flux transported with the flow. Both should coincide if the model works properly.

Input:
- **grid**: Radial grid given by the AT
- **eta**: Chosen grid for η
- **height**: h as determined by the AT
- **lambda**: λ as determined by the AT
- **delta_t**: δ_T as determined by the ATHT
- **ParameterOnPlate**: θ_0 or $\dot{q}h$ depending on the chosen boundary condition
- **ThetaSurface**: θ_S as determined by the ATHT
- **CaseSpecifications**: Specifications of the case given as a row vector with the following entries

  (i) **OrderOfPolynomial**: Order of the polynomial used for the modelling of the temperature

  (ii) **BoundaryCondition**: Set to 1 if a constant heat flux is used and to 2 if a constant plate temperature is used.

Output:
- **Hf**: Heat transported with the flow
- **Hp**: Heat transferred between the plate and the flow

```
Command:
[Hf,Hp] =
ComputeHeatFlux(grid,eta,height,lambda,delta_t,ParameterOnPlate,
ThetaSurface,CaseSpecifications)
```

FindPrTh.m

Determines $\mathrm{Pr_{th}}$

Input:

- **grid**: Radial grid given by the AT
- **height**: h as determined by the AT
- **lambda**: λ as determined by the AT
- **OrderPoly**: Order of the polynomials used to model the temperature field in the region described by Φ and the region described by θ_0 and possibly θ_S given as a row vector.
- **HeatTransferParameter**: Parameter required for the solution of the temperature field given as a row vector. The parameters are:
  (i) Pr: Prandtl number of the treated case
  (ii) rStartHeating: Radius at which the heat transfer between the plate and the fluid starts must be smaller or equal to the initial radius of the radial grid. This option has for now only been implemented if a fourth-order polynomial is used.
- **precision**: Maximum deviation between the actual value and the determined value. It needs to be kept in mind that a precise determination of $\mathrm{Pr_{th}}$ requires a fine radial mesh.

Command: Pr = FindPrTh(grid,height,lambda,OrderPoly,HeatTransferParameter,precision)

Scripts contained in 4_Recirculation_region

AnalyseRecirculation.m

Gives different parameters describing the recirculation region

---

Input:

- **grid**: Radial grid given by the AT
- **lambda**: λ as determined by the AT
- **height**: h as determined by the AT

Output:

- **FZero**: Heights in the recirculation region for which $f(r,\eta) = 0$ holds
- **DFZero**: Heights in the recirculation region for which $\mathrm{d}f(r,\eta)/\mathrm{d}\eta = 0$ holds
- **IntFZero**: Heights in the recirculation region for which $\int_0^1 f(r,\eta)\,\mathrm{d}\eta = 0$ holds. This is the outer boundary of the recirculation bubble
- **EtaFZero**: Heights in the recirculation region for which $f(r,\eta) = 0$ holds
- **EtaDFZero**: Heights in the recirculation region for which $\mathrm{d}f(r,\eta)/\mathrm{d}\eta = 0$ holds
- **EtaIntFZero**: Heights in the recirculation for which which $\int_0^1 f(r,\eta)\,\mathrm{d}\eta = 0$ holds. This is the outer boundary of the recirculation bubble.
- **gridRecirculation**:Section of the radial grid in which the recirculation bubble lays
- **lambdaRecirculation**: λ in the section in which the recirculation bubble lays
- **heightRecirculation**: h in the section in which the recirculation bubble lays

Command:
[FZero,DFZero,IntFZero,EtaFZero,EtaDFZero,EtaIntFZero,gridRecirculation,
lambdaRecirculation,heightRecirculation]
= AnalyseRecirculation(grid,lambda,height)

GetIndicesRecirculation.m

Gives the indices of the radial points delimiting the recirculation bubble as a row vector. At the leading and the trailing edge of the recirculation bubble $\mathrm{d}u/\mathrm{d}\eta = 0$ holds at the plate surface. In the AT this means $\lambda = -3$.

Input:
- **lambda**: λ as determined by the AT

Command:
IndicesRecirculation = GetIndicesRecirculation(lambda)

REFERENCES

Bohr, T., Putkaradze, V. & Watanabe, S. 1997 Averaging theory for the structure of hydraulic jumps and separation in laminar free-surface flows. *Physical Review Letters* **79** (6), 1038–1041.

Solana Gómez, R., Bohr, T., Nielsen, S., Rohlfs, W., Kneer, R. & Askarizadeh, H. 2024 Averaging theory for heat transfer in circular hydraulic jumps with a separation bubble. *Journal of Fluid Mechanics* **979**, A41.

Watanabe, Sh., Putkaradze, V. & Bohr, T. 2003 Integral methods for shallow free-surface flows with separation. *Journal of Fluid Mechanics* **480**, 233–265.