Heat, Air and Moisture transport in Building Constructions
Version 2015
# INDEX

INDEX...........................................................................................................................................3

1. General Information.....................................................................................................................5
2. Matlab.........................................................................................................................................11
3. COMSOL Multiphysics..................................................................................................................19
4. Damage Indicators.......................................................................................................................24
5. Types of Errors...........................................................................................................................29
6. References ....................................................................................................................................29
7. Appendix ......................................................................................................................................30
   (A) Plots for Crefun.m..................................................................................................................30
   (B) Using self-made tables..........................................................................................................31
   (C) Wind-Driven Rain..................................................................................................................31
   (D) Sophisticated Convection Method .......................................................................................32
   (E) %RMSE and maximal error ..................................................................................................34
   (F) Mollier Diagram...................................................................................................................35
   (G) intersections.m ....................................................................................................................36
Heat, Air and Moisture transport in Building Constructions

HAM-BC is a manual which explains the implementation of a simulation model for heat, air and moisture transport in building constructions in COMSOL Multiphysics and Matlab. HAM-BC has the ability to implement heat and moisture transport by diffusion and convection in 1D, 2D and 3D, including rain, solar irradiation and latent heat. HAM-BC has the ability to implement the dependence of material properties on temperature and humidity. HAM-BC uses the temperature (T) in [°C] as potential for heat transport calculation; and the logarithmic capillary pressure (Lp_c) in [Pa] as potential for the moisture transport calculation. It is assumed that the user of this manual has basic knowledge about Matlab and COMSOL Multiphysics 5.0 (COMSOL for short). HAM-BC is based on [Goesten 2016], which is a further elaboration of [Schijndel 2006] and [Uittenbosch 2012].

Coefficients are made in Matlab, which creates txt-files that are imported in COMSOL. The coefficients and material properties which are implemented in HAM-BC are shown in figure 1.

![Figure 1: Material properties (left) which are implemented in the coefficients (right).](image)

1. General Information

The important formulas used in HAM-BC are discussed in this chapter. More information and formulas used in HAM-BC 2015 can be found in [Goesten 2016], including references.

The potential of moisture transport in HAM-BC is the logarithmic capillary pressure:

\[
Lp_c = 10^{log(p_c)}
\]

The relative humidity is determined by:

\[
RH = \frac{p_v}{p_{sat}} \cdot 100\% = \varphi \cdot 100\%
\]
\[ p_{\text{sat}} = \exp \left( 65.8094 - \frac{7066.27}{T} - 5.976 \cdot \ln(T) \right) \]  

RH = relative humidity [%]
φ = relative humidity [-]
p\_v = vapor pressure [Pa]
p\_sat = saturation pressure [Pa]
T = temperature [K]

The relationship between the capillary pressure \( P_c \) and the relative humidity \( \phi \) is:
\[ p_c = \rho_w \cdot R_v \cdot T \cdot \ln(\phi) \]  
\[ \phi = \exp \left( \frac{-p_c}{\rho_w \cdot R_v \cdot T} \right) \]

\( p_c \) = capillary pressure [Pa]
\( \rho_w \) = specific density of water [kg/m\(^3\)] = 1000 kg/m\(^3\)
\( R_v \) = gas constant of vapor [J/(kg·K)] = 462 J/(kg·K)
\( T \) = temperature [K]
\( \phi \) = relative humidity [-]

\[ \vec{g}_v = -\delta_v \cdot \text{grad} p_v = -\frac{\delta_a}{\mu} \cdot \text{grad} p_v \]

\( g_v \) = water vapor flow rate [kg/(m\(^2\)·s)]
\( \delta_v \) = vapor permeability of the material [(kg·m)/(s·Pa)]
\( \delta_a \) = vapor permeability of air [(kg·m)/(s·Pa)]
\( \mu \) = vapor diffusion resistance factor [-]
\( p_v \) = vapor pressure [Pa]

The moisture transportation by capillary suction is described by:
\[ \vec{g}_l = -k_m \cdot \text{grad} (p_c + \rho_w \cdot g \cdot z) \]

\( g_l \) = moisture flow rate [kg/(m\(^2\)·s)]
\( k_m \) = water permeability [kg/(s·m·Pa)] ~ [s]
\( p_c \) = capillary pressure [Pa]
\( \rho_w \) = specific density of water [kg/m\(^3\)] = 1000 kg/m\(^3\)
\( g \) = gravity [m/s\(^2\)] = 9.81 m/s\(^2\) (excluded)
\( z \) = vertical height [m] (excluded)

The air flux through a material can be described by:
\[ \vec{g}_a = -k_a \cdot (\text{grad} p_{\text{air}} + \rho_a \cdot g) \]

\( g_a \) = air flux through the material [kg/(m\(^2\)·s)]
\( k_a \) = air permeability [kg/(s·m·Pa)] ~ [s]
\( p_{\text{air}} \) = air pressure [Pa]
\( \rho_a \) = specific density of air [kg/m\(^3\)]
\( g \) = gravity [m/s\(^2\)] = 9.81 m/s\(^2\) (excluded)

**Boundary conditions for heat transport**
All boundary conditions for heat transport in HAM-BC are in [W/m\(^2\)]. Heat flux by diffusion, latent heat, infiltration of air, rain and solar irradiation are discussed in this paragraph. The formula of boundary conditions for heat by diffusion (conductivity) is:
\[ q = h_c \cdot (T_a - T_s) \]  

\( q \) = heat flux \([\text{W/m}^2]\)  
\( h_c \) = surface coefficient of heat transfer \([\text{W/m}^2\text{K}]\)  
\( T_a \) = ambient temperature \( [\degree \text{C}] \)  
\( T_s \) = surface temperature \( [\degree \text{C}] \)

The boundary conditions for heat are influenced by latent heat. The latent heat is dependent on the vapor flux:

\[ q = L \cdot \beta_p \cdot (p_{v_a} - p_{v_s}) \]  

\( q \) = heat flux \([\text{W/m}^2]\)  
\( L \) = latent heat \([\text{J/kg}] = 2.5 \cdot 10^6 \text{J/kg evaporation}\)  
\( \beta_p \) = surface coefficient of vapor transfer \([\text{s/m}]\)  
\( p_{v_a} \) = ambient water vapor pressure \([\text{Pa}]\)  
\( p_{v_s} \) = surface water vapor pressure \([\text{Pa}]\)

The heat boundary condition for infiltration of air is:

\[ q = c_p \cdot \vec{g_a} \cdot (T_a - T_s) + L \cdot 0.62 \cdot 10^{-5} \cdot \vec{g_a} \cdot (p_{v_a} - p_{v_s}) \]  

\( q \) = heat flux \([\text{W/m}^2]\)  
\( c_p \) = specific heat capacity of air \([\text{J/(kg \cdot K]}\)  
\( \vec{g_a} \) = air flux in mass \([\text{kg/(m}^2\text{\cdot s)}]\)  
\( L \) = latent heat \([\text{J/kg}] = 2.5 \cdot 10^6 \text{J/kg evaporation}\)  
\( T_a \) = ambient temperature \( [\degree \text{C}] \)  
\( T_s \) = surface temperature \( [\degree \text{C}] \)  
\( p_{v_a} \) = ambient water vapor pressure \([\text{Pa}]\)  
\( p_{v_s} \) = surface water vapor pressure \([\text{Pa}]\)

The terms \((T_a - T_s)\) and \((p_{v_a} - p_{v_s})\) in formula (11) must be changed to respectively \((T_s - T_a)\) and \((p_{v_s} - p_{v_a})\) at the surface where the air leaves the material. An alternative is to use the same formulas, but multiplying it with \(-1\).

The influence of rain on the heat flux is described by:

\[ q_{\text{rain}} = \vec{g_{\text{rain}}} \cdot c_l \cdot (T_{\text{rain}} - T_s) \]  

\( q_{\text{rain}} \) = heat flux caused by rain \([\text{W/m}^2]\)  
\( \vec{g_{\text{rain}}} \) = rain flux \([\text{kg/(m}^2\text{\cdot s)}]\)  
\( c_l \) = specific heat of liquid water \([\text{J/(kg \cdot K]}\) = 4200 \text{J/(kg \cdot K)}\)  
\( T_{\text{rain}} \) = temperature of the rain \( [\degree \text{C}] \approx \text{ambient temperature} [\degree \text{C}]\)  
\( T_s \) = surface temperature \( [\degree \text{C}]\)

Solar irradiation is implemented with the formula:

\[ q_{\text{sol}} = \alpha \cdot E_{\text{sol}} \]  

\( \alpha \) = absorption factor for solar irradiation \([-]\)  
\( E_{\text{sol}} \) = solar irradiation on a horizontal plane \([\text{W/m}^2]\)
The total formula for the boundary conditions for heat is:

\[ q = h_c \cdot (T_a - T_s) + L \cdot \beta_p \cdot (p_{va} - p_{vs}) + c_p \cdot g_a \cdot (T_a - T_s) + L \cdot 0.62 \cdot 10^{-5} \cdot g_a \cdot (p_{va} - p_{vs}) + q_{rain} + q_{sol} \]

**Boundary conditions for moisture transport**

The moisture boundary conditions caused by diffusion, air infiltration and rain are discussed in this section. All boundary conditions related to moisture have the dimension \([\text{kg}/(\text{m}^2\cdot\text{s})]\).

**Boundary conditions for moisture transport caused by infiltration of air** is described by:

\[ g = \beta_p \cdot (p_{va} - p_{vs}) \]  

\( g = \text{mass flux of moisture } [\text{kg}/(\text{m}^2\cdot\text{s})] \)

\( \beta_p = \text{surface coefficient of vapor transfer } [\text{s/m}] \)

\( p_{va} = \text{ambient water vapor pressure } [\text{Pa}] \)

\( p_{vs} = \text{surface water vapor pressure } [\text{Pa}] \)

The term \((p_{va} - p_{vs})\) in formula (16) must be changed to \((p_{vs} - p_{va})\) at the **surface where the air leaves the material**. An alternative is to multiply the formula with minus one.

When **rain** is taken into account, then the value \(g_{rain} [\text{kg}/(\text{m}^2\cdot\text{s})]\) must be added.

**The total boundary condition for moisture is:**

\[ g = \beta_p \cdot (p_{va} - p_{vs}) + 0.62 \cdot 10^{-5} \cdot g_a \cdot (p_{va} - p_{vs}) + g_{rain} \]  

**Heat and moisture balance equations**

This section describes how the heat balance equation and the moisture balance equation are implemented in COMSOL with the help of text-files generated by Matlab.

The formula for heat transfer and temperature change in the material is:

\[ (\rho \cdot c + c_t \cdot w) \frac{\partial T}{\partial t} = \left( \text{div} \lambda + L \left( \text{div} \frac{\delta_a}{\rho} - 0.62 \cdot 10^{-5} \cdot g_a \right) \cdot \varphi \cdot \frac{\partial p_{sat}}{\partial T} - c_p \cdot g_a \right) \cdot \text{grad} T + L \left( \text{div} \frac{\delta_a}{\rho} + 0.62 \cdot 10^{-5} \cdot g_a \right) \cdot \varphi \cdot \frac{\partial p}{\partial \mu} \cdot \text{grad} L_p \]

**The formula for moisture transfer and moisture storage is:**

\[ \frac{\partial w}{\partial p} \cdot \frac{\partial p}{\partial t} = \left( \text{div} \frac{\delta_a}{\rho} - 0.62 \cdot 10^{-5} \cdot g_a \right) \cdot \varphi \cdot \frac{\partial p_{sat}}{\partial T} \cdot \text{grad} T + \left( \left( \text{div} \frac{\delta_a}{\rho} + 0.62 \cdot 10^{-5} \cdot g_a \right) \cdot \varphi \cdot \frac{\partial p}{\partial \mu} \cdot \text{grad} L_p \right) \]
\[ p = \text{specific density [kg/m}^3]\]
\[ c = \text{specific heat capacity of the material [J/(kg·K)]}\]
\[ c_i = \text{specific heat capacity of water [J/(kg·K)] = 4200 J/(kg·K)}\]
\[ c_p = \text{specific heat capacity of air [J/(kg·K)]}\]
\[ w = \text{moisture content [kg}/m^3]\]
\[ T = \text{temperature [K]}\]
\[ t = \text{time [s]}\]
\[ \lambda = \text{thermal conductivity [W/(m·K)]}\]
\[ L = \text{latent heat of evaporation [J/kg] = 2.5 \cdot 10^6 J/kg}\]
\[ \delta_a = \text{water vapor permeability of air [s]}\]
\[ \mu = \text{water vapor resistance factor [-]}\]
\[ g_a = \text{air flux through the material [kg/(m}^2·s)]\]
\[ \varphi = \text{relative humidity [-]}\]
\[ p_{sat} = \text{saturation pressure [Pa]}\]
\[ \rho_w = \text{density of water [1000 kg/m}^3]\]
\[ R_v = \text{gas constant of water [J/(kg·K)] = 461.89 J/(kg·K)}\]
\[ p_c = \text{capillary pressure [Pa]}\]
\[ L_{p_c} = \text{logarithmic capillary pressure [Pa]}\]
\[ k_m = \text{liquid permeability [s]}\]

The formulas (18) and (19) combined in matrix form:
\[
\begin{bmatrix}
(p \cdot c + c_i \cdot w) & 0 & \frac{\partial w}{\partial r_c} & \frac{\partial p_c}{\partial L_{p_c}}
\end{bmatrix}
\begin{bmatrix}
\frac{\partial T}{\partial t}
\end{bmatrix}
\]
\[
\nabla \left( \lambda + L \cdot \frac{\delta_a}{\mu} \cdot \frac{\partial p_{sat}}{\partial T} \right)
\left[ -L \cdot \frac{\delta_a}{\mu} \cdot \frac{\partial p_{sat(T)}}{\partial T} \cdot \frac{\varphi}{\rho_{w}R_vT} \cdot \frac{\partial p_c}{\partial L_{p_c}} \right)
\left[ -C_p + L \cdot 0.62 \cdot 10^{-5} \cdot \varphi \cdot \frac{\partial p_{sat}}{\partial T} \right)
\left[ -0.62 \cdot 10^{-5} \cdot \varphi \cdot \frac{\partial p_{sat}}{\partial T} \right)
\left[ 0.62 \cdot 10^{-5} \cdot \frac{\partial p_{sat(T)}}{\partial T} \cdot \frac{\varphi}{\rho_{w}R_vT} \cdot \frac{\partial p_c}{\partial L_{p_c}} \right)
\end{bmatrix}
\left[ -\nabla \left( T \cdot p_{sat} \cdot L_{p_c} \right) \right)
\]
\[
\nabla \left( T \cdot p_{sat} \cdot L_{p_c} \right)
\]

**Implementation in COMSOL and Matlab**

The \( \frac{\partial T}{\partial t}, \frac{\partial p_{c}}{\partial t}, \nabla T \) and \( \nabla p_{c} \) are calculated with COMSOL, while the other parts are implemented in COMSOL as interpolation functions of the text-files generated by Matlab. These functions are implemented in different parameters. HAM-BC works with *Coefficient Form PDE*, which uses the equations:

\[
\begin{align*}
\left\{ e_a \cdot \frac{\partial^2 u}{\partial t^2} + d_a \cdot \frac{\partial u}{\partial t} + \nabla \cdot \left( -c \cdot \nabla u - \alpha \cdot u + \gamma \right) + \beta \cdot \nabla u + a \cdot u = f \quad \text{in } \Omega \\
- n \cdot \left( -c \cdot \nabla u - \alpha \cdot u + \gamma \right) = g - q \cdot u \quad \text{on } \partial \Omega
\end{align*}
\]

The \( u \) stands for the dependent variables \( T \) and \( L_{p_c} \). The \( \Omega \) stands for the computational domain, which is the union of all sub-domains. The domain boundary is symbolized by \( \partial \Omega \). The \( n \) symbolizes the outward unit normal vector on \( \partial \Omega \). The parameters \( e_a, a, \gamma, a, f, q \) are zero in HAM-BC. This leads to:

\[
\begin{align*}
\left\{ d_a \cdot \frac{\partial u}{\partial t} + \nabla \cdot \left( -c \cdot \nabla u \right) + \beta \cdot \nabla u = 0 \quad \text{in } \Omega \\
- n \cdot \left( -c \cdot \nabla u \right) = g \quad \text{on } \partial \Omega
\end{align*}
\]
The $g$ stands for the boundary conditions. The formulas (18) and (19) are implemented in the damping coefficient $d_a$, diffusion coefficient $c$ and convection coefficient $\beta$.

$$d_a = \begin{bmatrix} (\rho \cdot c + c_l \cdot w) & 0 \\ 0 & \frac{\partial w}{\partial p_c} \cdot \frac{\partial p_c}{\partial L_P c} \end{bmatrix}$$

$$c = \begin{bmatrix} \lambda + L \cdot \frac{\delta_a}{\mu} \cdot \phi \cdot \frac{\partial p_{\text{sat}}}{\partial T} & -L \cdot \frac{\delta_a}{\mu} \cdot p_{\text{sat}(T)} \cdot \frac{\phi}{\rho w R_p T} \cdot \frac{\partial p_c}{\partial L_P c} \\ \frac{\delta_a}{\mu} \cdot \phi \cdot \frac{\partial p_{\text{sat}}}{\partial T} & -k_m \cdot \frac{\partial p_c}{\partial L_P c} \end{bmatrix}$$

$$\beta = g_a \cdot \begin{bmatrix} -\left( \frac{\partial p_{\text{sat}}}{\partial T} \right) \cdot \frac{\phi}{\rho w R_p T} \cdot \frac{\partial p_c}{\partial L_P c} \\ -0.62 \cdot 10^{-5} \cdot \phi \cdot \frac{\partial p_{\text{sat}}}{\partial T} \\ 0.62 \cdot 10^{-5} \cdot \frac{\partial p_{\text{sat}}}{\partial T} \cdot \frac{\phi}{\rho w R_p T} \cdot \frac{\partial p_c}{\partial L_P c} \end{bmatrix}$$

All sub formulas in the matrix are separate coefficients, which have the following names:

$$\begin{align*}
B_T(L_P c, T) &= (\rho \cdot c + c_l \cdot w) \\
B_L(L_P c, T) &= \frac{\partial w}{\partial p_c} \cdot \frac{\partial p_c}{\partial L_P c} \\
D_{11}(L_P c, T) &= \lambda + L \cdot \frac{\delta_a}{\mu} \cdot \phi \cdot \frac{\partial p_{\text{sat}}}{\partial T} \\
D_{12}(L_P c, T) &= -L \cdot \frac{\delta_a}{\mu} \cdot p_{\text{sat}(T)} \cdot \frac{\phi}{\rho w R_p T} \cdot \frac{\partial p_c}{\partial L_P c} \\
D_{21}(L_P c, T) &= \frac{\delta_a}{\mu} \cdot \phi \cdot \frac{\partial p_{\text{sat}}}{\partial T} \\
D_{22}(L_P c, T) &= -k_m \cdot \frac{\partial p_c}{\partial L_P c} \end{align*}$$

The formulas for each coefficient are:

$$\begin{align*}
B_T(L_P c, T) &= (\rho \cdot c + c_l \cdot w) \\
B_L(L_P c, T) &= \frac{\partial w}{\partial p_c} \cdot \frac{\partial p_c}{\partial L_P c} \\
D_{11}(L_P c, T) &= \lambda + L \cdot \frac{\delta_a}{\mu} \cdot \phi \cdot \frac{\partial p_{\text{sat}}}{\partial T} \\
D_{12}(L_P c, T) &= -L \cdot \frac{\delta_a}{\mu} \cdot p_{\text{sat}(T)} \cdot \frac{\phi}{\rho w R_p T} \cdot \frac{\partial p_c}{\partial L_P c} \\
D_{21}(L_P c, T) &= \frac{\delta_a}{\mu} \cdot \phi \cdot \frac{\partial p_{\text{sat}}}{\partial T} \\
D_{22}(L_P c, T) &= -k_m \cdot \frac{\partial p_c}{\partial L_P c}
\end{align*}$$

The $B$ stands for Buffer, the $D$ for Diffusion and $C$ for Convection.

Figure 2: Location of the coefficients in the matrix used in HAM-BC. The $\frac{\partial T}{\partial t}$ is from the heat balance equation, while the $\frac{\partial L_P c}{\partial t}$ is from the moisture balance equation.
2. Matlab

Each coefficient has its own Matlab script. The text-files with the tables of each coefficient are generated by running the script `Crefun.m`. Another Matlab script is `Pv.m`, which determines the vapor pressure related to temperature (T) and logarithmic capillary pressure (LPC).

If the construction consists of several different materials, each material requires its own B-coefficients and D-coefficients. The letters a, b, c etcetera are placed behind the coefficient name. In this manual, a construction made out of a single material is used as example. The C-coefficients are material independent, because it is based on the properties of air. The material dependent value of air permeability will be inserted in COMSOL and not in the C-coefficients.

Two methods to insert dependent material properties are discussed in this manual, which are:
- Using a formula or constant values.
- Using a text-file of a material from Delphin.

The Matlab-codes in this manual can be directly copied into Matlab-scripts. It is possible that when you copy a code which is on several pages, the page number of this manual will be copied as well.

List of abbreviations in the Matlab-codes

The following abbreviations are used in the Matlab-files:

<table>
<thead>
<tr>
<th>Description</th>
<th>Dimension</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPc</td>
<td>logarithmic capillary pressure</td>
</tr>
<tr>
<td>Pc</td>
<td>capillary pressure</td>
</tr>
<tr>
<td>Tc</td>
<td>temperature in Celsius</td>
</tr>
<tr>
<td>Tk</td>
<td>temperature in Kelvin</td>
</tr>
<tr>
<td>phi</td>
<td>relative humidity (ratio)</td>
</tr>
<tr>
<td>rho</td>
<td>specific density of material</td>
</tr>
<tr>
<td>c</td>
<td>specific heat capacity of material</td>
</tr>
<tr>
<td>w</td>
<td>moisture content</td>
</tr>
<tr>
<td>lambda</td>
<td>thermal conductivity coefficient</td>
</tr>
<tr>
<td>psat</td>
<td>saturation pressure</td>
</tr>
<tr>
<td>dpsatdT</td>
<td>differentiation of saturation pressure over temperature</td>
</tr>
<tr>
<td>deltap</td>
<td>vapor permeability coefficient</td>
</tr>
<tr>
<td>mu</td>
<td>vapor diffusion resistance factor</td>
</tr>
<tr>
<td>K</td>
<td>liquid permeability = liquid water conductivity</td>
</tr>
<tr>
<td>cp</td>
<td>heat capacity of air</td>
</tr>
</tbody>
</table>

Global values

`Crefun.m` defines global values, which are used in the other scripts.

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
<th>Dimension</th>
</tr>
</thead>
<tbody>
<tr>
<td>rho_w</td>
<td>specific density of water</td>
<td>1000</td>
</tr>
<tr>
<td>R</td>
<td>universal gas constant</td>
<td>8.314</td>
</tr>
<tr>
<td>Mw</td>
<td>molar weight of water</td>
<td>0.018</td>
</tr>
<tr>
<td>Ry</td>
<td>gas constant of water vapor</td>
<td>462</td>
</tr>
<tr>
<td>L</td>
<td>latent heat of evaporation</td>
<td>2.5·10⁶</td>
</tr>
<tr>
<td>deltab</td>
<td>vapor permeability coefficient of air</td>
<td>1.8·10⁻¹⁰</td>
</tr>
</tbody>
</table>
Crefun.m

The following Matlab code of Crefun.m is for one material. In the case of several materials, than more “FunName” must be added under the existing list. Also the last number at the index (i) must be changed to the new total number of amount of FunName. In Appendix A, an optional code is shown, which generates plots of the results of the coefficients in Matlab when it is copied under the following code of Crefun.m. However, the plotting of those graphs costs the most computational time.

```matlab
% CREFUN
close all
clear all

global rho_w R Mw Rv L deltaa
rho_w=1000; R=8.314;
Mw=0.018; L=2.5*10^6; Rv=R/Mw;
deltaa=1.8*10^-10;

%%
x=[-10 0:0.01:10 20]; %Lpc
nx=length(x);
y=[-10:0.05:40]; %T
ny=length(y);
[xx,yy] = meshgrid(x,y);

FunName{1}='Pv';
FunName{2}='D11a';
FunName{3}='D12a';
FunName{4}='D21a';
FunName{5}='D22a';
FunName{6}='BLa';
FunName{7}='BTa';
FunName{8}='C11';
FunName{9}='C12';
FunName{10}='C21';
FunName{11}='C22';

for i=1:11 %Change last number in case of more coefficients
    eval(['data=' FunName{i} '(xx,yy);']);
data(:,1)=data(:,2);
data(:,nx)=data(:,(nx-1));
eval([FunName{i} 'fun.x=x;']);
eval([FunName{i} 'fun.y=y;']);
eval([FunName{i} 'fun.data=data;']);
end

Constant Coefficients
Several coefficients are constant, such as the vapor pressure (Pv) and the convection coefficients (C11, C12, C21 and C22). The C-coefficients are based on the material properties of air; therefore, the scripts for the C-coefficients do not have to be changed.
```
**Pv.m**

```matlab
%Pv
function pr=Pv(LPc,Tc)
global rho_w Rv
Pc=10.^LPc;
Tk=Tc+273.15;
phi=exp(-Pc./(rho_w.*Rv.*Tk));
psat=exp(65.8094-7066.27./Tk-5.976*log(Tk));
pr=psat.*phi;
```

**C11.m**

```matlab
% C11 = -(cp + L*0.62e-5*phi*dPsat/dT)
function y=C11(LPc,Tc)
global rho_w L Rv
Pc=10.^LPc;
Tk=Tc+273.15;
phi=exp(-Pc./(rho_w.*Rv.*Tk));
dpsatdT=(7066.27./Tk.^2-5.976./Tk).*exp(65.8094-7066.27./Tk-5.976*log(Tk));
cp=1250/1.2;
y=-(cp+(L.*0.62e-5.*phi.*dpsatdT))+LPc.*0;
```

**C12.m**

```matlab
% C12 = L*0.62e-5*phi*Psat*(Mw/rho*R*T)*dPc/dLPc
function y=C12(LPc,Tc)
global rho_w R Mw L Rv
Pc=10.^LPc;
Tk=Tc+273.15;
phi=exp(-Pc./(rho_w.*Rv.*Tk));
psat=exp(65.8094-(7066.27./Tk)-5.976*log(Tk));
y=L.*0.62e-5.*phi.*psat.*(Mw./(rho_w.*R.*Tk)).*(Pc*log(10));
```

**C21.m**

```matlab
% C21 = -0.62e-5*phi*dPsat/dT
function y=C21(LPc,Tc)
global rho_w Rv
Pc=10.^LPc;
Tk=Tc+273.15;
phi=exp(-Pc./(rho_w.*Rv.*Tk));
dpsatdT=(7066.27./Tk.^2-5.976./Tk).*exp(65.8094-7066.27./Tk-5.976*log(Tk));
y=-0.62e-5.*phi.*dpsatdT;
```

**C22.m**

```matlab
% C22 = 0.62e-5*phi*Psat*(Mw/rho*R*T)*dPc/dLPc
function y=C22(LPc,Tc)
global rho_w R Mw Rv
Pc=10.^LPc;
Tk=Tc+273.15;
phi=exp(-Pc./(rho_w.*Rv.*Tk));
psat=exp(65.8094-(7066.27./Tk)-5.976*log(Tk));
y=0.62e-5.*phi.*psat.*(Mw./(rho_w.*R.*Tk)).*(Pc*log(10));
```
Material dependent coefficients

The following coefficients are dependent on the used materials in the simulated construction. If there are several different materials, than several of the here below mentioned coefficients must be copied and distinguished by a, b, c, etcetera. The coefficients which are dependent on the material properties are $BT$, $BL$, $D_{11}$, $D_{12}$, $D_{21}$ and $D_{22}$. These are the diffusion coefficients and buffering coefficients. In this manual, there is an a added to the coefficients name; however, only one material example is used. Two examples of every coefficient are given: first with formulas and the second consisting of material properties from Delphin. In Appendix B, the implementation of material properties with self-made tables is shown.

Extra: It is possible to multiply the values of the material properties from Delphin. [Goesten 2016] knew the measured data of the materials used in the validation study and chooses a Delphin-material with the same shape of material properties and multiplied the data with a factor to get the approximate values. It combines the advantage of the many data points of Delphin-materials with the similar value as the given material data.

**BTa.m (with formula)**

```matlab
% BT = rho*c + w*cl
function y=BTa(LPc,Tc)
global rho_w R Mw L Rv

% Material Independent
Pc=10.^LPc;
Tk=Tc+273.15;

% Material Dependent
w=209.*(1.*(1+(2e-6.*Pc).^1.27).^((1-1.27)./1.27));
rho=790;
c=870;

y=(rho*c)+(4200.*w);
```

**BTa.m (with data file from Delphin)**

```matlab
% BT = rho*c + w*cl
function y=BTa(LPc,Tc)
global rho_w R Mw L Rv

% Material Independent
Pc=10.^LPc;
Tk=Tc+273.15;

% Material Dependent
load Normal_Brick_(2)\Thetal(pC).out
por=0.35;
pC_Thetal_pC=Thetal_pC(:,1);
Thetal_Thetal_pC=Thetal_pC(:,2);
w_Pc=Thetal_Thetal_pC*rho_w*por;
w=interp1(pC_Thetal_pC,w_Pc,Pc,'linear','extrap');

rho=790;
c=870;

y=(rho*c)+(4200.*w);
```
BLa.m  (with formula)

```matlab
% BL = (dw/dPc)*(dPc/dLpc)
function y=BLa(LPc,Tc)
  global rho_w Rv

  % Material Independent
  Pc=10.^LPc;
  Tk=Tc+273.15;

  % Material Dependent
  w=209.*(1.*(1+(2e-6.*Pc).^1.27).^((1-1.27)./1.27));

  % dw/dPc (differentiation of moisture content)
  dwdpc1=(diff(w')./diff(Pc'));
  dwdpc11=[dwdpc1;dwdpc1((length(dwdpc1)),:)];
  dwdpc=dwdpc11';

  y=(dwdpc).*(Pc.*(log(10)));
```

BLa.m  (with data from Delphin)

```matlab
% BL = (dw/dPc)*(dPc/dLpc)
function y=BLa(LPc,Tc)
  global rho_w R Mw Rv

  % Material Independent
  Tk=Tc+273.15;
  Pc=10.^LPc;

  % Material Dependent
  load Normal_Brick_(2)\Thetal(pC).out
  por=0.35;
  pC_Thetal_pC=Thetal_pC(:,1);
  Thetal_Thetal_pC=Thetal_pC(:,2);
  w_Pc=Thetal_Thetal_pC.*rho_w.*por;
  w=interp1(pC_Thetal_pC,w_Pc,Pc,'linear','extrap');
  dwdpc1=(diff(w')./diff(Pc'));
  dwdpc11=[dwdpc1;dwdpc1((length(dwdpc1)),:)];
  dwdpc=dwdpc11';

  y=(dwdpc).*(Pc.*log(10));
```

D11a.m  (with formula)

```matlab
% D11 = lambda + L*deltap*phi*dPsat/dT
function y=D11a(LPc,Tc)
  global rho_w R Mw L Rv

  % Material Independent
  Pc=10.^LPc;
  Tk=Tc+273.15;
  phi=exp(-Pc./(rho_w.*Rv.*Tk));
  dpsatdT=((7066.27./Tk.^2-5.976./Tk).*exp(65.8094-7066.27./Tk-5.976*log(Tk)));

  % Material Dependent
  w=209.*(1.*(1+(2e-6.*Pc).^1.27).^((1-1.27)./1.27));
  lambda=0.5+0.0045.*w;
  deltap=(Mw./(R.*Tk)).*(2e-6.*(((1-(w./17))./((0.3.*((1-(w./17)).^2)+0.5)))));

  y=lambda+(L.*deltap.*phi.*dpsatdT);
```
D11a.m (with data from Delphin)

```matlab
% D11 = lambda + L*deltap*phi*dPsat/dT
function y=D11a(LPc,Tc)
global rho_w R Mw L Rv

% Material Independent
Pc=10.^LPc;
Tk=Tc+273.15;
phi=exp(-Pc./(rho_w.*Rv.*Tk));
dpsatdT=(7066.27./Tk.^2-5.976./Tk).*exp(65.8094-7066.27./Tk-5.976*log(Tk));

% Material Dependent
load Normal_Brick_(2)\Thetal(pC).out
por=0.35;
pC_Thetal_pC_Thetal_pC_(:,1);
Thetal_Thetal_pC_Thetal_pC_(:,2);
w_Thetal_Thetal_pC_Thetal_pC_(:,:);
w=interp1(pC_Thetal_pC, w_Thetal_pC, por,'linear','extrap');

load Normal_Brick_(2)\lambda(Thetal).out
Thetal=Thetal.*Tk;
lambdaTh=lambda_Thetal_(:,1);
lambda=lambdaTh(:,2);
lambda=interp1(Th,lambdaTh,por,'linear','extrap');

phi2=exp(-Pc./(rho_w.*Rv.*293.15));
load Normal_Brick_(2)\Mew(RH).out
RH=Mew_RH_(:,1);
Mew=Mew_RH_(:,2);
mu=interp1(RH,Mew,phi2,'linear','extrap');
deltap=deltap/mu;

y=lambda+(L.*deltap.*phi.*dPsatdT);
```

D12a.m (with formula)

```matlab
% D12 = - L*deltap*phi*Psat*(Mw/rho*R*T)*dPc/dLPc
function y=D12a(LPc,Tc)
global rho_w R Mw L Rv

% Material Independent
Pc=10.^LPc;
Tk=Tc+273.15;
phi=exp(-Pc./(rho_w.*Rv.*Tk));
psat=exp(65.8094-7066.27./Tk-5.976*log(Tk));

% Material Dependent
w=209.*(1.*((1-2e-6.*Pc).*1.27).^((1-1.27)/1.27));
deltap=(Mw./(R.*Tk)).*(2e-6.*((1-(w./17))./(0.3.*((1-(w./17)).^2)+0.5))));

y=-L.*deltap.*phi.*Psat.*((Mw./(rho_w.*R.*Tk)).*Pc.*log(10));
```

D12a.m (with data from Delphin)

```matlab
% D12 = - L*deltap*phi*Psat*(Mw/rho*R*T)*dPc/dLPc
function y=D12a(LPc,Tc)
global rho_w R Mw L Rv

% Material Independent
Pc=10.^LPc;
Tk=Tc+273.15;
phi=exp(-Pc./(rho_w.*Rv.*Tk));
psat=exp(65.8094-7066.27./Tk-5.976*log(Tk));
```
```matlab
% Material Dependent
load Normal_Brick_(2)\Tetal\pC\out
por=0.35;
pC_Tetal_pC=Tetal\pC\(:,1);
Tetal_Tetal_pC=Tetal\pC\(:,2);
w_Pc=Tetal\Tetal\pC\*rho_w*por;
w=interp1(pC_Tetal_pC,w_Pc,Pc,'linear','extrap');

phi2=exp(-Pc./(rho_w.*Rv.*293.15));
load Normal_Brick_(2)\Mew\RH\out
RH=Mew\RH\(:,1);
Mew=Mew\RH\(:,2);
mu=interp1(RH,Mew,phi2,'linear','extrap');
deltap=deltaa./mu;

y=-L.*deltap.*phi.*psat.*(Mw./(rho_w.*R.*Tk)).*(Pc.*log(10));

D21a.m (with formula)

% D21 = deltap*phi*dpstatdT
function y=D21a(LPc,Tc)
global rho_w R Mw Rv

% Material Independent
Pc=10.^LPc;
Tk=Tc+273.15;
phi=exp(-Pc./(rho_w.*Rv.*Tk));
dpsatdT=(7066.27./Tk.^2-5.976./Tk).*exp(65.8094-7066.27./Tk-5.976*log(Tk));

% Material Dependent
w=209.*(1.*(1+2e-6.*Pc).^1.27).*((1-1.27)./1.27));
deltap=(Mw./(R.*Tk)).*(2e-6.*((1-(w./17))./((0.3.*(1-(w./17)).^2))+0.5)));

y=deltap.*phi.*dpsatdT;

D21a.m (with data from Delphin)

% D21 = deltap*phi*dpstatdT
function y=D21a(LPc,Tc)
global rho_w R Mw Rv

% Material Independent
Pc=10.^LPc;
Tk=Tc+273.15;
phi=exp(-Pc./(rho_w.*Rv.*Tk));
dpsatdT=(7066.27./Tk.^2-5.976./Tk).*exp(65.8094-7066.27./Tk-5.976*log(Tk));

% Material Dependent
load Normal_Brick_(2)\Tetal\pC\out
por=0.35;
pC_Tetal_pC=Tetal\pC\(:,1);
Tetal_Tetal_pC=Tetal\pC\(:,2);
w_Pc=Tetal\Tetal\pC\*rho_w*por;
w=interp1(pC_Tetal_pC,w_Pc,Pc,'linear','extrap');

phi2=exp(-Pc./(rho_w.*Rv.*293.15));
load Normal_Brick_(2)\Mew\RH\out
RH=Mew\RH\(:,1);
Mew=Mew\RH\(:,2);
mu=interp1(RH,Mew,phi2,'linear','extrap');
deltap=deltaa./mu;

y=deltap.*phi.*dpsatdT;
```
D22a.m (with formula)

```matlab
function y=D22a(aLPc,Tc)
global rho_w R Mw Rv

% Material Independent
Pc=10.^aLPc;
Tk=Tc+273.15;
phi=exp(-Pc./(rho_w.*Rv.*Tk));
psat=exp(65.8094-7066.27./Tk-5.976*log(Tk));

% Material Dependent
phi2=exp(-Pc./(rho_w.*Rv.*293.15));
load Normal_Brick_(2)\Mew(RH).out
RH=Mew_RH_(:,1);
Mew=Mew_RH_(:,2);
mu=interp1(RH,Mew,phi2,'linear','extrap');
deltap=deltaa./mu;
load Normal_Brick_(2)\Kl(pC).out
pC_Kl_pC=Kl_pC_(:,1);
Kl_Kl_pC=Kl_pC_(:,2);
K=interp1(pC_Kl_pC,Kl_Kl_pC,Pc,'linear','extrap');

y=-K.*(Pc.*log(10))-deltap.*phi.*psat.*(Mw./(rho_w.*R.*Tk)).*(Pc.*log(10));
```

D22a.m (with data from Delphin)

```matlab
function y=D22a(aLPc,Tc)
global rho_w R Mw Rv deltaa

% Material Independent
Pc=10.^aLPc;
Tk=Tc+273.15;
phi=exp(-Pc./(rho_w.*Rv.*Tk));
psat=exp(65.8094-7066.27./Tk-5.976*log(Tk));

% Material Dependent
phi2=exp(-Pc./(rho_w.*Rv.*293.15));
load Normal_Brick_(2)\Mew(RH).out
RH=Mew_RH_(:,1);
Mew=Mew_RH_(:,2);
mu=interp1(RH,Mew,phi2,'linear','extrap');
deltap=deltaa./mu;
load Normal_Brick_(2)\Kl(pC).out
pC_Kl_pC=Kl_pC_(:,1);
Kl_Kl_pC=Kl_pC_(:,2);
K=interp1(pC_Kl_pC,Kl_Kl_pC,Pc,'linear','extrap');

y=-K.*(Pc.*log(10))-deltap.*phi.*psat.*(Mw./(rho_w.*R.*Tk)).*(Pc.*log(10));
```
3. COMSOL Multiphysics

In COMSOL behind the input-boxes there are dimension noted, but these have no meaning when using HAM-BC. The temperatures must be inserted in degrees Celsius. In figure 3, the Model Tree is shown. The Model Tree shows the main COMSOL-input of HAM-BC, when all the input is inserted. Certain names are used in the manual, which refers to branches or parts shown in figure 3.

Inserting variables

Right-click on Definitions under the Global-branche and choose Variables. Certain variables are defined here to make the input in COMSOL shorter; and therefore, decrease the chance of mistakes. The vapor gas constant (Rv) and the latent heat (L) is always the same value. The air permeability is dependent on the used materials. When using more materials, then there must be several k-values such as ka, kb, and kc etcetera. If the influence of the convection heat and mass transfer is omitted, than the air permeability is unnecessary. The surface coefficients are dependent on the boundary conditions.

<table>
<thead>
<tr>
<th>Name</th>
<th>Expression</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ka</td>
<td>1*10^-5</td>
<td></td>
<td>Air permeability</td>
</tr>
<tr>
<td>Rv</td>
<td>8.314/0.018</td>
<td></td>
<td>Vapor Gas Constant</td>
</tr>
<tr>
<td>L</td>
<td>2.5e6</td>
<td></td>
<td>Latent heat</td>
</tr>
<tr>
<td>he</td>
<td>20</td>
<td></td>
<td>Surface coefficient external</td>
</tr>
<tr>
<td>hi</td>
<td>10</td>
<td></td>
<td>Surface coefficient internal</td>
</tr>
<tr>
<td>beta_e</td>
<td>1.882e-7</td>
<td></td>
<td>Surface coefficient of vapor transfer external</td>
</tr>
<tr>
<td>beta_i</td>
<td>5.862e-8</td>
<td></td>
<td>Surface coefficient of vapor transfer internal</td>
</tr>
</tbody>
</table>

The values for air permeability are hard to find. The air permeability values of several materials are:
- Wood: 5.5·10^-6 (s) [Uittenbosch 2012]
- Crack at wood: 5.5·10^-5 (s) [Uittenbosch 2012]
- Mineral wool: 1·10^-3 (s) [Delphin]
- Insulation: 2.5·10^-5 (s) [Uittenbosch 2012]
- Stone: 4.4·10^-5 (s) [Uittenbosch 2012]

Inserting coefficients

The text-files generated with Matlab must be inserted in COMSOL. This is done in the Global-branche in the part Definitions. Right-click on Definitions, and then choose Functions > Interpolation. In the case of the coefficients, the Data Source is a File with the data format Grid. The interpolation is Linear and the extrapolation is Constant. The names inside the brackets in the Model Tree are the names which need to be inserted inside the boxes. It is not required that the name before the brackets are the same as in the brackets.

Extra: To send a COMSOL-file to someone else, you can import all coefficients in the model. In that case you do not have to send all the text-files.
Inserting tables for boundary conditions
The indoor temperature (Ti) and external temperature (Te) in [°C] are required. For moisture it can be the indoor vapor pressure (Pi) and external vapor pressure (Pe) in [Pa] and/or the indoor relative humidity (RHi) and the external relative humidity (RHe) in [-] or [%]. The air pressure difference (dPa) in [Pa] is required, when convection is taken into account. This can be implemented with the interpolation function in the form of a file or as a table. The boundary conditions require the time in seconds in the first column and the boundary condition values in the second column. Interpolation is done with Linear or Piecewise cubic and the Extrapolation with Constant.

Solar irradiation can be implemented in [W/m²].

Rain can be implemented with the dimension [kg/(m²∙s)]. It is possible to estimate the horizontal wind-driven rain (WDR) with Appendix C.

Adding component
A component is a set of geometry and the corresponding physics model. Click on the button Add Component and determine if it is a 1D, 2D or a 3D problem. If there is one-directional flow or transfer, it is possible to choose 2D to create a more presentable result by the use of surface plots. If it is a cylinder, than the axisymmetric version of the 2D or 3D-model can be chosen.

Geometry
In this manual, a 2D model will be used as example. By right-clicking on the geometry you can choose the shape which you want to draw.

It is important to determine in this stage on which points the dependent variables are assessed. These points must be added by right-clicking on Geometry and choosing point. This is important, because by using the points from the geometry, COMSOL automatically creates nodes on these points when the mesh is created. If the assessment is not done on a node, than the results on that point is determined with a linear or piecewise cubic interpolation between the nearest nodes, which can cause large errors.

Physics
Click on Add Physics and choose Coefficient Form PDE.

Thereafter, click on Coefficient Form PDE and define two dependent variables by typing a 2 at Number of dependent variables. The dependent variables are T for temperature and LPc for the logarithmic capillary pressure. It is important to name the upper dependent variable T and the lower LPc, because otherwise the locations of inserting coefficients from Matlab will be different.
Coefficient Form PDE
By left clicking on the Coefficient Form PDE 1, the functions based on the text-files can be inserted. Every material has its own Coefficient Form PDE; therefore, extra Coefficient Form PDE must be implemented, if there are several different materials. This can be added by right-clicking on the Coefficient Form PDE (c).

At the Diffusion Coefficient (c) the D-functions must be inserted. Behind the coefficient name, the LPC and T must be typed down in brackets as shown in the figure. It is important to type first LPC and then the T. In this way the LPC and T calculated with COMSOL is inserted into the coefficients. The values are always Isotropic.

The Absorption Coefficient (a), Source Term (f), Mass Coefficient (ea), Conservative Flux Convection Coefficient (a) and the Conservative Flux Source (γ) must be zero.

At the Damping or Mass Coefficient, the BT and BL-functions must be inserted in the way shown in the print screen.

In the box of the Convection Coefficient, the C-coefficients multiplied by the air flux through the construction (ga) must be inserted. The simple convection method to determine the ga is with the formula: \(dPa(t) \times ka \times 1.2\). The 1.2 in the formula is the density of air. This is multiplied by the air permeability (ka) and the pressure difference (dPa) between the two boundaries. The minus before the coefficient is important.

A more sophisticated method to determine the air flux (ga) is explained in Appendix D. According to [Goesten 2016] this sophisticated formula generate for constructions without an air cavity to the same results. But for constructions with an air cavity, it is advised to simulate once with the sophisticated convection method and once with the simple convection method.

Boundary Conditions
The boundary conditions used in HAM-BC are implemented with the type flux/source. Boundaries which are adiabatic are implemented as zero flux, which is always automatically used by COMSOL. The adding of flux/source will diminish the amount of zero flux boundaries.

The above box of g is for the boundary conditions related to the heat balance equation and the lower insert box is for the boundary conditions related to the moisture balance equation.

In this section, the boundary conditions are given for diffusion and infiltration of air, including the influence of latent heat. After that, the additional and optional boundary conditions are given, such as solar irradiation and rain.

It is important to reverse the boundary conditions related to air flux (ga) at the side where the air leaves the material – when the pressure difference is positive – the temperature and pressure difference between ambient and surface.
First the boundary conditions are mentioned which must be used when the vapor pressure is known. The following formulas are described as **if the air leaves the external side when the air pressure difference is positive**.

**External boundary:**

\[
he(\text{Te}(t)-T)+\beta_e L(\text{Pe}(t)-\text{Pv}(\text{LPc},T))+ka \cdot 1.2 \cdot \text{dPa}(t) \cdot 1005 \cdot (\text{Te}(t)-T)+L \cdot k_a \cdot 1.2 \cdot \text{dPa}(t) \cdot 0.62 \cdot e^{-5} \cdot (\text{Pv}(\text{LPc},T)-\text{Pe}(t))
\]

\[
\text{Beta}_e(\text{Pe}(t)-\text{Pv}(\text{LPc},T))+ka \cdot 1.2 \cdot \text{dPa}(t) \cdot 0.62 \cdot e^{-5} \cdot (\text{Pv}(\text{LPc},T)-\text{Pe}(t))
\]

When the boundary vapor pressure is not known, the Pe(t) can be changed to:

\[
\text{Pv}\left(\log_{10}\left(\text{abs}\left(1000 \cdot \text{Rv} \cdot (\text{Te}(t)+273.15)\right)\cdot \text{abs}\left(\log\left(\text{RH}_e(t)/100\right)\right)\right),\text{Te}(t)\right)
\]

**Internal boundary:**

\[
hi(\text{Ti}(t)-T)+\beta_i L(\text{Pi}(t)-\text{Pv}(\text{LPc},T))+ka \cdot 1.2 \cdot \text{dPa}(t) \cdot 1005 \cdot (\text{Ti}(t)-T)+L \cdot k_a \cdot 1.2 \cdot \text{dPa}(t) \cdot 0.62 \cdot e^{-5} \cdot (\text{Pi}(t)-\text{Pv}(\text{LPc},T))
\]

\[
\text{Beta}_i(\text{Pi}(t)-\text{Pv}(\text{LPc},T))+ka \cdot 1.2 \cdot \text{dPa}(t) \cdot 0.62 \cdot e^{-5} \cdot (\text{Pi}(t)-\text{Pv}(\text{LPc},T))
\]

When the boundary vapor pressure is not known, the Pi(t) can be changed to:

\[
\text{Pv}\left(\log_{10}\left(\text{abs}\left(1000 \cdot \text{Rv} \cdot (\text{Ti}(t)+273.15)\right)\cdot \text{abs}\left(\log\left(\text{RH}_i(t)/100\right)\right)\right),\text{Ti}(t)\right)
\]

Another method to change the relative humidity to vapor pressure is the implementation of an **analytical function** at the Definitions-branche of the saturation pressure (Psat) with the formula

\[
\exp(65.8094-7066.27/(x+273.15)-5.976 \cdot \log(x+273.15)).
\]

Thereafter, creating at the Definitions-branche the variables Pe and Pi with \((\text{RHe}(t)/100) \cdot \text{Psat}(\text{Te}(t))\) and \((\text{RHi}(t)/100) \cdot \text{Psat}(\text{Ti}(t))\). The only difference with the above mentioned boundary conditions is that Pe(t) must be written as Pe and Pi(t) as Pi, because it is now an **analytical function** instead of an interpolation function.

**Solar irradiation**

The solar irradiation (Esol) in [W/m²] must be multiplied with the absorption factor of the external material. If it is measured at the same plane as the construction, this value can be directly used. Otherwise, the angle must be taken into account. **Angles in COMSOL must be inserted in radians, whereas π can be inserted as pi**. For example, a material with an absorption factor of 0.55 and an angle of 90 degrees (0.5π rad) between the measured plane and the normal of the façade has the following formula:

\[0.55 \cdot \text{Esol}(t) \cdot \cos(0.5 \cdot \pi)\]

**Rain**

Rain must be applied in kg/(m²·s). Wind-driven rain (WDR) can be estimated with the Matlab-code in Appendix C. When a material is saturated with water – i.e. the saturation capillary moisture content is reached – than the incoming rain will not be absorbed by the material and the rain will drop off. To simulate this effect, an **interpolation function** must be made – called satu – in table form at the Definitions-branche of the Global-branche which contains the saturation capillary moisture content value and states that the value of the function is zero when it becomes higher than the saturation moisture content. An example is shown right when the saturation moisture content is 225 kg/m³. In the first column the values of zero, saturation moisture content minus 0.01, the saturation moisture content and 1000 kg/m³ (or another high number) must be added.
A function must be added to calculate the moisture content of the material(s) which are located at the outside of the construction – called for example w. If the formula of the moisture content related to LPC is known, than an analytical function must be used. If the moisture retention curve is known, than the interpolation function must be used.

When the moisture retention curve or formula is made with variable p_c, the total rain flux is defined by:

\[ g_{\text{rain}}(t) \times \text{satu}(w(10^{p_c})) \]

It is assumed that the rain temperature is equal to the ambient temperature, without the influence of solar irradiation. The influence of the rain on the heat flux is determined by:

\[ 4200 \times g_{\text{rain}}(t) \times (T_e(t) - T) \]

**Total boundary conditions**

The total boundary conditions are, for example, at the external surface:

The **Boundary Absorption / Impedance Term** \(q\) must be zero.

*Extra: It is possible to use variable surface heat coefficients \(h_e\) and \(h_i\) and surface mass transfer coefficients \(\beta_e\) and \(\beta_i\). Also heat loss by radiation to the sky can be implemented. Extra boundary conditions can be implemented if the boundary conditions of heat have the dimension \([\text{W/m}^2]\) and the dimensions of the boundary conditions for moisture \([\text{kg/(m}^2\cdot\text{s})]\).*

**Initial Value**

The initial value is the value of the dependent values at the beginning of the simulation. The temperature can be chosen directly, but the LPC is dependent on the initial temperature and initial relative humidity. The formula for the initial LPC is for \(T_{\text{initial}} = 20^\circ\text{C}\) and \(R_{\text{H,initial}} = 43\%\):

\[ \log_{10}(\text{abs}(1000\times 462 \times (20 + 273.15) \times \log(43/100))) \]

The \(\log\) in COMSOL means the natural logarithm, while \(\log_{10}\) is the logarithmic with the base of 10. The initial values for the differentiation of the dependent variables over time must be zero.

**Mesh**

By using an automatically generated mesh, nodes are placed on the geometry points. The grid has a large influence on the quality of the simulation. However, a finer grid causes a higher computation time. A grid sensitivity analysis can be made to get a more optimum grid.

**Study**

The study is the calculation method which is used. HAM-BC works only with transient calculation. Click on **Add Study** and choose **Time Dependent**. If a steady state situation must be simulated, than the transient calculation must be done with constant boundary conditions and a large time period to decrease the influence of the initial values.
The starting time, time step and the end of the calculation must be defined in seconds. This must be inserted in the box *Times*, whereas the first value in the brackets is the starting time, the second is the time step and the third is the end time. When time unit seconds (s) is chosen, the calculation is done per second, but the results in the graphs, tables, plots, and etcetera are based on the time step, which has a significant influence on the results depicted in graphs and tables. It is advised to set the relative tolerance on $1e^{-11}$.

**Results**
The results are shown in the *Results*-branch in the form of graphs and tables.

If you want to plot the relative humidity [%], the expression is:

$$\left(\exp\left(-(10^6 LPc)/(1000*462*(T+273.15))\right)\right) \times 100$$

The vapor pressure can be plotted in the results with: $P_v(LPc,T)$

The moisture content of a material can also be plotted. In this case, the moisture content dependent on the capillary pressure is given by a table, than this table must be inserted as an interpolation function. With this function the moisture content could be plotted in graphs.

In Appendix E, a Matlab-code is shown which can be used for determining the %RMSE and the maximum error between simulated results and measured results. The %RMSE is also suitable for grid sensitivity analysis. A Matlab-code to generate a Mollier diagram is added in Appendix F, which can be used to visualize the times that the conditions are suitable for mold growth.

*Extra:* When creating graphs with COMSOL it is possible to define the text of the legend and the description of the axes. To insert subscripts or superscripts such as the degree-sign ($^\circ$) or the square ($^2$) or power to three ($^3$), the easiest way is to open *Microsoft Word* and using the tab *Insert* and choosing for example $^\circ$ or $^2$ and copy-paste it into the descriptions boxes at graphs in COMSOL.

### 4. Damage Indicators

In this chapter, the determining of the damage indicators defined in [Goesten 2016] is explained with a Matlab-script. [Goesten 2016] describes the damage types that may result from insulating historical buildings, excluding thermal cracks.

**COMSOL-part: Exporting**

Right-click on the *Derived Values* at the Result-branch and choose *Point Evaluation*. When defining the *Point Evaluation*, the point must be chosen and the *expression* must be set.

The evaluation is done by right-clicking the *Derived Values*-branch and choosing *Clear and Evaluate All*. The clear part is necessary when the Derived Values are evaluated before, because else COMSOL will add an extra column to the table with the new values. The evaluation creates several tables at the *Table*-subbranch of the *Result*-branch. Table 1 shows the result of evaluating the first input, table 2 of the second, etcetera. Certain tables must be exported in text-files for determining the Damage Indicators. Right click on the *Export*-subbranch of the *Results*-branch, the option *Table* must be chosen to export these results.
Extra: The tables which are generated by COMSOL Multiphysics 5.0 have a default limit of 10,000 rows. When this amount is exceeded, COMSOL automatically removes the rows at the top. This limit can be increased at the setting of each specific table. Imported tables or input files do not have this problem.

The following variables are required to be exported to determine the damage indicators. The Matlab-script of the indicators has the following abbreviations to export text-files:

**Tsi.txt** – Internal surface temperature [°C]
Right-click on Derived Values → choosing Point Evaluation → selecting the required geometrical point at the internal surface → inserting expression: $T$.

**RHsi.txt** – Internal surface relative humidity [%]
Right-click on Derived Values → choosing Point Evaluation → selecting the required geometrical point at the internal surface → inserting expression:

$$\left(\exp\left(-\frac{10^\text{LPc}}{1000*(8.314/0.018)*(T+273.15)}\right)\right) \times 100$$

**Tsi.txt** – External surface temperature [°C]
Right-click on Derived Values → choosing Point Evaluation → selecting the required geometrical point at the external surface → inserting expression: $T$.

**RHse.txt** – External surface relative humidity [%]
Right-click on Derived Values → choosing Point Evaluation → selecting the required geometrical point at the external surface → inserting expression:

$$\left(\exp\left(-\frac{10^\text{LPc}}{1000*(8.314/0.018)*(T+273.15)}\right)\right) \times 100$$

**RHall.txt** – Maximum relative humidity [%]
Right-click on Derived Values and choose Maximum. In case of a 2D-model choose Surface Maximum. In case of a 3D-model choose Volume Maximum. The required expression is:

$$\left(\exp\left(-\frac{10^\text{LPc}}{1000*(8.314/0.018)*(T+273.15)}\right)\right) \times 100$$

**w_wood.txt / wse.txt** – Maximum moisture content [kg/m³] of wood and external materials
This part must be done of wood for wood decay and external materials for frost damage. Right-click on Derived Values and choose Maximum. In the case of a 2D-model choose Surface Maximum. In most cases of a 3D-model choose Volume Maximum. The moisture content formula or interpolation function of the moisture content related to LPc (and maybe T) is required. Every different material has its own relationship between moisture content and relative humidity or (logarithmic) capillary pressure.

**Matlab-part**

The text-files in txt-format are used to assess the damage indicators. These indicators are based on [Goesten 2016], which addresses the main damage types which can occur when insulating external constructions of historical buildings. The assessed damage types are:
- Condensation
- Mold growth and Wood Decay
- Frost Damage
- Salt Damage

The indicator for salt damage only determines the amount of dissolution-crystallization-cycles of two specific salts. These dissolution-crystallization cycles are only causing damage at a certain salt content, which is not simulated with HAM-BC.
The information required for using the script **Damage_Indicators.m** are the internal surface temperature [°C]; external surface temperature [°C]; internal surface relative humidity [%]; the surface relative humidity at the wood [%]; maximum relative humidity of the whole structure [%]; moisture content of the material(s) located at the outside [kg/m³]; and the moisture content [kg/m³] of wood. If there is no wood or timber products in the simulated construction, than the values of moisture content wood and relative humidity at the surface of wood must be zero. The specific density of wood (rho_wood) in [kg/m³] must be inserted in Matlab. Also the saturation capillary moisture content (wcap) in kg/m³ must be inserted of the material at the outdoor side of the construction.

The Matlab-code for determining the Damage Indicators uses the function *intersections*. This only works when the code described in Appendix G is copied into a Matlab-file with the name **intersections.m**, This code is written by Douglas M. Schwarz. All his comments in the script are deleted to keep the code short. The script including the comments of Schwarz is on the website [http://blogs.mathworks.com](http://blogs.mathworks.com).

The results of the assessment of the damage indicators are plotted in Results_Damage_Indicators.txt. Example of this text file is:

```
Damage Indicators:

Percentage Condensation: 0.00
Time that mold growth criteria are met: 0.00
Percentage that mold growth criteria are met: 0.00
Time that wood decay criteria are met: 0.00
Total freeze-thaw-cycles: 0.00
Effective freeze-thaw-cycles: 0.00
Crystallization-dissolution-cycles 65 RH indoor: 0.00
Crystallization-dissolution-cycles 90 RH indoor: 0.00
Crystallization-dissolution-cycles 65 RH outdoor: 0.00
Crystallization-dissolution-cycles 90 RH outdoor: 0.00

Explanation:

Percentage of the time that condensation occurs.

Mold growth criteria: Indoor surface relative humidity > 80 percent.

Wood decay occurs when the surface relative humidity exceeds 95 percent and/or the mass moisture content exceeds 0.2 kg/kg.

Total freeze-thaw-cycles have the criteria: temperature below 0 degrees Celsius and is less important than effective cycles.

Effective freeze-thaw-cycles have the criteria: temperature below 0 degrees Celsius and capillary moisture content higher than 91 volume percent.

Each salt has a different critical RH-value. The two salts which causes the most severe physical damage are chosen. These have the critical RH-percentages of 65 and 90. The factor of salinity is not taken into account.

Thermal cracks are not taken into account.
```

Matlab-code for **Damage_Indicators.m**:
% Damage Indicators
% Results are plotted in Results_Damage_Indicators.txt
clear all; close all;

rho_wood=150; % Specific density of wood
wcap=75; % Capillary moisture content of external material

load Tsi.txt; TsiT=Tsi(:,2);
load Tse.txt; Tse_T=Tse(:,2);
load RHsi.txt; RHsi_v=RHsi(:,2);
load RHse.txt; RHse_v=RHse(:,2);
load RHs_wood.txt; RHsiw=RHs_wood(:,2);
load wwood.txt; w_wood=wwood(:,2);
load wse.txt; w_frost=wse(:,2);

load RHallmax.txt; RHall=RHallmax(:,2);
load RHallmax.txt; RHall_t=RHallmax(:,2);

%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% % Condensation
%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for t=1:length(RHall);
if RHall(t)>=99;
condensation(t)=1;
else
condensation(t)=0;
end
conden_T=(sum(condensation))./length(RHall)*100;
end

%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% % Mold Growth   RH exceeding 80%
%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for t=1:length(RHs_wood);
if TsiT(t)==0
if TsiT(t)<=50;
if RHs_wood(t)>=95;
wood_decay_rh(t)=1;
else
wood_decay_rh(t)=0;
end
else
wood_decay_rh(t)=0;
end
else
wood_decay_rh(t)=0;
end

if Tav(t)>=0
if Tav(t)<=50;
if mc_mp(t)>=0.20;
wood_decay_mc(t)=1;
else
wood_decay_mc(t)=0;
end
else
wood_decay_mc(t)=0;
end

if wood_decay_rh(t)==1;
if wood_decay_mc(t)==1;
wood_decay_double(t)=1;
end
```matlab
else
    wood_decay_double(t)=0;
end
else
    wood_decay_double(t)=0;
end
end

wood_decay_t=(sum(wood_decay_rh)+sum(wood_decay_mc))-
            sum(wood_decay_double);

%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Frost Damage
Zg=Tse_t.*0;
[xout,yout]=intersections(Tse_t,TseT,Tse_t,Zg,1);
Tcyc=((length(xout))/2);
freez_thaw_T=round(Tcyc);
% Total freeze-thaw-cycles
% Effective freeze-thaw-cycles
for t=1:1:length(w_frost);
    if (w_frost(t)/1000)<=(0.91*(wcap/1000));
        v_wcap(t)=1;
    else
        v_wcap(t)=0;
    end
Zkf=Tse_t.*0-(v_wcap(t).*1000);
[xoutT,youtT]=intersections(Tse_t,TseT,Tse_t,Zkf,1);
Tfluc=((length(xoutT))/2);
for t=1:1:length(w_frost);
    if TseT(t)<=0;
        v_T(t)=0;
    else
        v_T(t)=1000;
    end
Zw(t)=0.91.*(wcap./1000)-v_T(t);
[xoutW,youtW]=intersections(w_t,w_frost,w_t,Zw,1);
wfluc=((length(xoutW))/2);
freeze_thaw_cy=round((Tfluc+wfluc));

%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Salt Damage
Zk65i=65+RHall_t.*0;
[xout65i,yout65i]=intersections(RHsi_t,RHsi_v,RHsi_t,Zk65i,1);
salt_c_65i=round((length(xout65i))/2);
Zk90i=90+RHall_t.*0;
[xout90i,yout90i]=intersections(RHsi_t,RHsi_v,RHsi_t,Zk90i,1);
salt_c_90i=round((length(xout90i))/2);
Zk65e=65+RHall_t.*0;
[xout65e,yout65e]=intersections(RHse_t,RHse_v,RHse_t,Zk65e,1);
salt_c_65e=round((length(xout65e))/2);
Zk90e=90+RHall_t.*0;
[xout90e,yout90e]=intersections(RHse_t,RHse_v,RHse_t,Zk90e,1);
salt_c_90e=round((length(xout90e))/2);

%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% % Creating Text-file
fid = fopen('Results_Damage_Indicators.txt', 'w');
fprintf(fid, 'Damage Indicators:  
');
fprintf(fid, 'Percentage Condensation: %.2f \%', conden_T);
fprintf(fid, 'Time that mold growth criteria are met: %.2f \%', Mold_Time);
fprintf(fid, 'Time that wood decay criteria are met: %.2f \%', wood_decay_t);
fprintf(fid, 'Total freeze-thaw-cycles: %.2f \%', Tfluc);
fprintf(fid, 'Effective freeze-thaw-cycles: %.2f \%', Tfluc);
fprintf(fid, 'Crystallization-dissolution-cycles 65 RH indoor: %.2f \%', salt_c_65i);
fprintf(fid, 'Crystallization-dissolution-cycles 90 RH indoor: %.2f \%', salt_c_90i);
fprintf(fid, 'Crystallization-dissolution-cycles 65 RH outdoor: %.2f \%', salt_c_65e);
fprintf(fid, 'Crystallization-dissolution-cycles 90 RH outdoor: %.2f \%', salt_c_90e);
fprintf(fid, 'Explanation: \n');
fprintf(fid, 'Percentage of the time that condensation occurs. \n');
fprintf(fid, 'Mold growth criteria: Indoor surface relative humidity > 80 percent. \n');
fprintf(fid, 'Wood decay occurs when the surface relative humidity exceeds 95 \%');
fprintf(fid, 'percent and/or the mass moisture content exceeds 0.2 kg/kg. \n');
fprintf(fid, 'Total freeze-thaw-cycles have the criteria: temperature below \n');
fprintf(fid, '0 degrees Celsius and is less important than effective cycles. \n');
fprintf(fid, 'Effective freeze-thaw-cycles have the criteria: temperature \n');
fprintf(fid, 'below 0 degrees Celsius and capillary moisture content higher \n');
fprintf(fid, 'Than 91 volume percent. \n');
fprintf(fid, 'Each salt has a different critical RH-value. The two salts which \n');
```

5. Types of Errors

Possible errors are discussed in this section.

If COMSOL gives an error message which state that a singularity is reached, than this is mainly caused by a large sudden change or peak in the boundary conditions, such as temperature, vapor pressure or air pressure difference. Another cause can be variables related to boundary conditions, such as too high air permeability or too low surface coefficient of heat transfer. Also the singularity-error occurs when rain is simulated without the saturation effect.

Large errors occur when not using the points from the geometry for determining the results – and instead using the cut points. This problem does not occur when making line-plots.

Errors occur when the material properties are not precise enough. For example in [Goesten 2016], the moisture retention curve for coefficient BL,m in the form moisture content dependent on capillary pressure was initially not precise enough with the three known values. This problem was solved by using the moisture retention curve of the material properties from the Delphin-database which had around 100 values which was multiplied.

COMSOL has a default limit setting for the amount of rows (10,000 rows) in tables generated with Derived Values. When unexpected results occur during the making of graphs from tables, than this can be caused by the number of time steps exceeding this limit. This limit setting can be changed at every table separately.

It is possible that COMSOL delivers an error message which states that there is not enough disk space available. To prevent this, go to Study > Solver Configuration > Solver > Time-Dependent Solver and then clicking on the section Output, where you can set off everything, except store solution on disk.

In the case of an error message including the text: Undefined value found in the stiffness matrix. (...) giving NaN/Inf in the matrix rows for the variable comp1.LPc than it can be caused by two factors. First it can be caused if the Matlab-files are still open with the coefficients which are simulated in COMSOL. The second reason can happen when Delphin-materials are used, by which this error can be solved by deleting the ‘extrap’ part in the interpolation code of Matlab (interp1) in all material dependent coefficients.

6. References


7. Appendix

(A) Plots for Crefun.m

This code generates 2D-plots of the coefficients and can be added beneath the code described for Matlab script Crefun.m. COMSOL can generate 3D-plots of the coefficients, which are clearer than these 2D-plots.

```matlab
%% plot (optional)
figure(1)
subplot(321)
surf(xx,yy,BTa(xx,yy))
view(2); shading interp; colorbar
title('BTa')
ylabel(['T oC']); axis([0 10 -10 40])

subplot(323)
surf(xx,yy,D11a(xx,yy))
view(2); shading interp; colorbar
title('D11a')
ylabel(['T oC']); axis([0 10 -10 40])

subplot(325)
surf(xx,yy,D12a(xx,yy))
view(2); shading interp; colorbar
title('D12a')
ylabel(['T oC']); xlabel('LPc'); axis([0 10 -10 40])

subplot(322)
surf(xx,yy,BLa(xx,yy))
view(2); shading interp; colorbar
title('BLa')
axis([0 10 -10 40])

subplot(324)
surf(xx,yy,D21a(xx,yy))
view(2); shading interp; colorbar
title('D21a')
axis([0 10 -10 40])

subplot(326)
surf(xx,yy,D22a(xx,yy))
view(2); shading interp; colorbar
title('D22a')
xlabel('LPc'); axis([0 10 -10 40])

figure(2)
subplot(221)
surf(xx,yy,C11(xx,yy))
view(2); shading interp; colorbar
title('C11')
ylabel(['T oC']); axis([0 10 -10 40])

subplot(223)
surf(xx,yy,C12(xx,yy))
view(2); shading interp; colorbar
title('C12')
ylabel(['T oC']); xlabel('LPc'); axis([0 10 -10 40])

subplot(222)
surf(xx,yy,C21(xx,yy))
view(2); shading interp; colorbar
title('C21')
axis([0 10 -10 40])

subplot(224)
surf(xx,yy,C22(xx,yy))
view(2); shading interp; colorbar
title('C22')
xlabel('LPc'); axis([0 10 -10 40])
```
Using self-made tables

The coefficients required to insert material dependent properties are discussed as formulas (and constant values) and the use of Delphin-material properties. In this Appendix is shown how to use self-made tables.

Moisture content related to relative humidity – when the values are derived from measurements at 20 degrees Celsius (293.15 Kelvin) – has the Matlab-code:

\[
\phi_2 = \exp\left(-\frac{P_c}{\rho_w \cdot R_v \cdot 293.15}\right); \\
\text{RH}_2 = [0, 0.33, 0.75, 0.93, 1]; \\
w_2 = [4.8, 4.8, 6.4, 13, 13]; \\
w = \text{interp1}([\text{RH}_2, w_2], \phi_2, 'linear', 'extrap');
\]

For the use of a self-made table for the moisture retention curve in the coefficient BL it is required to use the moisture content related to capillary pressure. The relative humidity [-] can be changed into the capillary pressure with the equation: \( p_c = \rho_w \cdot R_v \cdot T \cdot \ln(\phi) = 1000 \times 462 \times 293.15 \times \ln(\phi) \). The result of this calculation is always negative, but the minus must be disregarded. Thereafter, the capillary pressure values and the corresponding moisture content values must be sorted from the lowest \( p_c \) to the highest \( p_c \). This lead for the above table to:

\[
P_{c\_\text{pos}} = [0, 9.8286e6, 3.8962e7, 1.5015e8, 1e9]; \\
w_{P_c} = [13, 13, 6.4, 4.8, 4.8]; \\
w = \text{interp1}([P_{c\_\text{pos}}, w_{P_c}], P_c, 'linear', 'extrap');
\]

An example of a table for the vapor diffusion factor:

\[
\phi_2 = \exp\left(-\frac{P_c}{\rho_w \cdot R_v \cdot 293.15}\right); \\
\text{RH}_2 = [0, 0.33, 0.75, 0.93, 1]; \\
m_2 = [12, 12, 14, 14, 14]; \\
mu = \text{interp1}([\text{RH}_2, m_2, \phi_2], 'linear', 'extrap'); \\
delta_p = \delta_{\text{aa}} / \mu;
\]

Wind-Driven Rain

The formula for estimating the Wind-Driven Rain (WDR) – when the vertical rain, wind direction and wind speed are known – on a vertical plane is:

\[
R_{wdr} = 0.222 \cdot U \cdot R_h^{0.88} \approx 0.222 \cdot U \cdot R_h
\]

\( R_{wdr} \) = wind-driven rain intensity [l/(m²·h)]
\( U \) = upstream horizontal wind velocity component at 10 meter height [m/s]
\( R_h \) = horizontal rainfall intensity [l/(m²·h)]

The value 0.222 (s/m) is an average value of the WDR coefficient which was empirically determined. The exponent 0.88 can be omitted from the formula, according to [Blocken et al. 2004]. Therefore, this exponent is omitted. This formula is called by [Blocken et al. 2004] a rough estimate.

The Matlab-script creates the file \( \text{Horizontal\_Wind-Driven\_Rain.txt} \).

Reference:
The Matlab-script (WDR.m) is:

```matlab
% Wind Driven Rain (WDR)
% Results are plotted into the file Horizontal_Wind-Driven_Rain.txt
% Vertical rain [l/m2h] changed to horizontal rain in kg/m2s

% Angles corresponding to wind directions
% North wind (north -> south) = 0°
% East wind (east -> west) = 90°
% South wind (south -> north) = 180°
% West wind (west -> east) = 270°

close all
clear all

%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Inserting information
load Vertical_Rain.txt
% l/m2h = mm/h
load Wind_Direction.txt
% degrees
load Wind_Speed.txt
% m/s

% Angle of the normal of the wall related to the North
gamma=330;
% degrees
% Wind angle that leads to WDR on the wall
Angle_rain_on_wall=80;
% Assumption !!!!!!!!!!!!!!

%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Standard code
time=Vertical_Rain(:,1); VR=Vertical_Rain(:,2);
WD=Wind_Direction(:,2); WS=Wind_Speed(:,2);
gammax=gamma+360; W Dx=W+360;
WDcrit_min=gammax-Angle_rain_on_wall;
WDcrit_max=gammax+Angle_rain_on_wall;

for i=1:length(WD);
    if WDx(i)>=WDcrit_min;
        if WDx(i)<=WDcrit_max;
            Status(i)=1;
        else
            Status(i)=0;
        end
    else
        Status(i)=0;
    end

    WDRv(i)=Status(i).*VR(i).*WS(i).*0.222; % l/m2h = mm/h
    WDRm(i)=WDRv(i)./3600; % kg/m2s
end

Horizontal_Rain(:,1)=time; Horizontal_Rain(:,2)=WDRm;
A = [time'; WDRm];
fid = fopen('Horizontal_Wind-Driven_Rain.txt','w');
fprintf(fid,'%6.2f %12.8f\n',A);
fclose(fid);
```

(D) Sophisticated Convection Method

It is advised to simulate a construction twice if it contains an air cavity – *i.e.* one time with the simple convection method and one time with the sophisticated convection method. The simple convection method is described in the main text. The sophisticated convection method is described here. The formula of the air flux through a material is:

\[
g_a = -k_a \cdot (\nabla p + \rho_a \cdot g)
\]

\(g_a\) = Air flux through the material [kg/(m²·s)]
\(k_a\) = Air permeability [kg/(s·m·Pa)] \~ [s]
\(p\) = Air pressure [Pa]
\(\rho_a\) = Specific density of air [kg/m³]

The sophisticated method uses this formula:
\[ \vec{g}_a = -k_a \cdot \nabla p \cdot \rho_a \]

The simple convection method uses the air pressure difference over the whole construction \((dPa)\) at each location of the simulated geometry. The sophisticated convection method uses the local air pressure \((Pair)\) at each location.

The input of the **Convection Coefficient** in the heat and moisture part must be changed. The \(dPa(t)\) must be replaced with \(Pair\).

The boundary conditions which are related to infiltration of air differ also from the simple convection method. Instead of \(dPa(t)\), the sophisticated convection method uses \(Pair\). The boundary conditions have the following formulas:

**External boundary condition when the air leaves the material at the external side:**

\[
he^*(T_e(t)-T)+Beta_e*L*(Pe(t)-Pv(LPc,T))+ka*1.2*Pair*0.62e^{-5}*(Pv(LPc,T)-Pe(t))
\]

**For the internal surface when the air enters the material at this side:**

\[
h_i*(T_i(t)-T)+Beta_i*L*(Pi(t)-Pv(LPc,T))+ka*1.2*Pair*0.62e^{-5}*(Pi(t)-Pv(LPc,T))
\]

The above formulas about the boundary conditions are without solar irradiation and rain flux, but these factors can also be included.

**Adding Physics Interface for air flux**

An extra physics interface must be implemented. Click on **Add Physics** and choosing **Coefficient Form PDE**. Click on **Coefficient Form PDE** and define the **Dependent Variables** as \(Pair\), which is the local air pressure.

Left click on **Coefficient Form PDE** and insert the air permeability in the location of the **diffusion coefficient** \((c)\). The air permeability is not a kind of diffusion, but it must be inserted in this location, because the air permeability is a scalar and not a vector. The **damping or mass coefficient** \((d_a)\) must be 1. All the other values must be zero.

The boundary conditions must be inserted as **Flux/Source** based on the following formula:

\[
g = \beta_{air} \cdot (P_{air \ surface} - P_{ambient \ air})
\]

\(g\) = Air flux at surface \([kg/m^2s]\)

\(\beta_{air}\) = Air convection exchange coefficient \([s/m]\) = \(0.1 \text{ s/m}\) according to Delphin

\(P_{ambient \ air}\) = Air pressure of the ambient air \([Pa]\)

\(P_{air \ surface}\) = Air pressure inside the material at the surface \([Pa]\)

[Goesten 2016] only tested the sophisticated convection method when the air pressure difference between inside and outside was known.
This is the boundary conditions at the side were the air leaves the material when the air pressure difference is positive. The other boundary condition must be zero.

According to the material properties from Delphin, the air permeability of an air cavity is approximate 0.099 kg/(s·m·Pa) or 0.1 kg/(s·m·Pa).

(E) %RMSE and maximal error

The Matlab-script for determining the maximum deviation and the percentage Root Mean Square Error are given in this section. The formula of %RMSE is:

\[
%RMSE = \sqrt{\frac{\sum (V_{meas} - V_{sim})^2}{n}} \cdot \frac{100 \cdot n}{\sum V_{meas}}
\]

%RMSE = Percentage Root Mean Square Error

\(V_{meas}\) = Values of the measurement

\(V_{sim}\) = Values determined with the simulations

n = Amount of values

The two error-indicators are shown in temperature and the relative humidity. This example is chosen for the external surface, but the same codes can be used for other locations. The results are shown in the Command Window of Matlab. Only the loading of the text files have to be changed.

```matlab
% %RMSE
clear all
%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% %RMSE of Temperature
% Value_measurement_T
load Tsem.txt
VM_T=Tsem(:,2);
% Value_simulation_T
load Tse.txt
VS_T=Tse(:,2);

% Amount of values
n_T=length(VS_T);
% %RMSE
RMSE_Tso=sqrt((sum((VM_T-VS_T).^2))/n_T)*((100*n_T)/sum(VM_T))
% Maximum error
Max_Tso=max(abs(VM_T-VS_T))
clear VM_T VS_T n_T

%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% %RMSE of Relative Humidity
% Value_measurement_RH
load RHSEM.txt
VM_RH=RHSEM(:,2);
% Value_simulation_RH
load RHSE.txt
VS_RH=RHSE(:,2);

% Amount of values
n_RH=length(VS_RH);
% %RMSE
RMSE_RHe=sqrt((sum((VM_RH-VS_RH).^2))/n_RH)*((100*n_RH)/sum(VM_RH))
% Maximum error
Max_RHe=max(abs(VM_RH-VS_RH))
clear VM_RH VS_RH n_RH
```
(F) Mollier Diagram

The following Matlab-script generates a Mollier-diagram. Only the temperature [°C] and relative humidity [%] must be loaded into Matlab. If the vapor pressure must be inserted, than in the standard code, the variable pg must not be the formula but directly the vapor pressure data. The RH-line of 80% is made red, because that is the RH-threshold of mold growth.

```matlab
% Mollier Diagram
clear all
close all

% Inserting Temperature in Celsius
load Tsi.txt
Tg=Tsi(:,2);

% Inserting Relative Humidity in percentage
load RHsi.txt
RHg=RHsi(:,2);
%% $\text{Standard code}$
% Temperature range in Celsius
Tc=[-15:1:30];
% Saturation pressure
Psat=exp(65.8094-7066.27./(Tc+273.15)-5.976*log(Tc+273.15));
% Information for plotting the points
PsatG=exp(65.8094-7066.27./(Tg+273.15)-5.976*log(Tg+273.15));
RHg(RHg>100)=100; % It is impossible to have a RH higher than 100%
pg=(RHg./100).*PsatG; % Vapor pressure [Pa]

% Plotting
figure(1)
plot(Psat,Tc); hold on
plot(Psat*0.9,Tc); hold on
plot(Psat*0.8,Tc,'r') % Threshold RH-value for mold growth in red
hold on
plot(Psat*0.7,Tc); hold on
plot(Psat*0.6,Tc); hold on
plot(Psat*0.5,Tc); hold on
plot(Psat*0.4,Tc); hold on
plot(Psat*0.3,Tc); hold on
plot(Psat*0.2,Tc); hold on
plot(Psat*0.1,Tc); hold on
plot(pg,Tg,'r.') % Plotting the points in red

% Creating text
title('Mollier Diagram')
set(gca,'xlim',[0 4500],'ylim',[-15 30])
ylabel('Temperature [°C]')
xlabel('Vapor Pressure [Pa]')
grid on
```

Figure E1: Mollier Diagram. The vapor pressure [Pa] is depicted on the x-axes and the temperature (°C) on the y-axes. The chance of mold growth is high, when a data point depicts a larger relative humidity than 80%.

(G) intersections.m

This code must be copied into a Matlab-file with the name intersections.m. This code is written by Douglas M. Schwartz. To keep the code short all his comments are deleted.

```matlab
function [x0,y0,iout,jout] = intersections(x1,y1,x2,y2,robust)
% INTERSECTIONS
% Author: Douglas M. Schwarz Version: 1.12, 27 January 2010
error(nargchk(2,5,nargin))
switch nargin
    case 2
        robust = true; x2 = x1; y2 = y1;
        self_intersect = true;
    case 3
        robust = x2; x2 = x1; y2 = y1;
        self_intersect = true;
    case 4
        robust = true; self_intersect = false;
    case 5
        self_intersect = false;
end
if sum(size(x1) > 1) ~= 1 || sum(size(y1) > 1) ~= 1 || ...
    length(x1) ~= length(y1)
    error('X1 and Y1 must be equal-length vectors of at least 2 points.')
end
if sum(size(x2) > 1) ~= 1 || sum(size(y2) > 1) ~= 1 || ...
    length(x2) ~= length(y2)
    error('X2 and Y2 must be equal-length vectors of at least 2 points.')
end
x1 = x1(:); y1 = y1(:); x2 = x2(:); y2 = y2(:);
 n1 = length(x1) - 1; n2 = length(x2) - 1;
 xyl = [x1 y1]; xy2 = [x2 y2];
 dxy1 = diff(xyl); dxy2 = diff(xy2);
[1,j] = find(repmat(min(x1(1:end-1),x1(2:end)),1,n2) <= ...
            repmat(max(x1(1:end-1),x1(2:end))',n1,1) & ...
            repmat(max(x1(1:end-1),x1(2:end))',1,n2));
```
repmat(min(x2(1:end-1),x2(2:end)).',n1,1) & ... 
repmat(min(y1(1:end-1),y1(2:end))).',1,n2) <= ...
repmat(max(y2(1:end-1),y2(2:end))).',n1,1) & ... 
repmat(max(y1(1:end-1),y1(2:end))).',1,n2) >= ...
repmat(min(y2(1:end-1),y2(2:end)).',n1,1));
i = reshape(i,[],1); j = reshape(j,[],1);
if self_intersect
    remove = isnan(sum(dxy1(i,:),+ dxy2(j,:),2)) | j <= i + 1;
else
    remove = isnan(sum(dxy1(i,:),+ dxy2(j,:),2));
end
i(remove) = [];
if remove
    j(remove) = [];
end
n = length(i);
T = zeros(4,n);
AA = zeros(4,4,n);
AA([1 2],3,:) = -1; AA([3 4],4,:) = -1;
AA([1 3],1,:) = dxy1(i,:); AA([2 4],2,:) = dxy2(j,:);
B = -[x1(i) x2(j) y1(i) y2(j)];
if robust
    warning_state = warning('off','MATLAB:singularMatrix');
    try
        lastwarn('')
        for k = 1:n
            T(:,k) = AA(:,:,k)
            lastwarn('')
            if strcmp(last_warn,'MATLAB:singularMatrix')
                T(1,k) = NaN;
                overlap(k) = rcond([dxy1(i(k),:);xy2(j(k),:) - xy1(i(k),:)]) < eps;
            end
        end
        warning(warning_state)
    catch
        err
        warning(warning_state)
        rethrow(err)
    end
    in_range = (T(1,:) >= 0 & T(2,:) >= 0 & T(1,:) <= 1 & T(2,:) <= 1).
    if any(overlap)
        ia = i(overlap);
        ja = j(overlap);
        T(3,overlap) = (max(min(x1(ia),x1(ia+1)),min(x2(ja),x2(ja+1)))) + ... 
        min(max(x1(ia),x1(ia+1)),max(x2(ja),x2(ja+1))))/2;
        T(4,overlap) = (max(min(y1(ia),y1(ia+1)),min(y2(ja),y2(ja+1)))) + ... 
        min(max(y1(ia),y1(ia+1)),max(y2(ja),y2(ja+1))))/2;
    end
    selected = in_range | overlap;
else
    selected = in_range;
end
xy0 = T(3:4,selected).
[x0,index] = unique(xy0,'rows');
x0 = x0(:,1); y0 = x0(:,2);
if nargout > 2
    sel_index = find(selected);
    sel = sel_index(index);
    iout = i(sel) + T(1,sel).
    jout = j(sel) + T(2,sel).
else
    for k = 1:n
        [L,U] = lu(AA(:, :, k));
        T(:, k) = U(L)
    end
    in_range = (T(1,:) >= 0 & T(2,:) >= 0 & T(1,:) < 1 & T(2,:) < 1).
    x0 = T(3,in_range).
    y0 = T(4,in_range).
    if nargout > 2
        iout = i(in_range) + T(1,in_range).
        jout = j(in_range) + T(2,in_range).
    end
end