



# Report

On

## Implementing custom boundary conditions using 'codedFixedValue' boundary condition in OpenFOAM

Submitted by

**Mohamed Adnan K N**

*Working Professional*

Mentored by

**Mr. John Pinto**

*Department of Mechanical Engineering, Indian Institute of Technology, Bombay*

Under the guidance of

**Dr. Harikrishnan S**

*Division of Mechanical Engineering, School of Engineering, Cochin University of  
Science and Technology, Kochi, Kerala, India*

## 1. Introduction

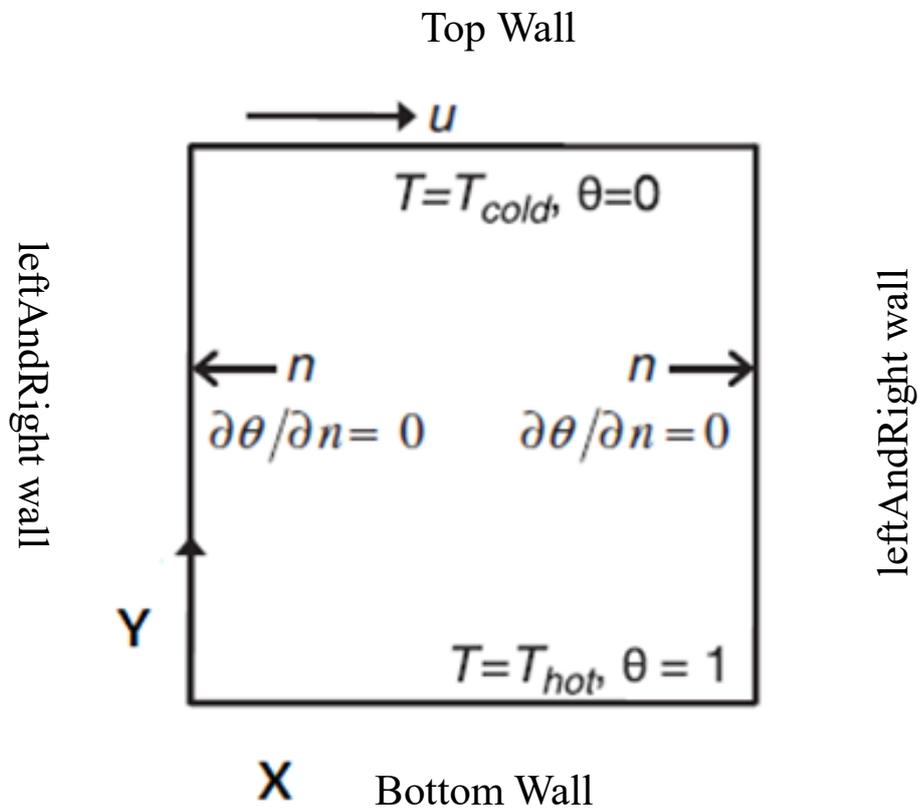
In Computational Fluid Dynamics, boundary conditions play a crucial role in defining the behavior of physical systems. Among these, the fixed value boundary condition is widely used for its simplicity and effectiveness in various applications. This report focuses on the ‘CodedFixedValue Boundary Condition’, a method that integrates coding techniques to enhance the implementation and versatility of fixed value boundary conditions. With this utility, various complex boundary conditions can be implemented in OpenFOAM without using external libraries (which are not available with standard OpenFOAM distribution) or high-level programming (modifying existing OpenFOAM libraries). The ‘CodedFixedValue’ boundary condition is the most user-friendly and accessible method available in the OpenFOAM library. It allows the user to embed a piece of C++ code directly within the boundary condition files for implementing complex custom boundary conditions that are required for simulations. This report demonstrates various applications of ‘CodedFixedValue’ boundary conditions with examples. Section 2 introduces a specific CFD problem and outlines the necessary case setup for demonstrating the application of the "CodedFixedValue" boundary condition. Section 3 outlines the numerical method used in this study, including the governing equations and boundary conditions. Section 4 provides a detailed demonstration of how the 'CodedFixedValue' utility is applied to implement various boundary conditions. Section 5 concludes the report with a summary of findings and potential future developments.

## 2. Problem Statement

To illustrate the applications of the 'CodedFixedValue' boundary condition, a two-dimensional lid-driven cavity with mixed convection was chosen as a representative problem. The simulation conditions outlined in [1] and [2] were adopted for this study. Figure 1 presents the schematic diagram of the computational domain. The bottom, leftAndRight walls were assumed to have no-slip conditions, while a velocity boundary condition was applied to the top wall. Regarding thermal boundary conditions, the bottom wall was considered hot, the top wall cold, and the remaining walls are considered as adiabatic. Table 1 summarizes the geometric and flow details

used in this study. The 'CodedFixedValue' utility was employed to implement various boundary conditions as follows:

- Time-varying boundary conditions :- imposing time dependent boundary condition at top wall.
- Space varying boundary conditions:- Imposing a spatially varying velocity profile at top wall.
- Boundary conditions with conditional statements:- Varying boundary conditions based on user-defined conditional statement given by user.
- Output based input boundary conditions:- Boundary conditions that adjusts/vary based on the results obtained from previous time step solutions.



**Figure 1:** Schematic view of Case Setup

**Table 1:** Details of geometry and flow conditions

<b>Geometry</b>	Length of the cavity(x) = 1m Height of the cavity (y) = 1m Depth of the cavity (z) = 1m
<b>Mesh size</b>	81× 81 × 1 cells
<b>Reynolds number</b>	Re = 100
<b>Thermal expansion coefficient</b>	$\beta_{T,air} = 0.00317 K^{-1}$
<b>Reference Temperature</b>	$T_0 = 315K$
<b>Non-dimensional temperature</b>	$\theta = \frac{(T - T_{cold})}{(T_{hot} - T_{cold})}$
<b>Reference length scale</b>	L = 1m
<b>Prandtl number</b>	Pr = 0.71
<b>Grashof number</b>	$Gr_T = \frac{g\beta(T_{hot} - T_{cold})L^3}{\nu^2}$ $Gr_T = 100$

### 3. Numerical Methods

In the report, the **buoyantPimpleFoam** (Boussinesq’s approximation) solver from the OpenFOAM was used to model the buoyancy-driven flow. The solver discretizes the governing equations using the finite volume method (FVM), with implicit time-stepping to ensure stability. “Gauss linear” scheme for gradients and “Gauss upwind” for convective terms is used to maintain numerical stability. For the diffusion terms, the “Gauss linear corrected” scheme was used to enhance accuracy. The pressure-velocity coupling was handled by the PIMPLE algorithm, combining the PISO and SIMPLE algorithms to correct the pressure field and ensure mass conservation iteratively. The flow was considered to be laminar and hence turbulence fields were ignored. Convergence criteria considered for this case were 1e-6 to ensure accurate and stable solutions. Probe function utility has been used to visualize field variations over a period of time. By default, OpenFOAM employs linear interpolation

to estimate field values at probe locations based on the values at neighbouring cell centers. Linear interpolation scheme produces inaccurate results at boundaries where field values change rapidly over time. To mitigate this issue, the **cellPoint** interpolation scheme is used to obtain field values at probe locations. This scheme identifies the cell center that is closest to the probe location. The field value at the probe location is taken directly from this cell center instead of interpolation thus yielding accurate results.

### 3.1 Governing Equations

#### 3.1.1 Mass conservation equation

The continuity equation is expressed as:  $\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho U) = 0$  (1)

#### 3.1.2 Momentum Conservation Equation

The solver is designed for incompressible flow, assuming that the density ( $\rho$ ) remains constant throughout the flow field, except when considering buoyancy effects in the source term and expressed as follows:

$$\frac{\partial U}{\partial t} + \nabla \cdot (UU) = -\frac{1}{\rho_0} (\nabla p - \rho g) + \nabla \cdot [v_{eff} (\nabla U + (\nabla U)^T)] \quad (2)$$

where,  $\rho = \rho_0 [1 - \beta(T - T_0)]$ ,  $\rho_0, T_0$  is reference density and temperature respectively. In terms of implementation in OpenFOAM, the pressure gradient and gravity force terms are rearranged in the following form:

$$-\nabla p + \rho g = -\nabla p_{rgh} - (g \cdot r) \nabla \rho \quad (3)$$

#### 3.1.3 Energy equation

The energy conservation equation for internal energy ( $e$ ) is expressed as:

$$\frac{\partial(\rho e)}{\partial t} + \nabla \cdot (\rho U e) + \frac{\partial(\rho K)}{\partial t} + \nabla \cdot (\rho U K) + \nabla \cdot (pU) = \nabla \cdot (\alpha_{eff} \nabla e) + \rho u \cdot g \quad (4)$$

where,  $\alpha_{eff} = \frac{\rho v_t}{Pr_t} + \frac{\mu}{Pr}$ .

### 3.2 Boundary conditions

In our simulations, we applied a codedFixedValue boundary condition to the top face of the cavity to specify a custom boundary condition. In general, patch names and their corresponding boundary condition type were presented in Tables 2 and 3. Other boundary fields are set to be zeroGradient.

**Table 2:** Boundary conditions for “U” field

Patch name	Boundary condition
top	codedFixedValue
bottom	noSlip
leftAndright	noSlip
frontAndBack	empty

**Table 3:** Boundary conditions for “T” field

Patch name	Boundary condition
top	300 K
bottom	330 K
leftAndright	zeroGradient
frontAndBack	empty

## 4. Demonstrations

To illustrate the possibilities offered using codedFixedValue in OpenFOAM, let's explore several case studies that demonstrate its application in various scenarios. Each example showcases how ‘codedFixedValue’ can be used to define dynamic and complex boundary conditions efficiently.

### 4.1 Time-varying boundary condition

In this example, we will impose a time-varying velocity profile on the top wall given by equation  $u = u_0 \sin(\omega\tau)$ , where,  $\tau$  is defined as the non-dimensional time ( $\tau = \frac{u_0 t}{H}$ ) and  $\omega$  is the oscillation frequency which is taken as 1. We apply a time-varying boundary condition on the top patch of the cavity. The velocity oscillates sinusoidally in the time with a frequency defined by omega, amplitude defined by A, and scaled by non-dimensional time  $\tau$ . Time-varying velocity boundary conditions can be

implemented in OpenFOAM by simply copying and pasting the code snippet shown in Figure 2 into the 0/U file.

```

top
{
    type            codedFixedValue;
    value           uniform (0 0 0);           // Initial value
    name            codedPathBC;              // Custom name for the boundary condition

    code
    #{
        const scalar omega = 1;                // oscillation frequency
        const scalar A = 9.66;                 // maximum oscillating speed
        const scalar time = this->db().time().value(); // Current simulation time
        const scalar H = 1;                    // Height or characteristic length
        const scalar tau = (A*time)/H;         // Non-dimensional time
        vector U(A * sin(omega * tau),0,0);    // Velocity vector with sinusoidal variation

        // Apply the calculated velocity to each face of the boundary patch
        forAll(this->patch().Cf(), faceI)
        {
            operator[](faceI) = U;
        }
    };
}

```

**Figure 2:** Code snippet of time-varying velocity profile using ‘codedFixedValue’ boundary condition

Probe function utility has been used to visualize the imposed boundary condition during the simulation. **Figure 3** indicates the code snippet corresponding to the probe function from the ‘controlDict’ file and the variation of the x-component of velocity with time for one of the locations at the top wall.

```

(a) probes
{
    // Where to load it from
    functionObjectLibs ( "libsampling.so" );

    type            probes;

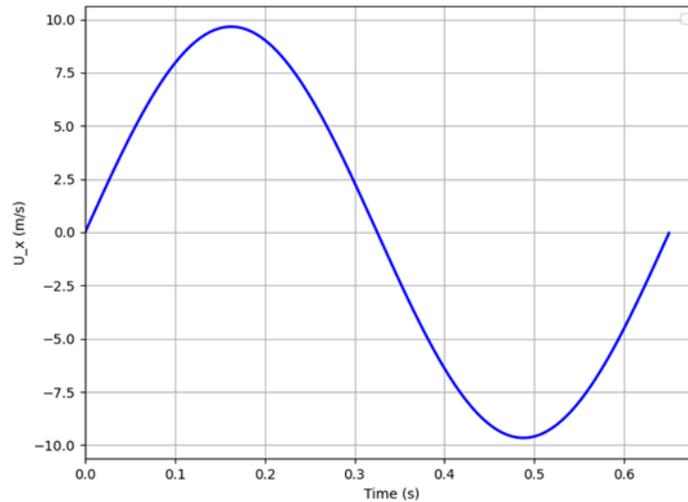
    // Name of the directory for probe data
    name            probes;

    // Fields to be probed
    fields (
        U
    );

    probeLocations
    (
        ( 0.5 1 1)
    );

    interpolationScheme cellPoint;
}

```



(b)

**Figure 3:** (a) Probe punction code snippet and, (b) imposed time varying boundary condition at top wall ( $U_x$  vs Time at (0.5,1,1))

## 4.2 Space varying boundary condition

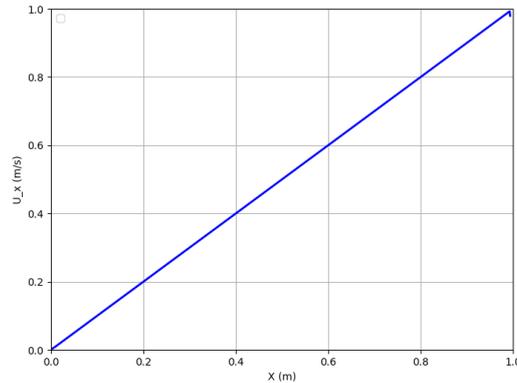
In this example, we will impose a spatially varying velocity profile on the top wall. The velocity will vary linearly from 0 to 1 across all the faces on the top wall based on the x location of the face center. Space-varying velocity boundary conditions can be implemented in OpenFOAM by simply copying and pasting the code snippet shown in Fig. 4 into the 0/U file.

```
top
{
    type            codedFixedValue;
    value           uniform (0 0 0);           // Initial value
    name            codedPathBC;              // Custom name for the boundary condition

    code
    #{
        const scalar minX = 0.0; // Minimum x-coordinate of the patch
        const scalar maxX = 1.0; // Maximum x-coordinate of the patch
        // Looping through each face of the boundary patch
        forAll(this->patch().Cf(), faceI)
        {
            // x-coordinate of the face center
            const scalar x = this->patch().Cf()[faceI].x();
            scalar velocity_linear = (x - minX) / (maxX - minX); // Linear variation
            vector U(velocity_linear , 0, 0); // Velocity vector
            // Assign the velocity to the boundary face
            operator[](faceI) = U;
        }
    #};
}
```

**Figure 4:** Code snippet of space varying velocity profile using ‘codedFixedValue’ boundary condition

Plot over line function in ParaView to visualize the variation of simulation results along a specific line in the domain. In this case, velocity varies across the top faces of the cavity. Specifically, we plotted the results from the points (0, 1, 1) to (1, 1, 1). **Figure 5** illustrates the variation of the x-component of velocity along the specified line.



**Figure 5:** Imposed space varying boundary condition at top wall ( $U_x$  vs  $X$ ) from point (0,1,1) to (1,1,1)

### 4.3 Boundary conditions with conditional statements

#### 4.3.1 Time dependent boundary conditions with conditional statements

In this example, we will impose time-dependent velocity on the top wall. Specifically, we will:

- Apply a constant velocity  $u(9.66,0,0)$  for the first 4 seconds of the simulation.
- Switch to sinusoidal velocity after the first 4 seconds given by equation  $u = u_0 \sin(\omega\tau)$ , where,  $\tau$  is defined as the non-dimensional time ( $(\tau) = \frac{u_0 t}{H}$ ) and  $\omega$  is the oscillation frequency which is taken as 1.

Time-dependent velocity boundary conditions can be implemented in OpenFOAM by simply copying and pasting the code snippet shown in Fig. 6 into the 0/U file.

```

top
{
  type          codedFixedValue;
  value         uniform (0 0 0);          // Initial value
  name         codedPathBC;              // Custom name for the boundary condition

  code
  #{
    const scalar time = this->db().time().value(); // Current simulation time
    const scalar omega = 1; // Frequency for sinusoidal velocity
    const scalar A = 9.66; // Amplitude for sinusoidal velocity
    vector U; // Velocity vector to be applied

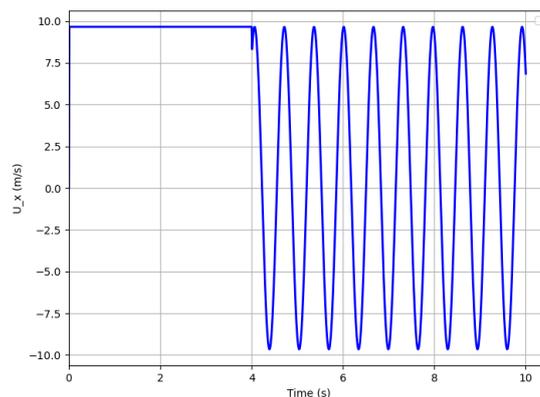
    if (time <= 4.0)
    {
      U = vector(9.66, 0, 0); // constant velocity
    }
    else
    {
      const scalar H = 1; // Characteristic length
      const scalar tau = (A * (time)) / H; // Non-dimensional time
      U = vector(A * sin(omega * tau), 0, 0); // sinusoidal velocity
    }

    // Applying the calculated velocity to each face of the boundary patch
    forAll(this->patch().Cf(), faceI)
    {
      operator[](faceI) = U;
    }
  };
}

```

**Figure 6:** Code snippet of implementing time dependent velocity using ‘codedFixedValue’ boundary condition

Probe function utility has been used to visualize the imposed boundary condition during the simulation similar to case 4.1. **Figure 7** illustrates the variation of the x-component of velocity with time for one of the locations at the top wall.



**Figure 7:** Imposed time dependent boundary condition at top wall (U<sub>x</sub> vs Time at (0.5,1,1))

### 4.3.2 Impulse condition

In this example, we will impose a sudden change or impulse in the velocity over a short period. Specifically, we will

- Apply zero velocity at  $t = 0$
- Switch to constant velocity  $u(9.66,0,0)$  between  $t = 4s$  and  $t = 5s$
- Switch back to zero velocity after  $t = 5s$

Velocity impulse boundary conditions can be implemented in OpenFOAM by simply copying and pasting the code snippet shown in Fig. 8 into a 0/U file.

```

top
{
    type            codedFixedValue;
    value           uniform (0 0 0);           // Initial value
    name            codedPathBC;              // Custom name for the boundary condition

    code

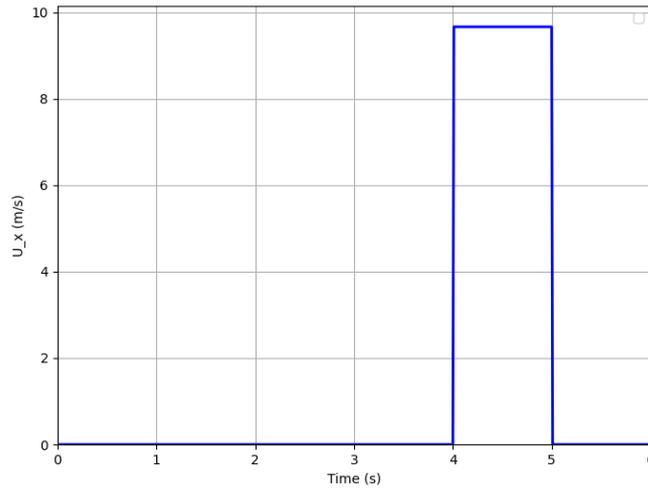
    #{
        const scalar time = this->db().time().value(); // Current simulation time
        vector U;                                     // Velocity vector to be applied

        if (time == 0)
        {
            U = vector(0, 0, 0);
        }
        else if (time > 4.0 && time < 5.0)
        {
            U = vector(9.66, 0, 0); // Constant velocity
        }
        else
        {
            U = vector(0, 0, 0); // Zero velocity
        }
        // Apply the calculated velocity to each face of the boundary patch
        forAll(this->patch().Cf(), faceI)
        {
            operator[](faceI) = U;
        }
    };
}

```

**Figure 8:** Code snippet of implementing velocity impulse using ‘codedFixedValue’ boundary condition

Probe function utility has been used to visualize the imposed boundary condition during the simulation similar to case 4.1. **Figure 9** illustrates the variation of the x-component of velocity with time for one of the locations at the top wall.



**Figure 9:** Imposed velocity impulse boundary condition at top wall  
(U<sub>x</sub> vs Time at (0.5,1,1))

### 4.3.3 Output-based input

A simulation variable-dependent boundary condition is one where the boundary value varies based on a specific simulation variable. In this case, we will change the velocity boundary condition based on the volume averaged domain temperature values. Specifically, we will apply the following velocity boundary condition at the top wall;

- Apply constant velocity  $U(2,0,0)$  if the volume averaged domain temperature is  $\geq 315$  K
- Apply constant velocity  $U(9.66,0,0)$  if the volume averaged domain temperature is  $< 315$  K

The following code as shown in Fig. 10 (a) should be there in the 0/U file to properly implement this case. To verify our results, we need to calculate the volume averaged temperature field in the domain. For storing the temperature field, we need to make some modifications in the system/controlDict file. Figure 10 (b) shows the code snippet that needs to be added for calculating the volAverage temp field.

```

boundaryField
{
    top
    {
        type            codedFixedValue;
        value            uniform (0 0 0);          // Initial value
        name            codedPathBC;             // Custom name for the boundary condition

        code
        #{
            // accessing temperature variable
            const volScalarField& T = this->db().lookupObject<volScalarField>("T");
            const scalar TAVg = gAverage(T);      // Volume-averaged temperature
            vectorField U(this->patch().size());

            if (TAVg < 315)
            {
                U = vector(9.66, 0, 0);
            }
            else
            {
                U = vector(2, 0, 0);
            }

            operator==(U);
        #};
    }
}

```

(a)

```

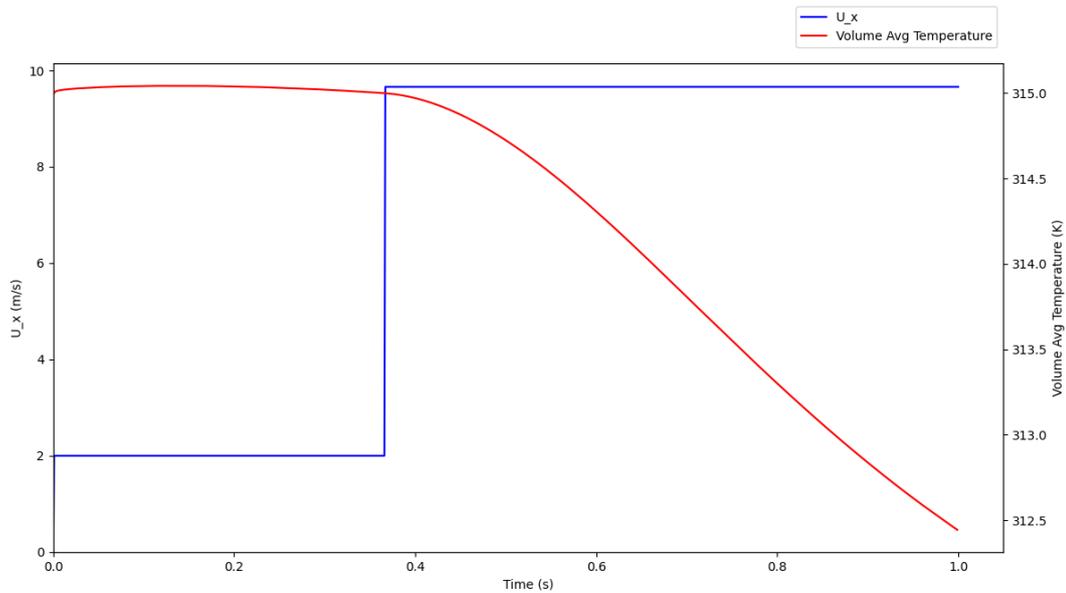
volFieldValue1
{
    type            volFieldValue;
    functionObjectLibs ("libfieldFunctionObjects.so");
    fields
    (
        T
    );
    operation        volAverage;
    regionType        all;
    postOperation     mag;
    writeFields        false;
    scalingFactor     1.0;
    writePrecision     8;
    writeToFile        true;
    useUserTime        true;
    region            region0;
    enabled            true;
    log                true;
    timeStart          0;
    timeEnd            1;
    executeControl     timeStep;
    executeInterval    1;
    writeControl        timeStep;
    writeInterval      1;
}

```

(b)

**Figure 10:** (a) Implementation of simulation variable dependent condition, (b) Implementation of volAverage Temperature field

Probe function utility has been used to visualize the imposed boundary condition during the simulation similar to case 4.1. **Figure 11** illustrates the variation of the x-component of velocity with time for one of the locations at the top wall and the variation of volume averaged temperature with time.



**Figure 11:** Variation of Volume averaged temperature with time, and Variation of x-component of velocity at top wall (0.5,1,1) with time

## 5. Conclusion

This report has shown the applications of ‘codedFixedValue’ boundary conditions in OpenFOAM. The ‘codedFixedValue’ boundary condition offers significant advantages over traditional ‘fixedValue’ conditions by enabling dynamic control over boundary values based on user-defined code. Changing boundary conditions in real-time based on simulation results, and variables and implementing conditions that vary over time to capture transient phenomena opens doors to more realistic and intricate simulations. By leveraging the capabilities of ‘codedFixedValue’, researchers and engineers can achieve a greater degree of control over their OpenFOAM simulations, leading to more accurate and insightful results.

## Acknowledgements

I am extremely grateful to the FOSSEE team for giving me the opportunity to participate in a summer fellowship. I am also deeply indebted to my advisor **Dr. Harikrishnan S**, for his unwavering support, guidance, and encouragement throughout the fellowship. His expertise in the field of Computational Fluid Dynamics

(CFD) has been instrumental in shaping my understanding of this complex domain and instilling my curiosity to do more. I would also like to extend my heartfelt thanks to my mentor, **Mr. John Pinto**, for his mentorship and profound insights into OpenFOAM. His support has been crucial in the successful completion of my report.

## REFERENCES

- [1] Huang, J., Carrica, P.M. and Stern, F. (2010). A method to compute ship exhaust plumes with waves and wind. *International Journal for Numerical Methods in Fluids*, 68(2), pp.160–180
- [2] www.openfoam.com. (n.d.). *OpenFOAM: User Guide: Tutorials*. [online] Available at: <https://www.openfoam.com/documentation/guides/v2112/doc/guide-tutorials.html> [Accessed 20 Jul. 2024].