PDE-ODE Solution Using OpenFoam

Nasr Ghoniem

August 2024

1 Example Equations

Let's assume we have the following set of equations:

$$\frac{\partial y[1]}{\partial t} = D_1 \nabla^2 y[1] + P(x) - \alpha y[1] y[2] - \beta y[1] y[3] - \lambda_1 y[1] \quad (1)$$

$$\frac{\partial y[2]}{\partial t} = D_2 \nabla^2 y[2] + P(x) - \alpha y[1] y[2] - \lambda_1 y[2] \quad (2)$$

$$\frac{\partial y[3]}{\partial t} = D_3 \nabla^2 y[3] + K(x) \lambda_2 y[3] \quad (3)$$

$$\frac{dy[4]}{dt} = \beta y[1] y[3] - \lambda_3 y[4] \quad (4)$$

The solution space and boundary conditions are shown in Fig 1.

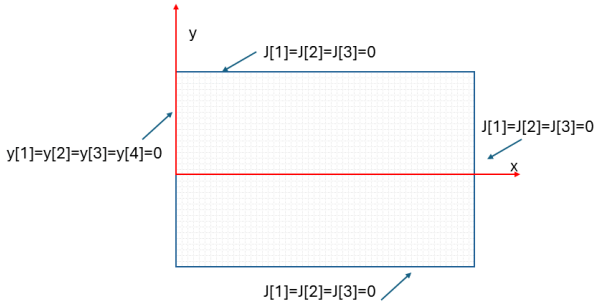

Figure 1: Solution space

The input to this problem would be the diffusion coefficients D_1, D_2, D_3, the production rates (P(x), K(x)), which are functions of space, the reaction constants (α, β) and the linear loss constants (λ_1, λ_2).

2 OpenFoam Implementation

Yes, OpenFOAM and similar finite volume method (FVM) frameworks can be adapted to solve systems that involve both coupled partial differential equations (PDEs) and ordinary differential equations (ODEs). While OpenFOAM is primarily designed for solving PDEs using the finite volume method, it can be extended or customized to include ODEs in the simulation. Here's how this can be done:

1. **Coupled PDEs and ODEs in OpenFOAM**

In many physical systems, PDEs describe the spatial and temporal evolution of fields (like temperature, concentration, or velocity), while ODEs might describe local processes such as chemical reactions, population dynamics, or other phenomena that do not explicitly depend on spatial variables.

2. **Approach to Solve Coupled PDEs and ODEs**

A. **Integrating ODEs into the Finite Volume Framework**
- **Source Terms**: ODEs can be integrated into the PDEs as source terms. For instance, in a reaction-diffusion system, the reaction kinetics described by ODEs can be included as source terms in the PDEs. - **Implicit or Explicit Integration**: The ODEs can be solved explicitly at each time step, or they can be integrated implicitly, depending on the stiffness of the equations.

B. **Custom Solvers in OpenFOAM**
- **Custom Solver Development**: If the problem requires solving ODEs coupled with PDEs, you may need to develop a custom solver in OpenFOAM. This involves extending existing solvers to include ODEs. - **‘odeFoam‘**: Although not a standard OpenFOAM solver, ‘odeFoam‘ is an example of a custom solver developed by users for solving coupled ODEs and PDEs. You can find community-contributed solvers like ‘odeFoam‘ or develop your own based on similar principles.

C. **Example: Reaction-Diffusion System**
- **PDEs**: The PDEs describe the diffusion of chemical species. - **ODEs**: The ODEs describe the reaction kinetics (e.g., chemical reactions) at each point in space.

In this case, the reaction terms in the PDEs would involve solving ODEs that describe how the concentration of species changes over time due to reactions.

3. **Implementing the Solution**

Here's a basic outline of how you might structure the solution:

1. **Discretize the PDEs**: Use the finite volume method to discretize the PDEs over the computational domain.

2. **Integrate ODEs Locally**:
- For each cell in the mesh, integrate the ODEs using an appropriate numerical method (e.g., Runge-Kutta, Euler). - These ODEs would typically be solved at each time step, updating the source terms in the PDEs.

3. **Time-Stepping Loop**:
- For each time step: - Solve the PDEs using the finite volume method. - Update the source terms by solving the ODEs. - Write the results for post-processing.

4. **Custom C++ Code**: - You may need to write custom C++ code within the OpenFOAM framework to implement the ODE integration. This might involve modifying existing solvers or creating a new one.

To create code specific to the problem described in the image, we will need to solve a system of coupled partial differential equations (PDEs) and ordinary differential equations (ODEs) within a defined solution space and boundary conditions. Given the structure of the equations and the boundary conditions:

Governing Equations

The system involves four equations: 1. $\frac{\partial y[1]}{\partial t} = D_1\nabla^2 y[1] + P(x) - \alpha y[1]y[2] - \beta y[1]y[3] - \lambda_1 y[1]$ 2. $\frac{\partial y[2]}{\partial t} = D_2\nabla^2 y[2] + P(x) - \alpha y[1]y[2] - \lambda_1 y[2]$ 3. $\frac{\partial y[3]}{\partial t} = D_3\nabla^2 y[3] + K(x)\lambda_2 y[3]$ 4. $\frac{dy[4]}{dt} = \beta y[1]y[3] - \lambda_3 y[4]$

Boundary Conditions

The boundary conditions are: - $J[1] = J[2] = J[3] = 0$ at the boundaries. - $y[1] = y[2] = y[3] = y[4] = 0$ at the initial state or certain boundary conditions.

Pseudo-Code Specific to this Problem

The following C++ pseudo-code is tailored to this problem using a framework like OpenFOAM or a custom solver. The main components include the setup of the PDEs, the boundary conditions, and the time-stepping loop.

```cpp
#include "fvCFD.h"

int main(int argc, char *argv[])
{
    #include "setRootCase.H"
    #include "createTime.H"
    #include "createMesh.H"
    #include "createFields.H"

    // Define diffusion coefficients, reaction terms, and other constants
    const scalar D1 = 1.0; // Diffusion coefficient for y[1]
    const scalar D2 = 1.0; // Diffusion coefficient for y[2]
    const scalar D3 = 1.0; // Diffusion coefficient for y[3]
    const scalar alpha = 1.0; // Reaction rate constant
    const scalar beta = 1.0;  // Reaction rate constant
    const scalar lambda1 = 1.0;
    const scalar lambda2 = 1.0;
```

```
const scalar lambda3 = 1.0;

// Define spatially varying functions P(x) and K(x)
volScalarField P = ... ; // Define P as a function of x
volScalarField K = ... ; // Define K as a function of x

// Define the time loop
for (runTime++; !runTime.end(); runTime++)
{
    Info << "Time = " << runTime.timeName() << nl << endl;

    // Solve the first PDE
    solve
    (
        fvm::ddt(y1)
      - fvm::laplacian(D1, y1)
      + P
      - alpha * y1 * y2
      - beta * y1 * y3
      - lambda1 * y1
    );

    // Solve the second PDE
    solve
    (
        fvm::ddt(y2)
      - fvm::laplacian(D2, y2)
      + P
      - alpha * y1 * y2
      - lambda1 * y2
    );

    // Solve the third PDE
    solve
    (
        fvm::ddt(y3)
      - fvm::laplacian(D3, y3)
      + K * lambda2 * y3
    );

    // Solve the ODE for y4
    forAll(mesh, cellI)
    {
        y4[cellI] += dt * (beta * y1[cellI] * y3[cellI] - lambda3 * y4[cellI]);
    }
```

```
        runTime.write(); // Write results for post-processing
    }

    Info << "End" << endl;
    return 0;
}
```

Explanation:
- **Diffusion Coefficients**: 'D1', 'D2', and 'D3' represent the diffusion coefficients for 'y[1]', 'y[2]', and 'y[3]', respectively.

- **Reaction Terms**: 'alpha', 'beta', 'lambda1', 'lambda2', and 'lambda3' are constants defining the reaction rates and decay terms.
- **Spatial Functions**: 'P(x)' and 'K(x)' are defined as spatially varying fields. You would need to implement these based on the problem specifics.
- **Time-Stepping Loop**: The code loops over time steps, solving the PDEs for 'y[1]', 'y[2]', and 'y[3]' and the ODE for 'y[4]' at each time step.
- **Boundary and Initial Conditions**: These should be defined in the 'createFields.H' and boundary condition setup files, typically by initializing fields and applying conditions at the mesh boundaries.

Customization: - **Mesh and Solver Settings**: The mesh generation, solver settings, and boundary conditions would need to be defined in the associated setup files in an OpenFOAM environment.
- **Function Definitions**: The spatial functions 'P(x)' and 'K(x)' need to be defined based on your specific problem requirements.
- **Post-Processing**: Use OpenFOAM tools or external software like ParaView to visualize the results based on the written output files.

This structure will allow you to solve the system of PDEs and ODEs as described in your problem statement.