Cite as: Sohn, J. : Modifying buoyantBoussinesqSimpleFoam solver to consider a crop transpiration model using fvOptions. In Proceedings of CFD with OpenSource Software, 2024, Edited by Nilsson. H.,
http://dx.doi.org/10.17196/OS_CFD#YEAR_2024

CFD with OpenSource software

A course at Chalmers University of Technology
Taught by Håkan Nilsson

Modifying buoyantBoussinesqSimpleFoam solver to consider a crop transpiration model using fvOptions

Developed for OpenFOAM-v2112

Author:
Jonas Sohn
Aarhus University
jsohn@btech.au.dk

Peer reviewed by:
Javier Camacho
Saeed Salehi
George Xydis

Disclaimer: This is a student project work, done as part of a course where OpenFOAM and some other OpenSource software are introduced to the students. Any reader should be aware that it might not be free of errors. Still, it might be useful for someone who would like to learn some details similar to the ones presented in the report and in the accompanying files. The material has gone through a review process. The role of the reviewer is to go through the tutorial and make sure that it works, that it is possible to follow, and to some extent correct the writing. The reviewer has no responsibility for the contents.

January 24, 2025

Learning outcomes

The main requirements of a tutorial in the course is that it should teach the four points: How to use it, The theory of it, How it is implemented, and How to modify it. Therefore the list of learning outcomes is organized with those headers.

The reader will learn:

How to use it:

- How to use `fvOptions` to apply custom source terms in `OpenFOAM`.
- How to use the `DarcyForchheimer` porous media model in `OpenFOAM`.

The theory of it:

- The theory of the `buoyantBoussinesqSimpleFoam` solver
- To understand the Darcy-Forchheimer equation to simulate drag in a porous media, applicable to canopy drag in crop models.
- To understand the crop transpiration model and crop energy balance.

How it is implemented:

- How the `explicitPorositySource` class is implemented
- How the `codedSource` class is implemented

How to modify it:

- How to modify the `buoyantBoussinesqSimpleFoam` solver to account for water vapor
- How to implement additional sink and source terms using `fvOptions`.

Prerequisites

The reader is expected to know the following in order to get maximum benefit out of this report:

- How to run standard document tutorials like `hotRoom` tutorial.
- Fundamentals of Computational Methods for Fluid Dynamics, Book by J. H. Ferziger and M. Peric
- How to customize a solver and do top-level application programming.

Contents

Nomenclature

Acronyms

AH	Absolute Humidity
CFD	Computational Fluid Dynamics
LAI	Leaf Area Index
PPFD	Photosynthetic Photon Flux Density

English symbols

C_1	Viscous Resistance Factor	m^{-2}
C_2	Inertial Resistance Factor	m^{-1}
c_p	Specific Heat at Constant Pressure	$\mathrm{J/(kgK)}$
D_{eff}	Effective Diffusivity	m^2/s
g	Gravitational Acceleration	$9.81 m/s^2$
k	Thermal Conductivity	$\mathrm{W/(mK)}$
l	Mean Leaf Diameter	m
Q_{lat}	Latent Heat Flux	$\mathrm{W/m}^2$
Q_{sen}	Sensible Heat Flux	$\mathrm{W/m}^2$
R_{net}	Net Radiation	$\mathrm{W/m}^2$
Sc	Schmidt number	
Sct	Turbulent Schmidt number	
T	Temperature	K
u	Local Airspeed	m/s
V	Cell Volume	m^3

Greek symbols

α	Thermal Diffusivity	m^2/s
β	Coefficient of Thermal Expansion	1/K
ϵ	Psychrometric Constant	dimensionless
λ_W	Latent Heat of Vaporization	J/kg
μ	Dynamic Viscosity	Pas
ν	Kinematic Viscosity	m^2/s
ρ	Density	$\mathrm{kg/m}^3$

Subscripts

a	Ambient Air
l	Leaf Surface
Ref	Reference
s	Stomatal

Chapter 1

Introduction

1.1 Background

Crop transpiration is a critical physiological process that plays an essential role in plant growth and environmental control within controlled environments such as greenhouses or vertical farms. The process involves the transfer of water vapor from plant leaves to the surrounding air, affecting both heat and moisture exchange. Accurate modeling of transpiration is key to optimize indoor climate for uniform crop growth and energy efficiency. Traditional studies have extensively used Computational Fluid Dynamics (CFD) tools such as ANSYS Fluent to model airflow, heat transfer, and crop response, incorporating user-defined functions to simulate complex interactions [1, 2, 3, 4]. An updated version of the Penman-Monteith equation and the "big leaf" model by Graamans et al. [5] have been widely applied to quantify energy balances and latent heat dissipation in vertical farming setups. While ANSYS-based studies have provided valuable insights, their closed-source nature limits adaptability and customization. This limitation has driven my interest in integrating such model into open-source platforms like `OpenFOAM`, allowing broader accessibility and flexibility.

1.2 Objectives

This report aims to enhance the modeling capabilities of `OpenFOAM` by integrating a crop transpiration model into the application of a modified `buoyantBoussinesqSimpleFoam` solver. The primary objective is to incorporate physiological processes such as water vapor exchange into the simulation to improve the prediction of environmental conditions in controlled agricultural environments. Specifically, the focus lies in adapting the solver to account for crop transpiration effects by utilizing the `fvOptions` framework. This involves modifying the `buoyantBoussinesqSimpleFoam` solver and advancing its capabilities to consider water vapor transport by coupling heat and mass transfer, which enables simulations to reflect the dynamic interaction of crop processes with their environment. The report seeks to provide a comprehensive guide to the theory and practical implementation of these modifications. It addresses the limitations of closed-source solutions, promoting flexibility and adaptability in agricultural and environmental modeling using `OpenFOAM` open-source platform. The modified solver and crop transpiration model is demonstrated in a simple tutorial case showing temperature and humidity dynamics in porous media regions.

1.3 Report structure

This report is organized into six main chapters to guide the reader from theoretical concepts to practical implementation. It aims to offer a comprehensive understanding of the available theory and implementation as well as the crop transpiration model development and its integration into the `OpenFOAM` framework:

Chapter 2: Theory. This chapter discusses the theoretical foundations of the `buoyantBoussinesqSimpleFoam` solver, the porous media model, and the crop transpiration model. It establishes the basis for understanding the subsequent implementation and simulation results.
Chapter 3: Existing implementations. Here, `OpenFOAM`'s existing capabilities are examined to incorporate custom source terms and resistance models, providing a framework for implementing new functionalities to represent a crop model.
Chapter 4: Implementations. Relevant modifications for both, the solver and `fvOptions`, are described and presented, including the coupling of heat and mass transfer as well as the integration of the crop transpiration model through `fvOptions`.
Chapter 5: Instructions for solver application. A step-by-step guide is provided for the proposed solver and how to create a tutorial case, demonstrating the practical application of the modified solver. This section outlines the solver, its compilation, simulation setup, and presents results verifying the new implementation.
Chapter 6: Conclusion and future work. In the final chapter, key findings and contributions are summarized and opportunities for future work proposed to enhance the capability and validity of the presented work.

Chapter 2

Theory

2.1 Fundamentals of `buoyantBoussinesqSimpleFoam` solver

In cases where density variations are relatively small, the Boussinesq approximation provides a practical and efficient approach to solving these flows without fully resolving compressibility effects. The `buoyantBoussinesqSimpleFoam` solver in `OpenFOAM` utilizes the Boussinesq approximation to model steady-state, incompressible buoyant flows. Thus, the governing equations of fluid motion are simplified by assuming that density variations are negligible in all terms of the equations except the buoyancy term. This assumption is valid for low Mach number flows (typically less than 0.3) where temperature-induced density changes are small but sufficient to create buoyant forces [6]. In short, the Boussinesq approximation assumes density, ρ, as constant everywhere except in the buoyancy term of the momentum equation, and the buoyant force is approximated as a linear function of temperature relative to a reference temperature. Thus, the density variations can be expressed as

$$\rho = \rho_0(1 - \beta\rho_0(T - T_{\text{Ref}})) \tag{2.1}$$

where T is the relative temperature, T_{Ref} is the reference temperature, ρ represents the density and β the coefficient of thermal expansion. These considerations lead to a system of equations that is more computationally efficient to solve, while still capturing the essential physics of buoyant flow. As a result, the continuity equation for conservation of mass can be reduced and formulated as

$$\nabla \cdot \mathbf{U} = 0 \tag{2.2}$$

where $\mathbf{U}$ is the local velocity of a fluid and the density variation is ignored due to the Boussinesq approximation. However, in the momentum equation, it must be accounted for density variations with a fixed part and linear dependent part on temperature described in Eq. (2.1). This results in a equation that can be described as

$$\rho\mathbf{U} \cdot \nabla\mathbf{U} = -\nabla p + \nabla \cdot \left(\mu\nabla^2\mathbf{U}\right) + \mathbf{g}\beta\left(T - T_{\text{Ref}}\right) \tag{2.3}$$

where the left-hand side term is the convective term, describing momentum transfer due to the fluid's velocity field. On the right side, $-\nabla\rho$ is the pressure gradient term, accounting for pressure forces, $\nabla \cdot \mu\nabla^2\mathbf{U}$ represents the viscous term, modeling momentum diffusion due to fluid viscosity, and lastly, the buoyancy term $\mathbf{g}\beta(T - T_{\text{Ref}})$, representing the effect of temperature differences on fluid motion.

The energy equation is typically formulated in terms of temperature or enthalpy, where buoyancy effects are modeled as density variations proportional to temperature changes. The temperature equation for this solver can be written as

$$\nabla \cdot (\rho\mathbf{U}T) - \nabla \cdot (\alpha\nabla T) = S_{\text{T}} \tag{2.4}$$

where the convective term $\nabla\cdot(\rho\mathbf{U}T)$ represents the transport of temperature by the flow field $\mathbf{U}$. The diffusive term $\nabla \cdot (\alpha\nabla T)$ represents thermal diffusion, where α is the thermal diffusivity ($\alpha = \frac{k}{\rho c_{\text{p}}}$,

with k as thermal conductivity, ρ as density, and c_p as specific heat at constant pressure). The source term accounts for any additional sources of thermal energy, such as internal heat generation, radiation or sink terms. The energy equation works alongside the momentum and continuity equations to solve the flow field, where buoyancy effects are incorporated into the momentum equation using the Boussinesq approximation.

2.2 Fundamentals of the porous media model

With the application of the porous media model, the simulation domain assumes a material or medium with voids, pores or is filled with solid parts that accounts for the interaction between the fluid and medium within the porous structure, capturing phenomena such as resistance to flow, momentum transfer, and energy dissipation [7]. This is a bidirectional concept where the flow can pass through opposite direction but properties such as porosity, permeability, and inertial resistance, significantly influence the flow behavior. Using a porous media model can simplify the simulation domain, reduces mesh complexity and saves computational time. In turbulent flow through porous media, the momentum and continuity equations are modified to account for the porous matrix and resistance forces. The Darcy–Forchheimer equation, an extension of Darcy's law, adds an inertial term to the viscous resistance, enabling accurate modeling of turbulent effects. The Darcy–Forchheimer equation provides a robust framework for addressing these interactions by incorporating both viscous and inertial resistance forces [7, 8].

The following section explains the theoretical implementation of the porous media resistance model in `OpenFOAM`. The implementation of this equation in `OpenFOAM` involves splitting the resistance forces into linear (Darcy) and quadratic (Forchheimer) terms. These terms are applied in the momentum equation explicitly. In `OpenFOAM`, this formulation is implemented within the `explicitPorositySource` and `DarcyForchheimer` classes. These classes allow users to define the resistance parameters and apply the porous media model to specific regions in the computational domain. The consideration of resistance factors and thus, the Darcy-Forchheimer source term in the momentum equation can be expressed as a volumetric source term added to the momentum equation in the following general form of

$$S = \underbrace{\frac{\mu}{K}\mathbf{u}}_{\textbf{Viscous Resistance}} + \underbrace{\frac{\rho C_\mathrm{f}}{\sqrt{K}}\|\mathbf{u}\|\mathbf{u}}_{\textbf{Inertial Resistance}}, \tag{2.5}$$

where K represents the permeability of the porous medium (m^2) and $\frac{\mu}{K}$ is the viscous drag coefficient which is multiplied with the local velocity. The first term represents the Darcy (linear) resistance coefficients for each direction and dominates at low flow velocities (laminar flow regimes). The non-linear momentum loss coefficient C_f is used for the viscous resistance. This term accounts for nonlinear effects due to inertial forces and dominates at higher flow velocities (turbulent or high-speed flow regimes). Thus, the viscous (C_1) and inertial resistance factor (C_2) can be calculated as

$$C_1 = \frac{1}{K}, \tag{2.6}$$

and

$$C_2 = \frac{C_\mathrm{f}}{\sqrt{K}}, \tag{2.7}$$

relevant for their application in porous regions using `fvOptions` and the `explicitPorositySource` class in Section 3.2. For the test case application in Chapter 5, a permeability of 0.02 m^2 and a non-linear momentum loss coefficient of 0.134 is assumed, which results in $C_1 = 50\ \mathrm{m}^{-2}$ and $C_2 = 2.0\ \mathrm{m}^{-1}$ [4]. Although, the porous media model provides a computational efficient method to consider crop resistance in a simulation, it is important to note that the complex structure of plant will be simplified. Thus, detailed airflow patterns in and around plants can not be captured and analyzed, which might yield valuable insights for designing more efficient airflow control systems.

2.3 Crop transpiration model

A robust crop transpiration model is vital for the evaluation and optimization of environmental conditions. It provides a numerical framework incorporates that heat, moisture, and momentum exchange between the crops and the surrounding air. This model can be integrated as a porous media representation of the crop canopy [1], which simplifies the computational complexity by avoiding explicit geometric modeling of individual plants [3]. Graamans et al. established a validated crop transpiration model of lettuce to account for heat and moisture exchange between crops and ambient air in simulations based on the concept of the "big leaf" [5]. It considers the energy balance of the crop due to the net radiation absorbed by the crop and its sensible and latent heat exchange illustrated in Figure 2.1a. Thus, the crop transpiration rate, ET, can be derived and calculated using the equation

$$ET = \mathrm{LAI} \cdot \frac{\chi_\mathrm{l} - \chi_\mathrm{a}}{r_\mathrm{s} + r_\mathrm{a}}, \tag{2.8}$$

where LAI is the leaf area index, defined as the one-sided leaf area per unit ground area. The variables χ_l and χ_a denote the water vapor concentration at the leaf surface and in the surrounding air, respectively. The leaf surface water concentration, χ_l, is assumed to be the saturated vapor concentration at the leaf surface temperature, while χ_a represents the ambient water vapor concentration. Solving for ET, the added mass source of water vapor can be derived. The aerodynamic resistance, r_a, and stomatal resistance, r_s, regulate the rate of transpiration. The aerodynamic resistance is computed as

$$r_\mathrm{a} = 350 \left(\frac{l}{u}\right)^{0.5} \mathrm{LAI}^{-1}, \tag{2.9}$$

and stomatal resistance can be calculated as

$$r_\mathrm{s} = 60 \frac{1500 + \mathrm{PPFD}}{200 + \mathrm{PPFD}}, \tag{2.10}$$

where l is the mean leaf diameter, u is the local airspeed derived from CFD simulations, and $PPFD$ represents the photosynthetic photon flux density. The sum of these two resistances represent the total resistance between inside leaves and air as illustrated in Figure 2.1b. However, before solving the crop transpiration, the energy balance Eq. (2.11) must be solved. Therefore, the leaf surface temperature, T_l, is determined by solving the energy balance of the crop canopy through an iterative method (which is explained in Section 4.3). The energy absorbed from net radiation, R_net, must balance the sensible heat flux, Q_sen, and latent heat flux, Q_lat, as expressed by

$$R_\mathrm{net} - Q_\mathrm{sen} - Q_\mathrm{lat} = 0, \tag{2.11}$$

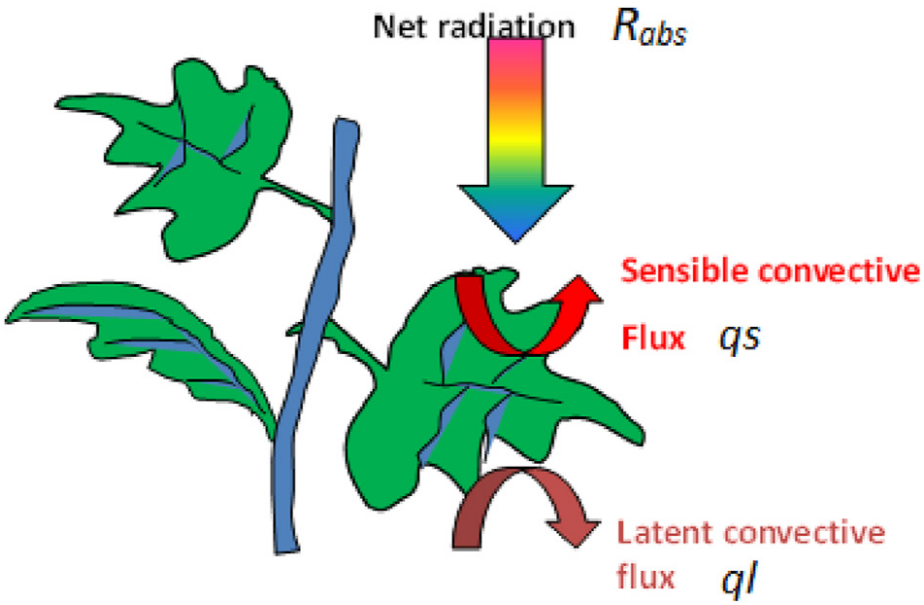
$$R_\mathrm{net} = (1 - \rho_\mathrm{r}) \cdot I_\mathrm{lighting} \cdot \mathrm{CAC}, \tag{2.12}$$

$$Q_\mathrm{sen} = \mathrm{LAI} \cdot \rho_\mathrm{a} \cdot c_\mathrm{p} \cdot \frac{T_\mathrm{l} - T_\mathrm{a}}{r_\mathrm{a}}, \tag{2.13}$$

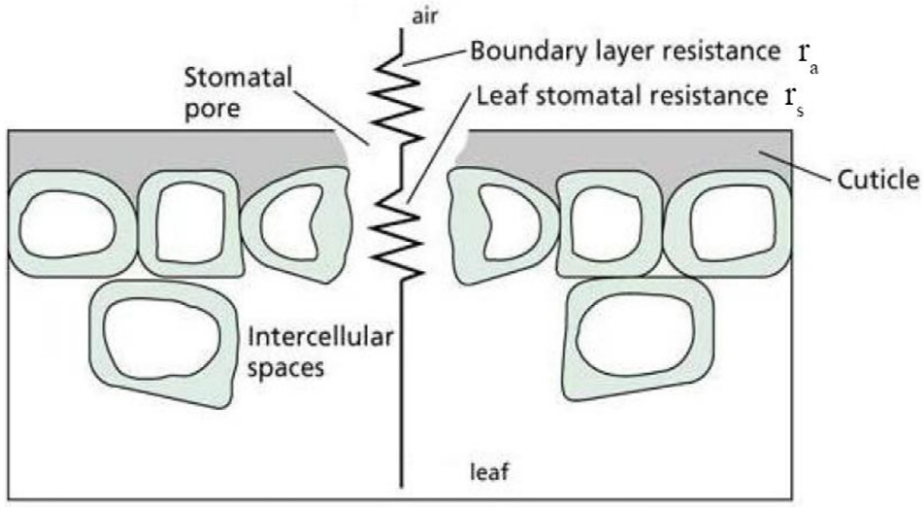
$$Q_\mathrm{lat} = \lambda_\mathrm{W} \cdot ET, \tag{2.14}$$

where ρ_r is the reflection coefficient of a crop, I_lighting is the photosynthetically active radiation irradiance, and CAC is the cultivation area coverage. The sensible heat flux, Q_sen, is derived from ρ_a the air density, c_p the specific heat of air, and T_a the ambient air temperature. The latent heat flux, Q_lat, is directly related to the transpiration rate ET multiplied by λ_W the latent heat of vaporization. Using the above equations, the leaf surface temperature is solved iteratively by ensuring energy balance. Once T_l is determined, the crop transpiration rate, ET, is recalculated for the source term of the absolute humidity.

[1] "Plant canopy structure is the spatial arrangement of the above-ground organs of plants in a plant community. Leaves and other photosynthetic organs on a plant serve both as solar energy collectors and as exchangers for gases. Stems and branches support these exchange surfaces in such a way that radiative and convective exchange can occur in an efficient manner. Canopy structure affects radiative and convective exchange of the plant community, so information about canopy structure is necessary for modelling these processes." 1. Campbell GS, Norman JM. The description and measurement of plant canopy structure. In: Russell G, Marshall B, Jarvis PG, eds. Plant Canopies: Their Growth, Form and Function. Society for Experimental Biology Seminar Series. Cambridge University Press; 1989:1-20



(a)



(b)

Figure 2.1: (a) Net radiation, sensible and latent heat balances of leaves and (b) Resistances to water vapor transfer between leaf and air [9].

Chapter 3

Existing implementations

3.1 codedSource

The following section will provide an overview of the relevant source code for the codedSource class, which can be used for the consideration of user defined source terms in OpenFOAM. The class implementation can be found under the OpenFOAM-v2112\src\fvOptions\sources\general\-codedSource directory and includes three file CodedFvSource.H, CodedFvSource.C, and CodedFv-Sources.C. The focus of this section is to explain how the code works as part of fvOptions.

First, the header file CodedFvSource.H declares the codedSource class, a template that integrates on-the-fly source terms. "On-the-fly" refers to dynamically adding, compiling, and running source terms from case configuration files (e.g, fvOptions) during simulation runtime, without the need for OpenFOAM's source code to be modified or pre-compiled. It uses header guards to prevent multiple inclusions of the header file and includes base classes as well provides base functionality for source terms on specific cell regions through including cellSetOption.H and allows runtime compil ation of user-provided code with codedBase.H. The class CodedSource is templated to support different types (scalar, vector, etc.), inherit from base fvOption for cell-specific options and provides hooks for dynamic code compilation as shown in Listing 3.1.

Listing 3.1: Class declaration

```
137 template<class Type>
138 class CodedSource
139 :
140     public fv::cellSetOption,
141     protected codedBase
```

As protected data, the class uses name_ to name the source term and strings to include the user-supplied code snippets (e.g., codeCorrect_, codeAddSup_, codeAddSupRho_, and codeConstrain_). The redirectOptionPtr_ is used to point to another fv::option for redirection of the source term. The CodedFvSource.C file implements the class functionality and its constructor, shown in Listing 3.2, defines and initializes the object with user inputs (name, modelType, dict, and mesh) and calls read() to read the dictionary and initialize fields and code snippets.

Listing 3.2: Constructor definition and initalization

```
139 template<class Type>
140 Foam::fv::CodedSource<Type>::CodedSource
141 (
142     const word& name,
143     const word& modelType,
144     const dictionary& dict,
145     const fvMesh& mesh
146 )
147 :
148     fv::cellSetOption(name, modelType, dict, mesh)
```

```
{
    read(dict);
}
```

Relevant member functions in this class are `correct()`, `addSup()`, and `constrain()`. In case of code implementation under `codeCorrect`, the `correct()` function is called to correct a field based on the user-defined function/implementation. `constrain()` allows setting constraints on the matrix under the `codeConstrain` shown in Listing 3.8. `addSup()` function is a overloaded function as it can be used for either compressible or incompressible flows depending of its input parameters. Since incompressible is the preferred flow for this report, the code implementation shown in Listing 3.3 must be considered. For compressible flow, an additional input parameter ρ is used. The `addSup()` function calls dynamically complied code (described under `codeAddSup` in `fvOptions`) to add a source term to the right side of the equation (`eqn`) for a given `field` (`fieldi`). A similar procedure is applied to the other two functions, `correct()` and `constrain()`.

Listing 3.3: `addSup()` implementation for incompressible flow

```
template<class Type>
void Foam::fv::CodedSource<Type>::addSup
(
    fvMatrix<Type>& eqn,
    const label fieldi
)
{
    DebugInfo
        << "fv::CodedSource<" << pTraits<Type>::typeName
        << ">::addSup for source " << name_ << endl;

    updateLibrary(name_);
    redirectOption().addSup(eqn, fieldi);
}
```

3.1.1 Application in OpenFOAM

The following example demonstrates the implementation of a custom source term for the temperature field `T` in `OpenFOAM` using the `fvOptions` framework. As for all header of an `OpenFOAM` file, metadata and structural information are provided. First, the header banner indicates which version of `OpenFOAM` is the following code compatible with (here `v2112`) and points to the official website for documentation and resources. The following block provides the dictionary metadata describing the file's format and purpose. It specifies that the file content is written in ASCII (human-readable text) format and belongs to the dictionary class, which is used for input configuration in `OpenFOAM`. Here the object is defined as `fvOptions` which allows the user to apply custom source terms, constraints, or other options to specific fields in `OpenFOAM` simulations.

Listing 3.4: `OpenFOAM` header including metadata and structural information

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      fvOptions;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
```

The fvOptions configuration includes several key components: codeInclude, codeCorrect, and codeAddSup, which are provided in the form of C++ code blocks within the OpenFOAM dictionary. The following example implementation allows dynamic compilation of C++ code during runtime, injecting a time-varying source term into the energy equation. The code begins by defining the type of the source term and specifying the target field. The type is set to scalarCodedSource, which means the source term applies to scalar fields, and the fields entry specifies the temperature field T. Furthermore, a unique name (temperatureSource) is assigned to the source term for identification as shown in Listing 3.5.

Listing 3.5: Type and field definition of custom source term

```
customTemperatureSource
{
    type            scalarCodedSource;

    scalarCodedSourceCoeffs
    {
        // Specify the field(s) to which the source term is applied
        fields          (T);

        // Unique name for the source term
        name            temperatureSource;
```

The codeInclude block Listing 3.6 can be used to include necessary C++ headers, but is not necessary to run codedSource. This additional function is part of the codedBase base class and it processes the dictionary inputs during the read() and perpare() phase. In this case, the cmath library is included to allow the use of mathematical functions such as std::sin or std::exp.

Listing 3.6: codeInclude block in fvOptions

```
        // Include external headers or libraries (e.g., for custom math or helper functions)
        codeInclude
        #{
            #include <cmath>    // For mathematical operations
        #};
```

In Listing 3.7, the codeCorrect block is optional and can be used to apply any field corrections. Here, it outputs a message to the terminal whenever it is executed:

Listing 3.7: codeCorrect block in fvOptions

```
        // Code to correct the field, if needed
        codeCorrect
        #{
            Pout << "**codeCorrect executed for T field**" << endl;
        #};
```

The codeAddSup block in Listing 3.8 contains the logic for adding a source term to the temperature field equation. The source term is defined as a function of time and cell volume V, given by $Q = \alpha \cdot t \cdot V$, where α is a user-defined coefficient, t is the simulation time, and V is the volume of each computational cell. Here, the current simulation time is accessed using time.value(), and the cell volumes are obtained from the mesh using mesh_.V(). The source term is added to the right-hand side of the matrix equation through eqn.source(). Finally, the codeConstrain block is defined as an optional constraint mechanism, printing a message when executed.

Listing 3.8: codeAddSup block in fvOptions

```
        // Code to add the source term to the temperature field
        codeAddSup
        #{
            // Access current simulation time
            const Time& time = mesh().time();

            // Access the volume of each cell
            const scalarField& V = mesh_.V();
```

```
49
50              // Access the source term field (right-hand side of the matrix equation)
51              scalarField& TSource = eqn.source();
52
53              // Define a time-varying source term: Q = alpha * t * V
54              const scalar alpha = 5.0; // Source term coefficient
55              TSource += alpha * time.value() * V;
56
57              Pout << "Adding source to T field at time: " << time.value() << endl;
58          #};
59
60          // Optional: Code to constrain the equation
61          codeConstrain
62          #{
63              Pout << "**codeConstrain executed for T field**" << endl;
64          #};
```

The implementation dynamically injects the specified source term into the temperature field T during runtime. The `codeInclude` block allows additional headers to be included, while the `codeAddSup` block defines the logic for the source term. Messages are printed to the terminal to confirm the execution of each block, ensuring transparency during simulation.

3.2 explicitPorositySource

The following code files explain the implementation of an explicit porosity source using `fvOptions` to add porosity effects into fluid simulations. Therefore, `explicitPorositySource` class is inherited from the `fv:cellSetOption`, which is a base class for applying source terms to cell regions (defined by a `cellSet`). All relevant files including the source code of this class can be found under the `OpenFOAM-v2112\src\fvOptions\sources\derived\explicitPorositySource` directory. The main function of this class is to add porosity resistance terms (momentum sinks) to momentum equations in specific zones. The pointer `autoPtr<porosityModel> porosityPtr` is used to encapsulate different models like Darcy-Forchheimer by pointing to a porosity model object that calculates resistance contributions. Therefore, the overloaded function `addSup()` is used to add resistance terms to the momentum equations in case of incompressible, compressible, or multiphase momentum equations. Thus, the function for incompressible flow is selected and described in more detail due to the use case proposed in this report. The function definition can be found in `explicitPorositySource.C` and is implemented as shown in Listing 3.9. It creates a temporary `fvMatrix<vector> porosityEqn` with the same size and dimensions as the target equation and calls `addResistance()` of the `porosityModel` class to compute the resistance contributions based on porosity parameters. At the end, it subtracts the resistance contributions (`porosityEqn`) from the target equation.

Listing 3.9: Function to add source terms for incompressible flow

```
77 void Foam::fv::explicitPorositySource::addSup
78 (
79     fvMatrix<vector>& eqn,
80     const label fieldi
81 )
82 {
83     fvMatrix<vector> porosityEqn(eqn.psi(), eqn.dimensions());
84     porosityPtr_->addResistance(porosityEqn);
85     eqn -= porosityEqn;
86 }
```

The function `addResistance()` modifies the momentum equation represented by the finite volume matrix (`fvVectorMatrix`) by adding porosity resistance terms. These terms account for the effects of a porous medium within specific cell zones of the computational domain, as defined by the porosity model. It ensures that resistance effects are incorporated into the governing equations during simulation. This function is defined in `porosityModel.C` as shown in Listing 3.10. Its

main components include checking if the porous region (defined by `cellZoneIDs_`) is non-empty, applying transformations to resistance coefficients based on the mesh geometry, and modifying the system matrix (`UEqn`) to account for resistance effects by invoking the `correct()` method. First, if no porous regions are defined (i.e., `cellZoneIDs_` is empty), the function exits early without modifying the momentum equation. If a porous region is defined, resistance coefficients are transformed to ensure that the resistance tensors (e.g., Darcy and Forchheimer coefficients) match the current mesh configuration, especially in cases involving moving or transformed meshes. At the end, the `correct()` function computes and applies the resistance contributions to the momentum equation. Both, `transformModel()` and `correct(UEqn)` are implemented in derived classes such as `DarcyForchheimer`. The complete source code of `porosityModel` and `DarcyForcheimer` class can be found under the `OpenFOAM-v2112\src\finiteVolume\cfdTools\porosity` directory.

Listing 3.10: Function for incompressible flow to add porosity resistance terms to `UEqn`

```
186 void Foam::porosityModel::addResistance(fvVectorMatrix& UEqn)
187 {
188     if (cellZoneIDs_.empty())
189     {
190         return;
191     }
192 
193     transformModelData();
194     this->correct(UEqn);
195 }
```

The `DarcyForchheimer` class extends the `porosityModel` base class and uses both Darcy and Forchheimer resistance terms. Its key data members are `dimensionedVector dXYZ_` and `fXYZ_` as well as `List<tensorField> D_` and `F_`, which represent user-specified Darcy or Forchheimer coefficient for spatial direction (viscous [m^{-2}] and inertial resistance [m^{-1}]) as well as converted Darcy and Forchheimer coefficient tensors for cells, respectively. The implemented member functions are `calcTransformModelData`, `correct` and `apply`. `calcTransformModelData` computes (`D_` and `F_`) for the porous region based on user-specified `dXYZ_` and `fXYZ_` components, which slightly differs depending on the selected porosityModel.

The `correct()` function modifies the momentum equation to account for the Darcy-Forchheimer resistance in the specified porous region. The function retrieves the velocity field (`U`), cell volumes (`V`), diagonal coefficients (`Udiag`), and source terms (`Usource`) from the matrix. Field names for density (`rhoName`), dynamic viscosity (`muName`), and kinematic viscosity (`nuName`) are prepared as shown in Listing 3.11 and an `if`-statement checks whether the momentum equation has force-like dimensions, which determines if density (ρ) should be included in the resistance calculation.

Listing 3.11: Retrieve required field values

```
188 void Foam::porosityModels::DarcyForchheimer::correct
189 (
190     fvVectorMatrix& UEqn
191 ) const
192 {
193     const volVectorField& U = UEqn.psi();
194     const scalarField& V = mesh_.V();
195     scalarField& Udiag = UEqn.diag();
196     vectorField& Usource = UEqn.source();
197 
198     word rhoName(IOobject::groupName(rhoName_, U.group()));
199     word muName(IOobject::groupName(muName_, U.group()));
200     word nuName(IOobject::groupName(nuName_, U.group()));
201 
202     if (UEqn.dimensions() == dimForce)
```

In case of a force-like dimension, it retrieves the density field (`rho`) and, if available, the dynamic viscosity (`mu`). If `mu` is not found, it computes dynamic viscosity using $\rho\nu$, where ν is the kinematic viscosity. In case of incompressible flows (`else`-condition), the function either uses the kinematic viscosity (`nu`), if available, or it computes it as `mu/rho`. Lastly, the resistance contributions are

computed and applied by calling the `apply()` function. Since the proposed tutorial case is based on incompressible flow, relevant lines show in Listing 3.12 will be executed.

Listing 3.12: Execute relevant `apply()` function inside `correct()` to update system equation `UEqn` for incompressible flow

```
219     else
220     {
221         if (mesh_.foundObject<volScalarField>(nuName))
222         {
223             const auto& nu = mesh_.lookupObject<volScalarField>(nuName);
224 
225             apply(Udiag, Usource, V, geometricOneField(), nu, U);
226         }
```

The core function `apply()` is a template function, it computes the resistance terms and modifies the system matrices as described in Listing 3.13 within the `DarcyForchheimerTemplates.C` file. The application of this function involves two key operations. It modifies the diagonal terms of the system matrix (`Udiag`), which corresponds to the coefficients of velocity in the momentum equation, and it updates the source term vector (`Usource`) to include the momentum sink caused by the porous medium. Therefore, `apply()` incorporates the Eq. (2.5) through direct application of the Darcy and Forchheimer terms and can be interpreted as followed: The outer loop iterates through all cell zones defined in the mesh, while the inner loop processes each cell within the zone. The tensor `Cd` is calculated as a combination of the viscous and inertial components: `Cd = mu[celli]` $\cdot$ `dZones[j] +` (`rho[celli]` $\cdot$ `mag(U[celli]))` $\cdot$ `fZones[j]`. The diagonal term `Udiag[celli]` is updated by adding the product of the volume factor `V` and the trace of the tensor Cd. The source term `Usource[celli]` is adjusted by subtracting the product of the volume factor `V` and the difference between `Cd` and the identity tensor scaled by its trace , dot-multiplied with the velocity vector `U`. This ensures that the momentum equation includes both viscous and inertial resistance effects from the porous medium.

Listing 3.13: Apply method for diagonal and source terms

```
30 template<class RhoFieldType>
31 void Foam::porosityModels::DarcyForchheimer::apply
32 (
33     scalarField& Udiag,
34     vectorField& Usource,
35     const scalarField& V,
36     const RhoFieldType& rho,
37     const scalarField& mu,
38     const vectorField& U
39 ) const
40 {
41     forAll(cellZoneIDs_, zoneI)
42     {
43         const tensorField& dZones = D_[zoneI];
44         const tensorField& fZones = F_[zoneI];
45 
46         const labelList& cells = mesh_.cellZones()[cellZoneIDs_[zoneI]];
47 
48         forAll(cells, i)
49         {
50             const label celli = cells[i];
51             const label j = this->fieldIndex(i);
52             const tensor Cd =
53                 mu[celli]*dZones[j] + (rho[celli]*mag(U[celli]))*fZones[j];
54 
55             const scalar isoCd = tr(Cd);
56 
57             Udiag[celli] += V[celli]*isoCd;
58             Usource[celli] -= V[celli]*((Cd - I*isoCd) & U[celli]);
59         }
60     }
61 }
```

3.2.1 Application in OpenFOAM

The explicitPorositySource class in OpenFOAM applies porosity resistance effects to the momentum equations within a specified region of the computational domain using the fvOptions framework. It models Darcy and Forchheimer resistance based on user-defined porosity coefficients, applied to designated regions. Within the system directory, the fvOptions file can be configured. The porosity model, coefficients, and region are specified in this file. Listing 3.14 illustrates a typical configuration using the DarcyForchheimer porosity model. It entails all key elements for the configuration and application of the explicitPorositySource function, such as:

- type explicitPorositySource: Specifies that this fvOption will apply an explicit porosity source.
- selectionMode: Specifies the mode of cell selection which can be all (use all cells in the computational domain, cellZone (use a given cellZone), cellSet (use a given cellSet), or points (use cells containing a given set of points. This handling of cell-set options in fvOptions is part of the intermediate abstract class Foam::fv::cellSetOption.
- cellZone: Specifies the porous region where the resistance terms are applied. This region must be pre-defined in the mesh as a cellZone.
- explicitPorositySourceCoeffs: Contains details about the porosity model and its coefficients.
- type DarcyForchheimer: Specifies the Darcy-Forchheimer model, which combines viscous (Darcy) and inertial (Forchheimer) resistance terms.
- d and f: Define the Darcy and Forchheimer coefficients, respectively, in tensor form for anisotropic resistance. For isotropic resistance, all tensor components can be equal.
- coordinateSystem: Defines the local coordinate system for the porous zone. This is useful for modeling anisotropic porosity, where resistance varies with direction.

Listing 3.14: explicitPorositySource application within system/fvOptions

```
porosity1
{
    // Type of the fvOption
    type            explicitPorositySource;
    active          true;

    explicitPorositySourceCoeffs
    {
        // Porosity model type
        type            DarcyForchheimer;
        // Region to apply - porosity zone
        selectionMode   cellZone;
        cellZone        porousZone;

        DarcyForchheimerCoeffs
        {
            // Darcy coefficients (viscous resistance)
            d  (50 50 50);

            // Forchheimer coefficients (inertial resitance)
            f  (2 2 2);

            // Coordinate system for anisotropic porosity
            coordinateSystem
            {
                origin  (0 0 0);
                e1      (1 0 0);
                e2      (0 1 0);
```

```
45            }
46        }
47    }
48 }
```

In the provided example implementation, the porosity source is applied only in cells defined by the `cellZone porousZone`. Hence, prior the application, a `cellZone` must be defined. This can be done using `topoSetDict` for example (see `topoSetDict-` file in A.2.2) It ensure that the `cellZone` is correctly defined in the mesh generation process.

Chapter 4

Implementations of water vapor dynamics

4.1 Water vapor implementation

While the original `buoyantBoussinesqSimpleFoam` solver only considers temperature-driven buoyancy, many real-world applications, such as humid airflow in HVAC systems or natural convection in moist air, require the inclusion of water vapor effects [10, 11]. In this section, the theoretical framework for incorporating water vapor, represented by its mass fraction `AH` (absolute humidity, non-dimensional), into the solver is presented. The inclusion of water vapor mass fraction in the `buoyantBoussinesqSimpleFoam` solver expands its applicability to humid airflow simulations. By solving the transport equation for `AH` alongside the momentum and temperature equations, the proposed solver will capture the interaction between temperature-driven and humidity-driven buoyancy forces. The extended solver includes four primary equations: the continuity equation, the momentum equation, the temperature transport equation, and a transport equation for the water vapor mass fraction.

4.1.1 Continuity equation

For incompressible flow under the Boussinesq approximation, the continuity equation is given by

$$\nabla \cdot \mathbf{U} = 0, \tag{4.1}$$

where $\mathbf{U}$ is the velocity field. This equation remains unchanged in the modified solver.

4.1.2 Momentum equation

The momentum equation is modified to account for buoyancy effects due to both temperature and water vapor. It is expressed as

$$\rho_0 \left(\mathbf{U} \cdot \nabla \mathbf{U}\right) = -\nabla p + \nabla \cdot \left(\mu \nabla^2 \mathbf{U}\right) + \rho_0 \mathbf{g} \left[\beta_{\mathrm{T}} (T - T_{\mathrm{Ref}}) + \beta_{\mathrm{AH}} (AH - AH_{\mathrm{Ref}})\right], \tag{4.2}$$

where ρ_0 is the reference density (constant under the Boussinesq approximation), p is the pressure field, μ is the dynamic viscosity, $\mathbf{g}$ is the gravitational acceleration vector, β_{T} is the thermal expansion coefficient, T_{Ref} is the reference temperature, β_{AH} is the water vapor expansion coefficient, and AH_{Ref} is the reference water vapor mass fraction.

The density-related term driving buoyancy is given by

$$\rho_k = 1 - \beta_T (T - T_{\mathrm{Ref}}) - \beta_{\mathrm{AH}} (AH - AH_{\mathrm{Ref}}), \tag{4.3}$$

which is used to compute the buoyancy forces.

4.1.3 Temperature equation

The temperature transport equation governs the distribution of thermal energy in the flow and is expressed as

$$\nabla \cdot (\rho \mathbf{U} T) - \nabla \cdot (\alpha \nabla T) = S_{\mathrm{T}}, \tag{4.4}$$

where $\alpha = \frac{k}{\rho c_{\mathrm{p}}}$ is the thermal diffusivity, with k as the thermal conductivity, ρ as density, and c_{p} as the specific heat capacity at constant pressure, and S_{T} represents source terms, such as radiation or user-defined energy inputs. Also this equation remains unchanged except for its coupling with the updated buoyancy term in the momentum equation.

4.1.4 Water vapor transport equation

The water vapor mass fraction AH is modeled as a passive scalar, governed by the advection-diffusion equation

$$\mathbf{U} \cdot \nabla AH - \nabla \cdot (D_{\mathrm{eff}} \nabla AH) = S_{\mathrm{AH}}, \tag{4.5}$$

where the effective diffusivity D_{eff} is

$$D_{\mathrm{eff}} = \frac{\nu}{Sc} + \frac{\nu_{\mathrm{t}}}{Sct}, \tag{4.6}$$

where ν is the molecular kinematic viscosity, ν_{t} is the turbulent kinematic viscosity, Sc is the Schmidt number for molecular diffusion, Sct is the turbulent Schmidt number, and S_{AH} represents source terms. This equation determines the distribution of `AH` and its contribution to buoyancy.

4.2 `buoyantBoussinesqSimpleFoam` modification

First, a new initial field `AH` will be implemented to account for the water vapor as absolute humidity (dimensionless) as shown in Listing 4.1 in the `createFields.H` header file. In addition, the buoyancy contribution of AH must be accounted for with an updated kinematic density for the buoyant force as described in Eq. (4.3). This implementation can be seen in Listing 4.2.

Listing 4.1: Implement new `AH` field in `createFields.H` file

```
45 Info<< "Reading field AH (absolute humidity - dimensionless)\n" << endl;
46 volScalarField AH
47 (
48     IOobject
49     (
50         "AH",
51         runTime.timeName(),
52         mesh,
53         IOobject::MUST_READ,
54         IOobject::AUTO_WRITE
55     ),
56     mesh
```

Listing 4.2: Updated kinematic density of buoyant force in `createFields.H` file

```
45 volScalarField rhok
46 (
47     IOobject
48     (
49         "rhok",
50         runTime.timeName(),
51         mesh
52     ),
53     // Buoyancy contribution of AH
54     1.0 - beta*(T - TRef) - betaAH*(AH - AHRef)
55 );
```

The water vapor transport equation, described in Eq. (4.5), is implemented as shown in Listing 4.3 and derives the distribution of water vapor mass fraction (`AH`). The equation is governed by the advection by the velocity field and effective diffusivity D_{eff} (via Schmidt numbers) described in Eq. (4.6) and implemented from `Line 4-10` of the following code. The solution of the governing equation determines how water vapor interacts with the flow and buoyancy.

Listing 4.3: New transport equation for `AH` in `AHEqn.H` file

```
{
  volScalarField DYEff("DYEff", turbulence->nu()/Sc + turbulence->nut()/Sct);

  fvScalarMatrix AHEqn
  (
      fvm::div(phi, AH)
    - fvm::laplacian(DYEff, AH)
    ==
      fvOptions(AH)
  );

  AHEqn.relax();

  fvOptions.constrain(AHEqn);

  solve(AHEqn);

  fvOptions.correct(AH);

}
```

All relevant additional parameter values for solving the water vapor related equations will be provided through the `readTransportProperties.H` file in `OpenFOAM`. Therefore, parameters such as β_{AH} - water vapor expansion coefficient, AH_{Ref} the reference water vapor mass fraction, and both Schmidt numbers for water vapor, laminar and turbulent (Sc and Sct, respectively) are added to the `readTransportProperties.H` file as shown in Listing 4.4 and thus, water properties are added to the solver loop. Lastly, the newly created `AHEqn.H` file must be included in the source code file of the solver (e.g., `buoyantBoussinesqSimpleFoam.C`). Therefore, a new line between `#include "TEqn.H"` and `#include "pEqn.H"` is added, including `#include "AHEqn.H"` as illustrated in Listing 4.5. The new solver can be compiled now.

Listing 4.4: Read additional parameter with `readTransportProperties.H` file to account for water vapor

```
dimensionedScalar betaAH
(
    "betaAH",
    dimless,
    laminarTransport
);

// Reference water vapor mass fraction
dimensionedScalar AHRef("AHRef", dimless, laminarTransport);

// Laminar and Turbulent Schmidt number for water vapor transport
dimensionedScalar Sc("Sc", dimless, laminarTransport);
dimensionedScalar Sct("Sct", dimless, laminarTransport);
```

Listing 4.5: Modified `buoyantBoussinesqSimpleFoam.C` file to account for transport of water vapor

```
            #include "TEqn.H"
            #include "AHEqn.H"
            #include "pEqn.H"
```

4.3 Utilization of fvOptions

The following section describes in detail the implementation of the crop transpiration model described in Section 2.3. The modified `fvOptions` file includes equations and relevant information about the porous media zone, as well as constant input parameters for add to the relevant source term and updating effected temperature fields. Crop transpiration is only applied in regions of a porous media, representing a crop region. Therefore, the proposed application example of an `explicitPorositySource` implementation is used from Section 3.2.1. The presented crop transpiration model and energy balance in Section 2.3 is based on Graamans et al. [5], while the implementation in Section 4.3, follows a python implementation[1] of Graamans et al. described theory.

The scalar coded source block implements a custom scalar source term using user-defined C++ code. The key entries include: The `type` is specified as `scalarCodedSource`, indicating a user-defined source term for a scalar field. It is set to be `active`, and the name of the source term is `AHSource`. The `scalarCodedSourceCoeffs` section specifies the configuration of this custom source. The `selectionMode` is defined as `cellZone`, meaning the source applies to the `porousZone` and the field affected by this source term is `AH`.

Listing 4.6: Definition of dictionary for scalar coded source block in `fvOptions`

```
51 AHSource
52 {
53     type            scalarCodedSource;
54     active          true;
55     name            AHSource;
56
57     scalarCodedSourceCoeffs
58     {
59         selectionMode  cellZone;
60         cellZone       porousZone;
61         fields         (AH);
```

The `codeInclude` section defines helper functions and constants related to physical processes such as sensible heat exchange, latent heat flux, and surface temperature calculations. It includes functions for vapor resistance, net radiation, and surface temperature, as well as temperature scale conversions and energy balance calculations. Constants for crop properties, atmospheric and fluid properties are also defined here, which are implemented at the beginning for this code block and consider values as described in Table 4.1. Constants are valid under standard reference conditions corresponding to a temperature of 20°C (293.15 K) and an atmospheric pressure of 101325 Pa.[2] Functions are declared and described from `Line 88-248` in Listing 4.7 and defined from `Line 254-428` (see Listing A.2.2 for the complete implementation).

[1] https://github.com/linucks/fu_evapotranspiration
[2] https://www.engineeringtoolbox.com/

Table 4.1: Constant input parameters

Name	Value	Unit	Source
Leaf Area Index	3.0	–	[5]
Mean Leaf Diameter	0.1	m	[5, 2]
Reflection Coefficient	0.05	–	[5]
Photosynthetic Photon Flux Density	200.0	μmol/m^2/s	[5]
Cultivation Area Coverage	0.90	–	[5, 2]
Atmospheric Pressure	101325	Pa	see 2
Heat Capacity of Air	1005.0	J/kg/K	see 2
Heat Capacity of Water	1859.0	J/kg/K	see 2
Density of Air	1.2041	kg/m^3	see 2
Density of Water	998.0	kg/m^3	see 2
Latent Heat of Water	2264705	J/kg	see 2
Molar Mass of Water	18.01528	g/mol	see 2
Psychrometric Constant	65.0	Pa/K	see 2
Ideal Gas Constant	8.3145	J/mol/K	see 2
Zero Degrees in Kelvin	273.15	K	see 2

Listing 4.7: Declaration and description of functions

```
88              //- Converts a temperature from Kelvin to Celsius.
89              // @param tempKev Temperature in Kelvin.
90              // @return Temperature in degree Celsius.
91              double convertKelvinToCelsius
92              (
93                  double tempKev
94              );
95
96              //- Calculates the relative humidity
97              // This is based on the saturation vapor pressure p_vs (Pa)
98              // as the Tetens equation equation is valid for
99              // temperature between 0 C  and 50 C )
100             // @param tempAir Temperature of the air in degree Celsius
101             // @param absolute_humidity Water vapor mass fraction (kg/kg)
102             // @return Relative humidity in percentage
103             double calcRelativeHumidity
104             (
105                 double tempAir,
106                 double absolute_humidity
107             );
108
109             //- Calculates the aerodynamic resistance
110             // @param l_leaf mean leaf diameter (m)
111             // @param lai leaf area index
112             // @param u local air speed (m/s)
113             // @return s/m
114             double calcVapourResistance
115             (
116                 double l_leaf,
117                 double u,
118                 double lai
119             );
120
121             //- Calculates the net radiation on the surface.
122             // @param ppfd Photosynthetic photon flux density (light intensity in  mol /m  /s).
123             // @param reflectionCoefficient Reflection coefficient of the surface.
124             // @param cultivationAreaCoverage Fraction of surface covered by vegetation or material.
125             // @return Net radiation on the surface in W/m  .
126             double calcNetRadiation
127             (
128                 double ppfd,
129                 double reflectionCoefficient,
```

```
130                 double cultivationAreaCoverage
131             );
132
133             //- Calculates the radiation from lighting based on PPFD.
134             // @param ppfd Photosynthetic photon flux density (light intensity in  mol /m  /s).
135             // @return Radiation from lighting in W/m  .
136             // @throws std::runtime_error if PPFD value is unsupported.
137             double calcLightingRadiation
138             (
139                 double ppfd
140             );
141
142             //- Calculates the sensible heat exchange between the surface and air.
143             // @param tempAir Air temperature in degrees Celsius.
144             // @param tempSurface Surface temperature in degrees Celsius.
145             // @param lai Leaf area index (area of leaves per unit ground area).
146             // @param vapourResistance Vapor resistance in s/m.
147             // @return Sensible heat exchange in W/m  .
148             double calcSensibleHeatExchange
149             (
150                 double tempAir,
151                 double tempSurface,
152                 double lai,
153                 double vapourResistance
154             );
155
156             //- Calculates the latent heat flux from evaporation or transpiration.
157             // @param tempAir Air temperature in degrees Celsius.
158             // @param tempSurface Surface temperature in degrees Celsius.
159             // @param relativeHumidity Relative humidity as a percentage.
160             // @param ppfd Photosynthetic photon flux density (light intensity in  mol /m  /s).
161             // @param lai Leaf area index (area of leaves per unit ground area).
162             // @param vapourResistance Vapor resistance in s/m.
163             // @return Latent heat flux in W/m  .
164             double calcLatentHeatFlux
165             (
166                 double tempAir,
167                 double tempSurface,
168                 double relativeHumidity,
169                 double ppfd,
170                 double lai,
171                 double vapourResistance
172             );
173
174             //- Calculates the vapor concentration in the air.
175             // @param tempAir Air temperature in degrees Celsius.
176             // @param relativeHumidity Relative humidity as a percentage.
177             // @return Vapor concentration of air in kg/m  .
178             double calcVapourConcentrationAir
179             (
180                 double tempAir,
181                 double relativeHumidity
182             );
183
184             //- Calculates the saturated vapor concentration of air at a given temperature.
185             // @param tempAir Air temperature in degrees Celsius.
186             // @return Saturated vapor concentration of air in kg/m  .
187             double calcSaturatedVapourConcentrationAir
188             (
189                 double tempAir
190             );
191
192             //- Calculates the saturated vapor pressure of air at a given temperature.
193             // @param tempAir Air temperature in degrees Celsius.
194             // @return Saturated vapor pressure in Pascals.
195             double calcSaturatedVapourPressureAir
196             (
197                 double tempAir
```

```
198             );
199
200             //- Calculates the vapor concentration at the surface.
201             // @param tempAir Air temperature in degrees Celsius.
202             // @param tempSurface Surface temperature in degrees Celsius.
203             // @param vapourConcentrationAir Vapor concentration of air in kg/m  .
204             // @return Vapor concentration at the surface in kg/m  .
205             double calcVapourConcentrationSurface
206             (
207                 double tempAir,
208                 double tempSurface,
209                 double vapourConcentrationAir
210             );
211
212             //- Calculates the psychrometric constant epsilon at a given air temperature.
213             // @param tempAir Air temperature in degrees Celsius.
214             // @return Psychrometric constant epsilon (dimensionless)
215             double calcEpsilon
216             (
217                 double tempAir
218             );
219
220             //- Calculates stomatal resistance based on light intensity.
221             // @param ppfd Photosynthetic photon flux density (light intensity in  mol /m  /s).
222             // @return Stomatal resistance in s/m.
223             double calcStomatalResistance
224             (
225                 double ppfd
226             );
227
228             //- Solves for surface temperature based on energy balance.
229             // Root finding algorithm, a bisection method, is used to
230             // iteratively find the solution.
231             // @param tempAir Air temperature in degrees Celsius.
232             // @param ppfd Photosynthetic photon flux density (light intensity in  mol /m  /s).
233             // @param relativeHumidity Relative humidity as a percentage.
234             // @param lai Leaf area index (area of leaves per unit ground area).
235             // @param vapourResistance Vapor resistance in s/m.
236             // @param reflectionCoefficient Reflection coefficient of the surface.
237             // @param cultivationAreaCoverage Fraction of surface covered by vegetation or material.
238             // @return Surface temperature in degrees Celsius
239             double calcTempSurface
240             (
241                 double tempAir,
242                 double ppfd,
243                 double relativeHumidity,
244                 double lai,
245                 double vapourResistance,
246                 double reflectionCoefficient,
247                 double cultivationAreaCoverage
248             );
```

The codeAddSup block in the provided fvOptions file is responsible for implementing a custom scalar source term in OpenFOAM into the right-hand side of the governing equations. This block adds the necessary modifications to the fields representing absolute humidity (AH) and temperature (T) for cells located within a defined porous zone. The primary goal is to model the effects of water vapor addition through transpiration and the associated thermal changes.

The first step in this block is accessing the necessary fields from the mesh, visible from Line 440-447 in Listing 4.11. The AH field represents absolute humidity, the U field represents velocity, and the T field represents temperature. The temperature field is fetched as a modifiable reference to allow adjustments reflecting thermal effects. A scalar source term, AHSource, is also prepared for modification as part of the governing equation for absolute humidity. The porous zone is identified by accessing the cellZones of the mesh and determining the zone corresponding to porousZone. Thus, the code iterates over all cells in the computational domain, and calculations are applied only to cells that belong to this porous zone.

For each cell in the porous zone, the relevant variables are initialized. The temperature in Kelvin (T[cellID]) is converted to degrees Celsius using the convertKelvinToCelsius() function and the absolute humidity (AH[cellID]) is converted to relative humidity using the calcRelative-Humidity() function, which is based on the Tetens equation for saturation vapor pressure. The vapor resistance, representing the aerodynamic resistance r_a (Eq. (2.9)) to vapor flow, is calculated using the calcVapourResistance() function, which depends on the leaf size, leaf area index, and local air velocity. Its definition can be see in Listing 4.8.

Listing 4.8: Definition of calcVapourResistance() functions

```
275             double calcVapourResistance
276             (
277                 double l_leaf,
278                 double u,
279                 double lai
280             )
281             {
282                 return 350* pow(l_leaf/u,0.5) * pow(lai,-1);
283             };
```

The leaf surface temperature of the porous region is calculated using the calcTempSurface() function. This function solves the crop energy balance equation (Eq. 2.11) that accounts for net radiation, sensible heat flux, and latent heat flux by using the root-finding algorithm. The bisection method is applied, a numerical technique, for solving equations of the form $f(x) = 0$. Therefore, an initial range of $[T_{air} - limit, T_{air} + limit]$ is chosen for the surface temperature and the midpoint of the range is iteratively refined based on the energy balance equation until the tolerance (10^{-6}) is satisfied. The functions definition can be seen in the following Listing 4.9.

Listing 4.9: Definition of calcTempSurface() functions

```
392
393             double calcTempSurface
394             (
395                 double tempAir,
396                 double ppfd,
397                 double relativeHumidity,
398                 double lai,
399                 double vapourResistance,
400                 double reflectionCoefficient,
401                 double cultivationAreaCoverage
402             )
403             {
404                 double netRadiation = calcNetRadiation
405                         (ppfd, reflectionCoefficient, cultivationAreaCoverage);
406                 auto calcEnergyBalance = [&](double tempSurface)
407                 {
408                     double sensibleHeat = calcSensibleHeatExchange
409                         (tempAir, tempSurface, lai, vapourResistance);
410                     double latentHeat = calcLatentHeatFlux
```

```
411                         (tempAir, tempSurface, relativeHumidity, ppfd, lai, vapourResistance);
412                     return netRadiation - sensibleHeat - latentHeat;
413                 };
414
415                 // Root finding using bisection method
416                 double limit = 10.0;
417                 double xa = tempAir - limit;
418                 double xb = tempAir + limit;
419
420                 double tol = 1e-6;
421                 while (std::fabs(xb - xa) > tol)
422                 {
423                     double xm = (xa + xb) / 2.0;
424                     if (calcEnergyBalance(xm) * calcEnergyBalance(xa) < 0) {
425                         xb = xm;
426                     } else {
427                         xa = xm;
428                     }
```

The net radiation is determined using a combination of light intensity (PPFD), reflection coefficients, and coverage factors, described as Eq. (2.12) and implemented in `calcNetRadiation()`. Sensible heat exchange between the air and the surface is computed using the `calcSensibleHeatExchange()` function, while latent heat flux due to water vapor transport is calculated using the `calcLatentHeatFlux()` function, following Eq. (2.13) and (2.14). The three implemented mathematical equations in `fvOptions` as C++ code can be seen in Listing 4.10

Listing 4.10: Implementation of R_{net} Q_{sen} and Q_{lat} in `fvOptions`

```
299             double calcNetRadiation
300             (
301                 double ppfd,
302                 double reflectionCoefficient,
303                 double cultivationAreaCoverage
304             )
305             {
306                 double lightingRadiation = calcLightingRadiation(ppfd);
307                 return (1.0 - reflectionCoefficient) * lightingRadiation * cultivationAreaCoverage;
308             }
309
310             double calcSensibleHeatExchange
311             (
312                 double tempAir,
313                 double tempSurface,
314                 double lai,
315                 double vapourResistance
316             )
317             {
318                 return lai * HEAT_CAPACITY_OF_AIR * DENSITY_OF_AIR * \
319                 (tempSurface - tempAir) / vapourResistance;
320             }
321
322             double calcLatentHeatFlux
323             (
324                 double tempAir,
325                 double tempSurface,
326                 double relativeHumidity,
327                 double ppfd,
328                 double lai,
329                 double vapourResistance
330             )
331             {
332                 double vapourConcentrationAir = calcVapourConcentrationAir(tempAir, relativeHumidity);
333                 double vapourConcentrationSurface = calcVapourConcentrationSurface(tempAir,
    tempSurface, vapourConcentrationAir);
334                 double stomatalResistance = calcStomatalResistance(ppfd);
335                 return lai * (LATENT_HEAT_WATER / 1000.0) * ((vapourConcentrationSurface -
    vapourConcentrationAir) / (stomatalResistance + vapourResistance));
```

Based on the calculated latent heat flux, the transpiration rate, denoted as ET, is calculated as the latent heat term Q_{lat} divided by the latent heat of vaporization and the height of the canopy [3]. This rate represents the water vapor added to the porous zone and is normalized by the density of water. The resulting value is subtracted from the scalar source term `AHSource`, effectively adding a water vapor source to the absolute humidity equation. The described calculation can be found from `Line 479-482` in Listing 4.11. In addition, the temperature field is updated to reflect the heat transfer effects, following `Line 484-490` in Listing 4.11. The change in temperature (ΔT) is computed using the formula

$$\Delta T = \frac{Q_{\text{sen}}}{m_{\text{air}} \cdot c_{\text{moist}}} \tag{4.7}$$

where Q_{sen} is the sensible heat flux, m_{air} is the mass of moist air, and c_{moist} is the specific heat capacity of moist air. Listing 4.11 shows the `codeAddSup` block which is crucial for simulating the interaction between humidity and heat transfer dynamics in porous zones, such as those encountered in greenhouse or environmental flow modeling. It captures the physical processes associated with transpiration and heat exchange effectively, by iteratively updating the absolute humidity and temperature fields.

Listing 4.11: Calculation of `ET` as additional source term in `AH` and a ΔT for `T` field correction in `codeAddSup`

```
436         #{
437             Pout<< "**codedCorrect**" << endl;
438         #};
439 
440         codeAddSup
441         #{
442             // Provide field values
443             const volScalarField& AH = mesh().lookupObject<volScalarField>("AH");
444             const volVectorField& U = mesh().lookupObject<volVectorField>("U");
445             // Allow modification of T
446             volScalarField& T = const_cast<volScalarField&>(mesh().lookupObject<volScalarField>("T"));
447             // Reference to the source term in the equation
448             scalarField& AHSource = eqn.source();
449             // Access the porous zone ID
450             const label porousZoneID = mesh().cellZones().findZoneID("porousZone");
451 
452             // Loop through the cells in mesh
453             forAll(AH, cellID)
454             {
455                 // Execute code if cell is in porousZone
456                 if (mesh().cellZones().whichZone(cellID) == porousZoneID)
457                 {
458                     // Get field values for cellID
459                     double tempInKelvin = T[cellID];
460                     double absolute_humidity = AH[cellID];
461                     vector Ucell = U[cellID];
462                     scalar u = mag(Ucell);
463                     // Convert K into  C
464                     double tempAir = convertKelvinToCelsius(tempInKelvin);
465                     // Convert AH into RH
466                     double relativeHumidity = calcRelativeHumidity(tempAir, absolute_humidity);
467                     // Calculate vapour/aerodynamic resistance
468                     double vapourResistance = calcVapourResistance(l_leaf, u, lai);
469 
470                     // Calculate surface temperature - root-finding method
471                     double tempSurface = calcTempSurface(tempAir, ppfd, relativeHumidity, lai,
472                                         vapourResistance, reflectionCoefficient,
    cultivationAreaCoverage);
473 
474                     // Calculate sensible heat exchange and latent heat flux
475                     double sensibleHeatExchange = calcSensibleHeatExchange(tempAir, tempSurface, lai,
476                                         vapourResistance);
477                     double latentHeatFlux = calcLatentHeatFlux(tempAir, tempSurface, relativeHumidity,
478                                         ppfd, lai, vapourResistance);
```

```
479
480                     // Calculate the rate of mass flux per volume dervied from transpiration rate (ET)
481                     scalar ET = latentHeatFlux / LATENT_HEAT_WATER / l_leaf;
482                     // Normalized the mass flux to a flux per time-unit
483                     ET = ET / DENSITY_OF_WATER;
484                     // Add as a source term to the right-hand side of the transport equation
485                     AHSource[cellID] -= ET;
486
487                     // Calculate the specific heat capacity of moist air
488                     scalar specific_heat_moist_air = HEAT_CAPACITY_OF_AIR * \
489                                     (1 - AH[cellID]) + (AH[cellID] * HEAT_CAPACITY_OF_WATER);
490                     // Calculate the mass of moist air
491                     scalar m_air = (ATMOSPHERIC_PRESSURE / (287 * tempInKelvin)) / (1 + 0.61 * AH[
    cellID]);
492                     scalar delta_T = sensibleHeatExchange / (m_air * specific_heat_moist_air);
493                     // Calculate temperature increase delta_T
```

The `codeCorrect`, in Listing 4.12, and `codeConstrain`, in Listing 4.13, blocks are placeholder for applying field corrections or constraints but currently only output a message for runtime monitoring.

Listing 4.12: `codeCorrect`

```
431                 return (xa + xb) / 2.0;
432             }
433         #};
```

Listing 4.13: `codeConstrain`

```
495                 }
496             }
497         #};
```

Chapter 5

Instructions for solver application

A modified version of the hotRoom tutorial case will be described including inlet and outlet condition to provide an example application of the newly developed solver and crop transpiration model implementation using `fvOptions`. This chapter is divided into two section. First, instructions are provided to modify the available `bouyantBoussinesqSimpleFoam` solver to account for the transport of water vapor. In the second part, an instruction is provided to set up a the test case, run the simulation and illustrate relevant results.

5.1 Solver modification

The following instructions describe how to create a new OpenFOAM solver named `cropTranFoam`, derived from the existing `buoyantBoussinesqSimpleFoam` solver. Open a terminal, source your `OpenFoam-v2112` application and execute the following commands. The provided commands navigate to the OpenFOAM project user directory and create the `heatTransfer` directory under `applications/solvers`. Then a copy of the `buoyantBoussinesqSimpleFoam` solver directory to a new directory named `cropTranFoam` is created. `Make/files` is modified to ensure the solver builds in the user application binary directory. Then all occurrences of `buoyantBoussinesqSimpleFoam` are replaced with `cropTranFoam` and the primary solver file `buoyantBoussinesqSimpleFoam.C` is renamed to `cropTranFoam.C`. The final lines add the new header file `AHEqn.H` in the main solver file and create a new (empty) file named `AHEqn.H` for custom modifications.

```
cd $WM_PROJECT_USER_DIR
mkdir -p applications/solvers/heatTransfer
cd applications/solvers/heatTransfer
cp -r $FOAM_APP/solvers/heatTransfer/buoyantBoussinesqSimpleFoam cropTranFoam
cd cropTranFoam
sed -i s/FOAM_APPBIN/FOAM_USER_APPBIN/g Make/files
find . -type f -exec sed -i 's/buoyantBoussinesqSimpleFoam/cropTranFoam/g' {} +
mv buoyantBoussinesqSimpleFoam.C cropTranFoam.C
sed -i '93a \ \ \ \ \ \ \ \ \ \ \ \ #include "AHEqn.H"' cropTranFoam.C
touch AHEqn.H
```

Now the user can open the relevant solver files (e.g., `cropTranFoam.C`, `Make/files`, `AHEqn.H`, etc.) using your preferred editor (e.g., `vim`, `VS Code`). It must be ensured that the content of each solver file aligns with the provided implementation in the Appendix A.1. Therefore, copy the provided code from Appendix A.1 into your newly created `cropTranFoam` solver files. After making all necessary modifications, navigate to the solver directory, source `OpenFOAM-v2112` application and compile the solver. Ensure no errors are encountered during compilation and the new solver executable `cropTranFoam` should be available in the user binary directory.

Notes to modified files

The files requiring updates may include:

- `Make/files`
- Main solver file: `cropTranFoam.C`
- Custom header file: `AHEqn.H`, `TEqn.H`, `createFields.H` and `readTransportProperties.H`

5.2 Running a tutorial case

Execute the provided code below in a terminal after you successfully compiled the `cropTranFoam` solver as instructed in the previous Section 5.1 and sourced your `OpenFOAM-v2112` application. After successful execution of the code, use a preferred editor to open all relevant files mentioned the appendix A.2, modify and/or overwrite the content of these files with the provided code in the appendix.

```
run
cp -r $FOAM_TUTORIAL/heatTransfer/buoyantBoussinesqSimpleFoam/hotRoom testCropTranFoam
cd testCropTranFoam
sed -i s/setFields/topoSet/g Allrun
rm system/setFieldsDict
touch system/topoSetDict
touch system/fvOptions
```

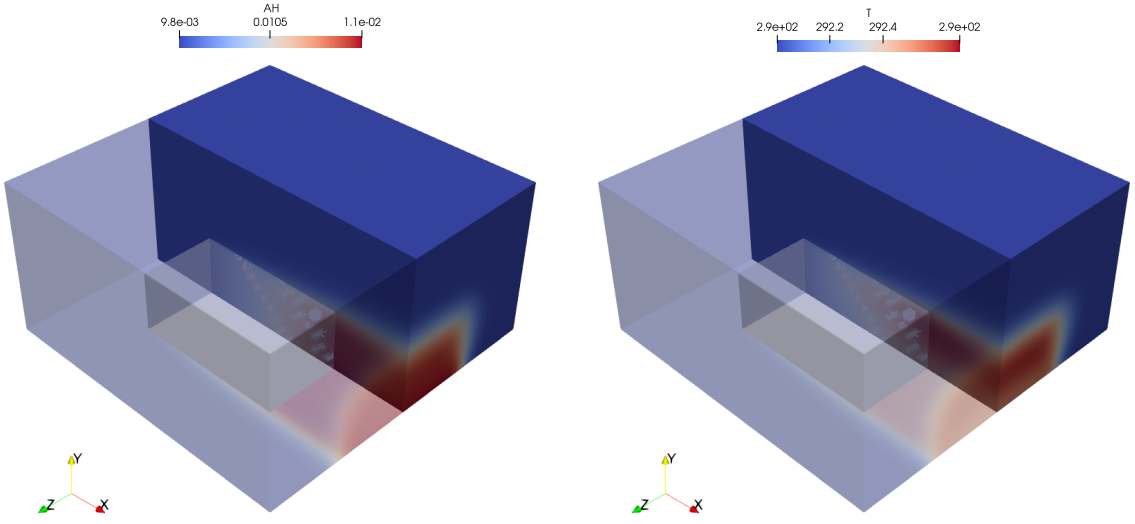
Once the case files are aligned with the shown code in the Appendix you can execute the following code to run the simulation.

```
./Allrun
paraFoam
```

Visualization of results

1. After the simulation finished and open ParaView either via the command `paraFoam` or as a normal application. Load the newly created `results.foam` file to view the results.
2. In the **Properties** panel, select the options:
   - **Read Zones**
   - **Copy Data to Cell Zones**

   and then load (**Apply**) the results.
3. Navigate to the **Styling** section and change the opacity of the results to **0.3**.
4. Apply the **Clip** filter and configure the **Plane Parameters** to show the plane on **Z Normal**.
5. Right-click in the **Pipeline Browser** and select **Show All**.
6. For the two sources in the pipeline, modify the displayed field to either **Absolute Humidity (AH)** or **Temperature (T)** and run the simulation.
7. Observe how the temperature and absolute humidity increases in the region where the porous media zone is located (you may select **Rescale to Data Range**). The expected results are shown in Figure 5.1.



(a) `AH` field results

(b) `AH` field results

Figure 5.1: Expected simulation results for absolute humidity (`AH` in kg/kg) in 5.1a and temperature (`T` in K) in 5.1b

Notes to modified files

The files requiring updates may include:

- `0.orig` directory: `AH, T, U, alphat, epsilon, k, nut, p, p_rgh`
- `system` directory: `blockMeshDict`, `controlDict`, `fvOptions`, `fvSchemes`, `fvSolution`, `topoSetDict`
- `constant` directory: `transportProperties`

Chapter 6

Conclusion and future work

This study successfully modified the `buoyantBoussinesqSimpleFoam` solver, enhancing its capability to simulate physiological and environmental dynamics within controlled agricultural environments. The incorporation of water vapor transport equations, along with modifications to account for heat and mass transfer coupling, enables the solver to capture the interplay between crop processes and their surroundings. The theoretical underpinnings, implementation details, and practical application of the enhanced solver have been systematically presented. Verification through a tutorial case highlights the effectiveness of the proposed modifications. It demonstrated the solver's capability to model temperature and humidity dynamics within a porous media, representing the crop canopy by using `fvOptions`. `OpenFOAM`'s open-source platform offered a flexible and adaptable framework to demonstrate the application of CFD simulations for agricultural modeling in environmental controlled systems such as vertical farms and greenhouses.

Future research should focus on the following directions to further enhance and expand the utility of the modified solver and proposed simulation case. The developed `fvOptions` file can be extended to incorporate a radiation heat model (e.g., a discrete ordinate model) instead of the current fixed net radiation approach, which relies on a constant PPFD value. This enhancement will enable the model to account for varying levels of transpiration based on the height of the crop region, improving the accuracy of simulations. Additionally, time-dependent environmental parameters, such as fluctuating light intensities and temperature profiles, should be explored to simulate diurnal and seasonal variations more effectively. Transient modeling capabilities can also be integrated to study the dynamic responses of crops under changing conditions, providing more realistic and comprehensive results. However, before these advancements are implemented, the presented simulation case must be experimental validated against real-world crop transpiration data to ensure accuracy and reliability.

Bibliography

[1] L. Kang and T. Van Hooff, "Numerical evaluation and optimization of air distribution system in a small vertical farm with lateral air supply," vol. 17, p. 100304. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S2666165923001862

[2] L. Kang, Y. Zhang, M. Kacira, and T. Van Hooff, "CFD simulation of air distributions in a small multi-layer vertical farm: Impact of computational and physical parameters," vol. 243, pp. 148–174. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1537511024001053

[3] Y. Zhang and M. Kacira, "Analysis of climate uniformity in indoor plant factory system with computational fluid dynamics (CFD)," vol. 220, pp. 73–86. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1537511022001210

[4] H. Fang, K. Li, G. Wu, R. Cheng, Y. Zhang, and Q. Yang, "A cfd analysis on improving lettuce canopy airflow distribution in a plant factory considering the crop resistance and leds heat dissipation," *Biosystems Engineering*, vol. 200, pp. 1–12, 2020.

[5] L. Graamans, A. Van Den Dobbelsteen, E. Meinen, and C. Stanghellini, "Plant factories; crop transpiration and energy balance," vol. 153, pp. 138–147. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0308521X16306515

[6] T. L. Bergman and A. S. Lavine, *Fundamentals of heat and mass transfer*, eighth edition ed. John Wiley & Sons.

[7] D. A. Nield and A. Bejan, *Convection in porous media*, 2013, vol. 9781461455.

[8] J. D. Wilson, "Numerical studies of flow through a windbreak," *Journal of Wind Engineering and Industrial Aerodynamics*, vol. 21, no. 2, pp. 119–154, 1985. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0167610585900017

[9] H. Fatnassi, P. E. Bournet, T. Boulard, J. C. Roy, F. D. Molina-Aiz, and R. Zaaboul, "Use of computational fluid dynamic tools to model the coupling of plant canopy activity and climate in greenhouses and closed plant growth systems: A review," *Biosystems Engineering*, vol. 230, pp. 388–408, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1537511023000983

[10] E. P. Chassignet, C. Cenedese, and J. Verron, *Buoyancy-driven flows*. Cambridge university press.

[11] D. J. Tritton, *Physical Fluid Dynamics*. Springer Netherlands. [Online]. Available: http://link.springer.com/10.1007/978-94-009-9992-3

Study questions

How to use it:

- What must be defined in `fvOptions` that a custom source term for U field is added under `codedAddSup`?
- Which `code` blocks must be provided in `fvOptions` to use `codedSource` class?
- Which modes of cell selection are available and how are they used?
- When will you define different X, Y, Z values for the Darcy and Forchheimer coefficients?

The theory of it:

- When is the density assumed not to be constant and how is the buoyant force approximated?
- What resistance forces represent the Darcy-Forchheimer equation?
- Which parameter must be solved first to derive the added mass source of water vapor and how?

How it is implemented:

- What header files are included in `CodedFvSource.H` and why?
- Why is the *addSup*() function in `explicitPorositySource` class overloaded?
- What function defines and uses the described Eq. (2.5) and which source file includes the relevant code?

How to modify it:

- Which file(s) must be created and/or modified to account for water vapor using the `buoyantBoussinesqSimpleFoam` solver?
- Why is `T` defined differently compared to `AH` and `U` under the `codedAddSup` block in Listing 4.11?
- Why and how is the root-finding algorithm implemented?

Appendix A

Developed codes

A.1 `cropTranFoam` solver

Water vapor transport equation `AHEqn.H`

```
{
  volScalarField DYEff("DYEff", turbulence->nu()/Sc + turbulence->nut()/Sct);

  fvScalarMatrix AHEqn
  (
      fvm::div(phi, AH)
    - fvm::laplacian(DYEff, AH)
    ==
      fvOptions(AH)
  );

  AHEqn.relax();

  fvOptions.constrain(AHEqn);

  solve(AHEqn);

  fvOptions.correct(AH);

}
```

Temperature equation `TEqn.H`

```
{
    alphat = turbulence->nut()/Prt;
    alphat.correctBoundaryConditions();

    volScalarField alphaEff("alphaEff", turbulence->nu()/Pr + alphat);

    fvScalarMatrix TEqn
    (
        fvm::div(phi, T)
      - fvm::laplacian(alphaEff, T)
     ==
      radiation->ST(rhoCpRef, T)
      + fvOptions(T)
    );

    TEqn.relax();

    fvOptions.constrain(TEqn);

    TEqn.solve();
```

```
    radiation->correct();

    fvOptions.correct(T);

    // Updated to include buoyancy effects from both temperature and moisture
    rhok = 1.0 - beta*(T - TRef) - betaAH*(AH - AHRef);
}
```

readTransportProperties.H

```
singlePhaseTransportModel laminarTransport(U, phi);

// Thermal expansion coefficient [1/K]
dimensionedScalar beta
(
    "beta",
    dimless/dimTemperature,
    laminarTransport
);

// Water expansion coefficient
dimensionedScalar betaAH
(
    "betaAH",
    dimless,
    laminarTransport
);

// Reference water vapor mass fraction
dimensionedScalar AHRef("AHRef", dimless, laminarTransport);

// Laminar and Turbulent Schmidt number for water vapor transport
dimensionedScalar Sc("Sc", dimless, laminarTransport);
dimensionedScalar Sct("Sct", dimless, laminarTransport);

// Reference temperature [K]
dimensionedScalar TRef("TRef", dimTemperature, laminarTransport);

// Laminar Prandtl number
dimensionedScalar Pr("Pr", dimless, laminarTransport);

// Turbulent Prandtl number
dimensionedScalar Prt("Prt", dimless, laminarTransport);
```

createFields.H file

```
Info<< "Reading thermophysical properties\n" << endl;

Info<< "Reading field T\n" << endl;
volScalarField T
(
    IOobject
    (
        "T",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);

Info<< "Reading field p_rgh\n" << endl;
volScalarField p_rgh
(
    IOobject
    (
```

```
        "p_rgh",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);

Info<< "Reading field U\n" << endl;
volVectorField U
(
    IOobject
    (
        "U",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);

// Declare and initialize water vapor mass fraction
Info<< "Reading field AH (absolute humidity - dimensionless)\n" << endl;
volScalarField AH
(
    IOobject
    (
        "AH",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);

#include "createPhi.H"
#include "readTransportProperties.H"

Info<< "Creating turbulence model\n" << endl;
autoPtr<incompressible::turbulenceModel> turbulence
(
    incompressible::turbulenceModel::New(U, phi, laminarTransport)
);

// Kinematic density for buoyancy force
volScalarField rhok
(
    IOobject
    (
        "rhok",
        runTime.timeName(),
        mesh
    ),
    // Buoyancy contribution of AH
    1.0 - beta*(T - TRef) - betaAH*(AH - AHRef)
);

// kinematic turbulent thermal thermal conductivity m2/s
Info<< "Reading field alphat\n" << endl;
volScalarField alphat
(
    IOobject
    (
        "alphat",
        runTime.timeName(),
```

```
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);


#include "readGravitationalAcceleration.H"
#include "readhRef.H"
#include "gh.H"


volScalarField p
(
    IOobject
    (
        "p",
        runTime.timeName(),
        mesh,
        IOobject::NO_READ,
        IOobject::AUTO_WRITE
    ),
    p_rgh + rhok*gh
);

label pRefCell = 0;
scalar pRefValue = 0.0;
setRefCell
(
    p,
    p_rgh,
    simple.dict(),
    pRefCell,
    pRefValue
);

if (p_rgh.needReference())
{
    p += dimensionedScalar
    (
        "p",
        p.dimensions(),
        pRefValue - getRefCellValue(p, pRefCell)
    );
}

mesh.setFluxRequired(p_rgh.name());

#include "createMRF.H"
#include "createIncompressibleRadiationModel.H"
#include "createFvOptions.H"
```

cropTranFoam.C

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | www.openfoam.com
     \\/     M anipulation  |
-------------------------------------------------------------------------------
    Copyright (C) 2011-2017 OpenFOAM Foundation
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software: you can redistribute it and/or modify it
```

```
    under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.

Application
    cropTranFoam

Group
    grpHeatTransferSolvers

Description
    Steady-state solver for buoyant, turbulent flow of incompressible fluids.

    Uses the Boussinesq approximation:
    \f[
        rho_{k} = 1 - beta(T - T_{ref} - betaAH*(AH - AH_{ref}))
    \f]

    where:
        \f$ rho_{k} \f$ = the effective (driving) density
        beta = thermal expansion coefficient [1/K]
        T = temperature [K]
        \f$ T_{ref} \f$ = reference temperature [K]
        betaAH = water expansion coefficient [-]
        AH = absolute humidity [kg kg^-1]
        \f$ AH_{ref} \f$ = reference absolute humidity [kg kg^-1]

    Valid when:
    \f[
        \frac{beta(T - T_{ref})}{rho_{ref}} << 1
    \f]

\*---------------------------------------------------------------------------*/

#include "fvCFD.H"
#include "singlePhaseTransportModel.H"
#include "turbulentTransportModel.H"
#include "radiationModel.H"
#include "fvOptions.H"
#include "simpleControl.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

int main(int argc, char *argv[])
{
    argList::addNote
    (
        "Steady-state solver for buoyant, turbulent flow"
        " of incompressible fluids."
    );

    #include "postProcess.H"

    #include "addCheckCaseOptions.H"
    #include "setRootCaseLists.H"
    #include "createTime.H"
    #include "createMesh.H"
    #include "createControl.H"
    #include "createFields.H"
    #include "initContinuityErrs.H"
```

```
    turbulence->validate();

    // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

    Info<< "\nStarting time loop\n" << endl;

    while (simple.loop())
    {
        Info<< "Time = " << runTime.timeName() << nl << endl;

        // Pressure-velocity SIMPLE corrector
        {
            #include "UEqn.H"
            #include "TEqn.H"
            #include "AHEqn.H"
            #include "pEqn.H"
        }

        laminarTransport.correct();
        turbulence->correct();

        runTime.write();

        runTime.printExecutionTime(Info);
    }

    Info<< "End\n" << endl;

    return 0;
}


// ************************************************************************* //
```

`files` - file in `Make` directory

```
cropTranFoam.C

EXE = $(FOAM_USER_APPBIN)/cropTranFoam
```

`options` - file in `Make` directory

```
EXE_INC = \
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -I$(LIB_SRC)/meshTools/lnInclude \
    -I$(LIB_SRC)/sampling/lnInclude \
    -I$(LIB_SRC)/TurbulenceModels/turbulenceModels/lnInclude \
    -I$(LIB_SRC)/TurbulenceModels/incompressible/lnInclude \
    -I$(LIB_SRC)/transportModels \
    -I$(LIB_SRC)/transportModels/incompressible/singlePhaseTransportModel \
    -I$(LIB_SRC)/thermophysicalModels/radiation/lnInclude

EXE_LIBS = \
    -lfiniteVolume \
    -lfvOptions \
    -lmeshTools \
    -lsampling \
    -lturbulenceModels \
    -lincompressibleTurbulenceModels \
    -lincompressibleTransportModels \
    -lradiationModels \
    -latmosphericModels
```

A.2 Tutorial case cropTranFoamCase

A.2.1 0.orig directory files

AH - field file

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      AH;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 0 0 0 0 0 0];

internalField   uniform 0.0098;

boundaryField
{
    floor
    {
        type            zeroGradient;
    }

    ceiling
    {
        type            zeroGradient;
    }

    leftWall
    {
        type            fixedValue;
        value           uniform 0.0098;
    }

    "(rightWall|backWall|frontWall)"
    {
        type            zeroGradient;
    }
}

// ************************************************************************* //
```

alphat - field file

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      alphat;
```

```
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 2 -1 0 0 0 0];

internalField   uniform 0;

boundaryField
{
    floor
    {
        type            alphatJayatillekeWallFunction;
        Prt             0.85;
        value           uniform 0;
    }

    ceiling
    {
        type            alphatJayatillekeWallFunction;
        Prt             0.85;
        value           uniform 0;
    }
    "(leftWall|rightWall)"
    {
        type zeroGradient;
    }
    "(backWall|frontWall)"
    {
        type            alphatJayatillekeWallFunction;
        Prt             0.85;
        value           uniform 0;
    }
}


// ************************************************************************* //
```

`epsilon` - field file

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      epsilon;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 2 -3 0 0 0 0];

internalField   uniform 0.00108;

boundaryField
{
    floor
    {
        type            epsilonWallFunction;
        value           uniform 0;
    }

    ceiling
```

```
    {
        type            epsilonWallFunction;
        value           uniform 0;
    }
    leftWall
    {
        type fixedValue;
        value uniform 0.00108;  // Example value; adjust based on your specific calculation
    }

    rightWall
    {
        type zeroGradient;
    }

    "(backWall|frontWall)"
    {
        type            epsilonWallFunction;
        value           uniform 0;
    }
}


// ************************************************************************* //
```

k - field file

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      k;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 2 -2 0 0 0 0];

internalField   uniform 0.00375;

boundaryField
{
    floor
    {
        type            kqRWallFunction;
        value           uniform 0;
    }

    ceiling
    {
        type            kqRWallFunction;
        value           uniform 0;
    }

    leftWall
    {
        type fixedValue;
        value uniform 0.00375;  // Example value; adjust based on your turbulence assumptions
    }
    rightWall
    {
```

```
        type            zeroGradient;
    }
    "(backWall|frontWall)"
    {
        type            kqRWallFunction;
        value           uniform 0;
    }
}


// ************************************************************************* //
```

nut - field file

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      nut;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 2 -1 0 0 0 0];

internalField   uniform 0.00117;

boundaryField
{
    floor
    {
        type            nutkWallFunction;
        value           uniform 0;
    }

    ceiling
    {
        type            nutkWallFunction;
        value           uniform 0;
    }
    leftWall
    {
        type calculated;
        value uniform 1e-5;  // Adjust based on the actual turbulence model parameters
    }
    rightWall
    {
        type zeroGradient;
    }

    "(backWall|frontWall)"
    {
        type            nutkWallFunction;
        value           uniform 0;
    }
}


// ************************************************************************* //
```

p - field file

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      p;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 2 -2 0 0 0 0];

internalField   uniform 0;

boundaryField
{
    floor
    {
        type            calculated;
        value           $internalField;
    }

    ceiling
    {
        type            calculated;
        value           $internalField;
    }
    leftWall
    {
        type            zeroGradient;
    }
    rightWall
    {
        type fixedValue;
        value uniform 0;
    }

    "(backWall|frontWall)"
    {
        type            calculated;
        value           $internalField;
    }
}


// ************************************************************************* //
```

p_rgh - field file

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
```

```
    class       volScalarField;
    object      p_rgh;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 2 -2 0 0 0 0];

internalField   uniform 0;

boundaryField
{
    floor
    {
        type            fixedFluxPressure;
        rho             rhok;
        value           uniform 0;
    }

    ceiling
    {
        type            fixedFluxPressure;
        rho             rhok;
        value           uniform 0;
    }

    leftWall
    {
        type            zeroGradient;
    }

    rightWall
    {
        type            fixedValue;
        value           uniform 0;
    }

    "(backWall|frontWall)"
    {
        type            fixedFluxPressure;
        rho             rhok;
        value           uniform 0;
    }
}


// ************************************************************************* //
```

T - field file - `T`

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      T;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 0 0 1 0 0 0];

internalField   uniform 297;
```

```
boundaryField
{
    floor
    {
        type            zeroGradient;
    }

    ceiling
    {
        type            zeroGradient;
    }

    leftWall
    {
        type fixedValue;
        value uniform 292;  // Set the desired inlet temperature
    }

    "(rightWall|backWall|frontWall)"
    {
        type            zeroGradient;
    }

}


// ************************************************************************* //
```

U - field file

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -1 0 0 0 0];

internalField   uniform (0 0 0);

boundaryField
{
    floor
    {
        type            noSlip;
    }

    ceiling
    {
        type            noSlip;
    }

    leftWall
    {
        type            fixedValue;
        value           uniform (1 0 0);
    }
```

```
    rightWall
    {
        type zeroGradient;
    }

    "(backWall|frontWall)"
    {
        type            noSlip;
    }
}


// ************************************************************************* //
```

A.2.2 system directory files

blockMeshDict -file

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

scale   0.1;

vertices
(
    (0 0 0)
    (10 0 0)
    (10 5 0)
    (0 5 0)
    (0 0 10)
    (10 0 10)
    (10 5 10)
    (0 5 10)
);

blocks
(
    hex (0 1 2 3 4 5 6 7) (40 20 40) simpleGrading (1 1 1)
);

edges
(
);

boundary
(
    floor
    {
        type wall;
        faces
        (
            (1 5 4 0)
```

```
        );
    }
    ceiling
    {
        type wall;
        faces
        (
            (3 7 6 2)
        );
    }
    leftWall
    {
        type patch;
        faces
        (
            (0 4 7 3)
        );
    }
    rightWall
    {
        type patch;
        faces
        (
            (1 5 6 2)
        );
    }
    frontWall
    {
        type wall;
        faces
        (
            (0 1 2 3)
        );
    }
    backWall
    {
        type wall;
        faces
        (
            (4 5 6 7)
        );
    }
);

mergePatchPairs
(
);


// ************************************************************************* //
```

controlDict - file

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
```

```
application     cropTranFoam;

startFrom       latestTime;

startTime       0;

stopAt          endTime;

endTime         100;

deltaT          1;

writeControl    timeStep;

writeInterval   25;

purgeWrite      0;

writeFormat     ascii;

writePrecision  10;

writeCompression off;

timeFormat      general;

timePrecision   6;

runTimeModifiable true;


// ************************************************************************* //
```

`fvOptions` - file for definition of source terms and crop transpiration model

```
/*---------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      fvOptions;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

porosity1
{
    // Type of the fvOption
    type            explicitPorositySource;
    active          true;

    explicitPorositySourceCoeffs
    {
        // Porosity model type
        type            DarcyForchheimer;
        // Region to apply - porosity zone
        selectionMode   cellZone;
        cellZone        porousZone;

        DarcyForchheimerCoeffs
        {
```

```
            // Darcy coefficients (viscous resistance)
            d  (50 50 50);

            // Forchheimer coefficients (inertial resitance)
            f  (2 2 2);

            // Coordinate system for anisotropic porosity
            coordinateSystem
            {
                origin  (0 0 0);
                e1      (1 0 0);
                e2      (0 1 0);
            }
        }
    }
}

// Calculation for absolute humdity source term to add water vapor from transpiration
AHSource
{
    type            scalarCodedSource;
    active          true;
    name            AHSource;

    scalarCodedSourceCoeffs
    {
        selectionMode  cellZone;
        cellZone       porousZone;
        fields         (AH);

        codeInclude
        #{
            // Constant values for case study
            const double lai = 3.0;                         // Leaf area index
            const double l_leaf = 0.1;                      // Mean leaf diameter (m)
            const double reflectionCoefficient = 0.05;      // Reflection coefficient of lettuce for
    PAR (0.05-0.08)
            const double ppfd = 200.0;                      // Light intensity ( mol /m^2/s)
            const double cultivationAreaCoverage = 0.90;    // Ratio of projected leaf area to
    cultivation area (0.9-1.0)

            // Constant values for fluids
            const double ATMOSPHERIC_PRESSURE = 101325;     // Pa (standard atmospheric pressure)
            const double HEAT_CAPACITY_OF_AIR = 1005.0;     // J kg^-1 K^-1
            const double HEAT_CAPACITY_OF_WATER = 1859.0;   // J kg^-1 K^-1
            const double DENSITY_OF_AIR = 1.2041;           // kg/m^3
            const double DENSITY_OF_WATER = 998;            // kg/m^3
            const double LATENT_HEAT_WATER = 2264705;       // J kg^-1
            const double PSYCHOMETRIC_CONSTANT = 65.0;      // Pa/K
            const double IDEAL_GAS_CONSTANT = 8.3145;       // J mol^-1 K^-1
            const double MOLAR_MASS_H2O_GRAMS = 18.01528;   // g mol^-1
            const double ZERO_DEGREES_IN_KELVIN = 273.15;

/*---------------------------------------------------------------------------*\
                            Function Declaration
\*---------------------------------------------------------------------------*/

            //- Converts a temperature from Kelvin to Celsius.
            // @param tempKev Temperature in Kelvin.
            // @return Temperature in degree Celsius.
            double convertKelvinToCelsius
            (
                double tempKev
            );

            //- Calculates the relative humidity
            // This is based on the saturation vapor pressure p_vs (Pa)
            // as the Tetens equation equation is valid for
```

```
            // temperature between 0 C  and 50 C )
            // @param tempAir Temperature of the air in degree Celsius
            // @param absolute_humidity Water vapor mass fraction (kg/kg)
            // @return Relative humidity in percentage
            double calcRelativeHumidity
            (
                double tempAir,
                double absolute_humidity
            );

            //- Calculates the aerodynamic resistance
            // @param l_leaf mean leaf diameter (m)
            // @param lai leaf area index
            // @param u local air speed (m/s)
            // @return s/m
            double calcVapourResistance
            (
                double l_leaf,
                double u,
                double lai
            );

            //- Calculates the net radiation on the surface.
            // @param ppfd Photosynthetic photon flux density (light intensity in  mol /m  /s).
            // @param reflectionCoefficient Reflection coefficient of the surface.
            // @param cultivationAreaCoverage Fraction of surface covered by vegetation or material.
            // @return Net radiation on the surface in W/m  .
            double calcNetRadiation
            (
                double ppfd,
                double reflectionCoefficient,
                double cultivationAreaCoverage
            );

            //- Calculates the radiation from lighting based on PPFD.
            // @param ppfd Photosynthetic photon flux density (light intensity in  mol /m  /s).
            // @return Radiation from lighting in W/m  .
            // @throws std::runtime_error if PPFD value is unsupported.
            double calcLightingRadiation
            (
                double ppfd
            );

            //- Calculates the sensible heat exchange between the surface and air.
            // @param tempAir Air temperature in degrees Celsius.
            // @param tempSurface Surface temperature in degrees Celsius.
            // @param lai Leaf area index (area of leaves per unit ground area).
            // @param vapourResistance Vapor resistance in s/m.
            // @return Sensible heat exchange in W/m  .
            double calcSensibleHeatExchange
            (
                double tempAir,
                double tempSurface,
                double lai,
                double vapourResistance
            );

            //- Calculates the latent heat flux from evaporation or transpiration.
            // @param tempAir Air temperature in degrees Celsius.
            // @param tempSurface Surface temperature in degrees Celsius.
            // @param relativeHumidity Relative humidity as a percentage.
            // @param ppfd Photosynthetic photon flux density (light intensity in  mol /m  /s).
            // @param lai Leaf area index (area of leaves per unit ground area).
            // @param vapourResistance Vapor resistance in s/m.
            // @return Latent heat flux in W/m  .
            double calcLatentHeatFlux
            (
                double tempAir,
```

```
            double tempSurface,
            double relativeHumidity,
            double ppfd,
            double lai,
            double vapourResistance
        );

        //- Calculates the vapor concentration in the air.
        // @param tempAir Air temperature in degrees Celsius.
        // @param relativeHumidity Relative humidity as a percentage.
        // @return Vapor concentration of air in kg/m  .
        double calcVapourConcentrationAir
        (
            double tempAir,
            double relativeHumidity
        );

        //- Calculates the saturated vapor concentration of air at a given temperature.
        // @param tempAir Air temperature in degrees Celsius.
        // @return Saturated vapor concentration of air in kg/m  .
        double calcSaturatedVapourConcentrationAir
        (
            double tempAir
        );

        //- Calculates the saturated vapor pressure of air at a given temperature.
        // @param tempAir Air temperature in degrees Celsius.
        // @return Saturated vapor pressure in Pascals.
        double calcSaturatedVapourPressureAir
        (
            double tempAir
        );

        //- Calculates the vapor concentration at the surface.
        // @param tempAir Air temperature in degrees Celsius.
        // @param tempSurface Surface temperature in degrees Celsius.
        // @param vapourConcentrationAir Vapor concentration of air in kg/m  .
        // @return Vapor concentration at the surface in kg/m  .
        double calcVapourConcentrationSurface
        (
            double tempAir,
            double tempSurface,
            double vapourConcentrationAir
        );

        //- Calculates the psychrometric constant epsilon at a given air temperature.
        // @param tempAir Air temperature in degrees Celsius.
        // @return Psychrometric constant epsilon (dimensionless)
        double calcEpsilon
        (
            double tempAir
        );

        //- Calculates stomatal resistance based on light intensity.
        // @param ppfd Photosynthetic photon flux density (light intensity in  mol /m  /s).
        // @return Stomatal resistance in s/m.
        double calcStomatalResistance
        (
            double ppfd
        );

        //- Solves for surface temperature based on energy balance.
        // Root finding algorithm, a bisection method, is used to
        // iteratively find the solution.
        // @param tempAir Air temperature in degrees Celsius.
        // @param ppfd Photosynthetic photon flux density (light intensity in  mol /m  /s).
        // @param relativeHumidity Relative humidity as a percentage.
        // @param lai Leaf area index (area of leaves per unit ground area).
```

```
            // @param vapourResistance Vapor resistance in s/m.
            // @param reflectionCoefficient Reflection coefficient of the surface.
            // @param cultivationAreaCoverage Fraction of surface covered by vegetation or material.
            // @return Surface temperature in degrees Celsius
            double calcTempSurface
            (
                double tempAir,
                double ppfd,
                double relativeHumidity,
                double lai,
                double vapourResistance,
                double reflectionCoefficient,
                double cultivationAreaCoverage
            );

/*---------------------------------------------------------------------------*\
                            Function Definition
\*---------------------------------------------------------------------------*/

            double convertKelvinToCelsius
            (
                double tempKev
            )
            {
                return (tempKev - ZERO_DEGREES_IN_KELVIN);
            };

            double calcRelativeHumidity
            (
                double tempAir,
                double absolute_humidity
            )
            {
                double teten = (7.5 * tempAir / (tempAir + 237.3));
                double p_vs = 610.94 * pow(10, teten);
                double p_v = absolute_humidity * ATMOSPHERIC_PRESSURE;

                return (p_v / p_vs) * 100;
            };

            double calcVapourResistance
            (
                double l_leaf,
                double u,
                double lai
            )
            {
                return 350* pow(l_leaf/u,0.5) * pow(lai,-1);
            };

            double calcLightingRadiation
            (
                double ppfd
            )
            {
                if (ppfd == 140) return 28.0;
                if (ppfd == 200) return 41.0;
                if (ppfd == 300) return 59.0;
                if (ppfd == 400) return 79.6;
                if (ppfd == 450) return 90.8;
                if (ppfd == 600) return 120.0;
                throw std::runtime_error("PPFD value not supported.");
            }

            double calcNetRadiation
            (
                double ppfd,
                double reflectionCoefficient,
```

```
            double cultivationAreaCoverage
        )
        {
            double lightingRadiation = calcLightingRadiation(ppfd);
            return (1.0 - reflectionCoefficient) * lightingRadiation * cultivationAreaCoverage;
        }

        double calcSensibleHeatExchange
        (
            double tempAir,
            double tempSurface,
            double lai,
            double vapourResistance
        )
        {
            return lai * HEAT_CAPACITY_OF_AIR * DENSITY_OF_AIR * \
            (tempSurface - tempAir) / vapourResistance;
        }

        double calcLatentHeatFlux
        (
            double tempAir,
            double tempSurface,
            double relativeHumidity,
            double ppfd,
            double lai,
            double vapourResistance
        )
        {
            double vapourConcentrationAir = calcVapourConcentrationAir(tempAir, relativeHumidity);
            double vapourConcentrationSurface = calcVapourConcentrationSurface(tempAir,
    tempSurface, vapourConcentrationAir);
            double stomatalResistance = calcStomatalResistance(ppfd);
            return lai * (LATENT_HEAT_WATER / 1000.0) * ((vapourConcentrationSurface -
    vapourConcentrationAir) / (stomatalResistance + vapourResistance));
        }

        double calcVapourConcentrationAir
        (
            double tempAir,
            double relativeHumidity
        )
        {
            return calcSaturatedVapourConcentrationAir(tempAir) * (relativeHumidity / 100.0);
        }

        double calcSaturatedVapourConcentrationAir
        (
            double tempAir
        )
        {
            double saturatedVapourPressure = calcSaturatedVapourPressureAir(tempAir);
            return (saturatedVapourPressure / (IDEAL_GAS_CONSTANT * (tempAir +
    ZERO_DEGREES_IN_KELVIN))) * MOLAR_MASS_H2O_GRAMS;
        }

        double calcSaturatedVapourPressureAir
        (
            double tempAir
        )
        {
            double tempAirK = tempAir + ZERO_DEGREES_IN_KELVIN;
            return std::exp(77.345 + (0.0057 * tempAirK) - (7235.0 / tempAirK)) / std::pow(
    tempAirK, 8.2);
        }

        double calcVapourConcentrationSurface
        (
```

```
            double tempAir,
            double tempSurface,
            double vapourConcentrationAir
        )
        {
            double epsilon = calcEpsilon(tempAir);
            return vapourConcentrationAir + ((HEAT_CAPACITY_OF_AIR * DENSITY_OF_AIR) /
LATENT_HEAT_WATER) * epsilon * (tempSurface - tempAir) * 1000.0;
        }

        double calcEpsilon
        (
            double tempAir
        )
        {
            double delta = 0.04145 * std::exp(0.06088 * tempAir);
            return (delta / PSYCHOMETRIC_CONSTANT) * 1000.0;
        }

        double calcStomatalResistance
        (
            double ppfd
        )
        {
            return 60.0 * (1500.0 + ppfd) / (200.0 + ppfd);
        }

        double calcTempSurface
        (
            double tempAir,
            double ppfd,
            double relativeHumidity,
            double lai,
            double vapourResistance,
            double reflectionCoefficient,
            double cultivationAreaCoverage
        )
        {
            double netRadiation = calcNetRadiation
                    (ppfd, reflectionCoefficient, cultivationAreaCoverage);
            auto calcEnergyBalance = [&](double tempSurface)
            {
                double sensibleHeat = calcSensibleHeatExchange
                    (tempAir, tempSurface, lai, vapourResistance);
                double latentHeat = calcLatentHeatFlux
                    (tempAir, tempSurface, relativeHumidity, ppfd, lai, vapourResistance);
                return netRadiation - sensibleHeat - latentHeat;
            };

            // Root finding using bisection method
            double limit = 10.0;
            double xa = tempAir - limit;
            double xb = tempAir + limit;

            double tol = 1e-6;
            while (std::fabs(xb - xa) > tol)
            {
                double xm = (xa + xb) / 2.0;
                if (calcEnergyBalance(xm) * calcEnergyBalance(xa) < 0) {
                    xb = xm;
                } else {
                    xa = xm;
                }
            }

            return (xa + xb) / 2.0;
        }
    #};
```

```
    codeCorrect
    #{
        Pout<< "**codedCorrect**" << endl;
    #};

    codeAddSup
    #{
        // Provide field values
        const volScalarField& AH = mesh().lookupObject<volScalarField>("AH");
        const volVectorField& U = mesh().lookupObject<volVectorField>("U");
        // Allow modification of T
        volScalarField& T = const_cast<volScalarField&>(mesh().lookupObject<volScalarField>("T"));
        // Reference to the source term in the equation
        scalarField& AHSource = eqn.source();
        // Access the porous zone ID
        const label porousZoneID = mesh().cellZones().findZoneID("porousZone");

        // Loop through the cells in mesh
        forAll(AH, cellID)
        {
            // Execute code if cell is in porousZone
            if (mesh().cellZones().whichZone(cellID) == porousZoneID)
            {
                // Get field values for cellID
                double tempInKelvin = T[cellID];
                double absolute_humidity = AH[cellID];
                vector Ucell = U[cellID];
                scalar u = mag(Ucell);
                // Convert K into  C
                double tempAir = convertKelvinToCelsius(tempInKelvin);
                // Convert AH into RH
                double relativeHumidity = calcRelativeHumidity(tempAir, absolute_humidity);
                // Calculate vapour/aerodynamic resistance
                double vapourResistance = calcVapourResistance(l_leaf, u, lai);

                // Calculate surface temperature - root-finding method
                double tempSurface = calcTempSurface(tempAir, ppfd, relativeHumidity, lai,
                                    vapourResistance, reflectionCoefficient,
    cultivationAreaCoverage);

                // Calculate sensible heat exchange and latent heat flux
                double sensibleHeatExchange = calcSensibleHeatExchange(tempAir, tempSurface, lai,
                                    vapourResistance);
                double latentHeatFlux = calcLatentHeatFlux(tempAir, tempSurface, relativeHumidity,
                                    ppfd, lai, vapourResistance);

                // Calculate the rate of mass flux per volume dervied from transpiration rate (ET)
                scalar ET = latentHeatFlux / LATENT_HEAT_WATER / l_leaf;
                // Normalized the mass flux to a flux per time-unit
                ET = ET / DENSITY_OF_WATER;
                // Add as a source term to the right-hand side of the transport equation
                AHSource[cellID] -= ET;

                // Calculate the specific heat capacity of moist air
                scalar specific_heat_moist_air = HEAT_CAPACITY_OF_AIR * \
                                (1 - AH[cellID]) + (AH[cellID] * HEAT_CAPACITY_OF_WATER);
                // Calculate the mass of moist air
                scalar m_air = (ATMOSPHERIC_PRESSURE / (287 * tempInKelvin)) / (1 + 0.61 * AH[
    cellID]);
                scalar delta_T = sensibleHeatExchange / (m_air * specific_heat_moist_air);
                // Calculate temperature increase delta_T
                T[cellID] += delta_T;
            }
        }
    #};

    codeConstrain
```

```
        #{
            Pout<< "**codeConstrain**" << endl;
        #};

    }
}

// ************************************************************************* //
```

fvSchemes - file

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

ddtSchemes
{
    default         steadyState;
}

gradSchemes
{
    default         Gauss linear;
}

divSchemes
{
    default                         none;

    turbulence                      bounded Gauss upwind;
    div(phi,AH)                     $turbulence;
    div(phi,U)                      $turbulence;
    div(phi,T)                      $turbulence;
    div(phi,k)                      $turbulence;
    div(phi,epsilon)                $turbulence;
    div((nuEff*dev2(T(grad(U)))))   Gauss linear;
}

laplacianSchemes
{
    default         Gauss linear corrected;
}

interpolationSchemes
{
    default         linear;
}

snGradSchemes
{
    default         corrected;
}


// ************************************************************************* //
```

`fvSolution` - file

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    AH
    {
        solver          PBiCG;  // Solver type
        preconditioner  DILU;   // Preconditioner
        tolerance       1e-8;   // Tolerance
        relTol          0.01;   // Relative tolerance
    }
    p_rgh
    {
        solver          PCG;
        preconditioner  DIC;
        tolerance       1e-08;
        relTol          0.01;
    }

    "(U|T|k|epsilon)"
    {
        solver          PBiCGStab;
        preconditioner  DILU;
        tolerance       1e-05;
        relTol          0.1;
    }
}

SIMPLE
{
    nNonOrthogonalCorrectors 0;
    pRefCell        0;
    pRefValue       0;

    residualControl
    {
        p_rgh           1e-2;
        U               1e-4;
        T               1e-2;

        // possibly check turbulence fields
        "(k|epsilon|omega)" 1e-3;
    }
}

relaxationFactors
{
    fields
    {
        p_rgh           0.7;
    }
    equations
```

```
    {
        U               0.5;
        T               0.5;
        "(k|epsilon)" 0.7;
    }
}


// ************************************************************************* //
```

topoSetDict to add porous media zone

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      topoSetDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

actions
(
    {
        name    porousCells;
        type    cellSet;
        action  new;
        source  boxToCell;
        box     (0.25 0 0.25) (0.75 0.1 0.75);
    }
    {
        name    porousZone;
        type    cellZoneSet;
        action  new;
        source  setToCellZone;
        set     porousCells;
    }
);
```

A.2.3 constant directory files

g - file for gravitational force

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       uniformDimensionedVectorField;
    object      g;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
```

```
dimensions      [0 1 -2 0 0 0 0];
value           (0 -9.81 0);


// ************************************************************************* //
```

transportProperties - file

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      transportProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

transportModel Newtonian;

// Laminar viscosity
nu              1.54e-05;

// Thermal expansion coefficient
beta            3e-03;

// Reference temperature
TRef            292;

// Laminar Prandtl number
Pr              0.7;

// Turbulent Prandtl number
Prt             0.85;

// Water expansion coefficient
betaAH          2.6e-5; // m^2/s for water vapor in air at 25 C

// Reference water vapor mass
AHRef           0.00968;

// Laminar Schmidt number
Sc              0.7;

// Turbulent Schmidt number
Sct             0.8;


// ************************************************************************* //
```

turbulenceProperties - file

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
```

```
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      turbulenceProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

simulationType      RAS;

RAS
{
    RASModel        kEpsilon;

    turbulence      on;

    printCoeffs     on;
}


// ************************************************************************* //
```

Index