

subFlow Library Documentation

A Lightweight C++ Framework for Multiphase Flow Simulation
in Porous Media

Generated Documentation

February 5, 2026

Contents

1 Introduction

subFlow is a lightweight, modular C++ library designed for analyzing multiphase flow in porous media with three phases: water, oil, and gas. The library implements a coupled numerical scheme combining a locally conservative Darcy solver with a saturation transport solver to simulate phase displacement and transport in heterogeneous reservoirs.

1.1 Key Features

- **Multiphase Formulation:** Water–oil–gas systems with flexible phase combinations
- **Darcy Flow Solver:** Compressible and incompressible options using locally conservative finite element formulation with $\mathbf{H}(\text{div})-L^2$ approximation pairs for total flux and pressure
- **Transport Solver:** Finite Volume scheme with robust upwinding, conservation properties, and gravitational segregation using Implicit Hybrid Upwind (IHU) strategy
- **Coupling Strategy:** Sequential Fully Implicit (SFI) method for handling strong nonlinear coupling between flow and transport
- **Flexible I/O:** Parameterized JSON-based input configuration and support for mesh files (.geo, .msh)
- **Modular Design:** Clear interfaces enabling extension and customization of physics, discretization, and coupling strategies

1.2 Typical Workflow

The standard workflow for simulating multiphase flow with subFlow consists of four main stages:

1. **Mesh and Data Preparation:** Build or import the computational mesh and provide rock/fluid properties
2. **Darcy Problem Solution:** Solve for pressure and total flux using $\mathbf{H}(\text{div})-L^2$ finite element method
3. **Transport Advancement:** Advance saturation profiles using Finite Volume transport solver with proper flux handling
4. **SFI Coupling:** Iterate until convergence for each time step using the Sequential Fully Implicit coupling scheme

2 Preprocessing Classes

Preprocessing classes handle the preparation of the computational problem, including reading input data, building meshes, and setting up the approximation spaces.

2.1 TSFProblemData

Purpose: Central data container that stores all information required to set up a reservoir problem.

Location: `src/TSFProblemData.h`

Responsibilities:

- Read and parse JSON configuration files containing simulation parameters
- Store geometric mesh information, domain properties, and boundary conditions
- Manage fluid properties (density, viscosity, compressibility models)
- Store rock properties (permeability, porosity) for each domain
- Manage numerical parameters (analysis type, time stepping, solver tolerances)
- Handle initial conditions (pressure and saturation profiles)
- Provide petro-physical parameters (relative permeability models, residual saturations)

Key Methods:

```

1 // Read problem data from JSON file
2 void ReadJSONFile(std::string filename);
3
4 // Serialization methods for saving/loading state
5 void Write(TPZStream &buf, int withclassid) const;
6 void Read(TPZStream &buf, void *context);
```

Data Structures:

The class contains nested structures for organizing related data:

Stores mesh information including domain names, material IDs, mesh source (GMSH), and standard material IDs for various element types

Contains numerical parameters: analysis type, time stepping, convergence tolerances, iteration limits, thread counts

Fluid data: density, viscosity, compressibility models for water and gas

Rock properties: relative permeability model and residual saturations

Initial conditions for pressure and saturation

Output control: post-processing frequency, threading options, VTK resolution

2.2 TSFApproxCreator

Purpose: Builds the finite element approximation spaces and computational meshes for Darcy and transport problems.

Location: `src/TSFApproxCreator.h`

Responsibilities:

~~TGeomElementSpaceDarcy~~ Create finite element space ($\mathbf{H}(\text{div})$ for fluxes, L^2 for pressure)

- Create atomic computational meshes for different physical spaces
- Construct multiphysics computational mesh coupling flux and pressure spaces
- Add material objects for Darcy equation
- Condense high-order elements to improve computational efficiency
- Build auxiliary transport mesh for post-processing and interface identification
- Insert interface elements between subdomains

Key Methods:

```

1 // Set problem data reference
2 void SetProblemData(TSFProblemData *simData);
3
4 // Configure Darcy approximation space
5 void ConfigureDarcySpace();
6
7 // Add Darcy materials to the mesh
8 void AddDarcyMaterials();
9
10 // Create complete approximation space (main driver)
11 TPZMultiphysicsCompMesh *CreateApproximationSpace();
12
13 // Condense elements for efficiency
14 void CondenseElements(TPZCompMesh *cmesh,
15                         char LagrangeLevelNotCondensed,
16                         bool keepmatrix = true);
17
18 // Build auxiliary transport mesh
19 void BuildTransportCmesh();
20
21 // Create interface elements
22 void CreateInterfaceElements();
23
24 // Get transport mesh reference
25 TPZCompMesh *GetTransportCmesh();

```

2.3 TPZFastCondensedElement

Purpose: Implements static condensation of high-order element degrees of freedom to reduce the system size.

Location: `src/TPZFastCondensedElement.h`

Responsibilities:

- Perform element-level static condensation (Schur complement)
- Eliminate interior degrees of freedom without solving the full system

- Improve computational efficiency for high-order approximations
- Maintain matrix structure for efficient storage and assembly

3 Input File Format

subFlow uses JSON-based configuration files to define simulation parameters. This section describes the structure and available options.

3.1 JSON File Structure Overview

A complete subFlow configuration file contains six main sections:

```

1  {
2      "UseGMsh": boolean,           // Use GMSH mesh format
3      "MshFile": "filename.msh",   // GMSH mesh file path
4      "Dimension": integer,       // Problem dimension (2 or 3)
5      "Domains": [...],          // Domain properties
6      "Boundary": [...],         // Boundary conditions
7      "Numerics": {...},         // Numerical parameters
8      "FluidProperties": {...},   // Fluid properties
9      "PetroPhysics": {...},     // Rock properties
10     "ReservoirProperties": {...}, // Initial conditions
11     "PostProcess": {...}        // Output control
12 }
```

3.2 Mesh Configuration

3.2.1 UseGMsh

```
1 "UseGMsh": true|false
```

Specifies whether to read mesh from a GMSH file. If `false`, the mesh is generated internally.

3.2.2 MshFile

```
1 "MshFile": "path/to/mesh.msh"
```

Path to the GMSH mesh file (relative to the input directory). Required if `UseGMsh` is `true`.

3.2.3 Dimension

```
1 "Dimension": 2|3
```

Problem spatial dimension (2D or 3D).

3.3 Domains Section

Defines regions within the reservoir with homogeneous properties.

```

1 "Domains": [
2     {
3         "name": "domain_name",           // String identifier
4         "matid": integer,              // Unique material ID
5         "K": float,                  // Permeability (absolute)
6         "phi": float                 // Porosity (0 to 1)
7     },
8     ...
9 ]
```

Example:

```

1 "Domains": [
2     {
3         "name": "sandstone",
4         "matid": 1,
5         "K": 1.83e-5,
6         "phi": 0.3
7     }
8 ]

```

3.4 Boundary Section

Defines boundary conditions on the domain edges/surfaces.

```

1 "Boundary": [
2     {
3         "name": "boundary_name",           // String identifier
4         "matid": integer,                // Unique material ID
5         "type": integer,                // 0: Dirichlet (pressure),
6                                         // 1: Neumann (flux)
7         "value": float,                 // BC value
8         "functionID": integer,          // Function ID (reserved)
9         "ExternalSaturation": float,   // Saturation for injection
10        "SaturationFunctionID": integer // Saturation function ID
11    },
12    ...
13 ]

```

Example:

```

1 "Boundary": [
2     {
3         "name": "inlet",
4         "matid": 5,
5         "type": 0,
6         "value": 10.0,
7         "functionID": 0,
8         "ExternalSaturation": 1.0,
9         "SaturationFunctionID": 0
10    },
11    {
12        "name": "outlet",
13        "matid": 3,
14        "type": 0,
15        "value": 0.0,
16        "functionID": 0,
17        "ExternalSaturation": 0.0,
18        "SaturationFunctionID": 0
19    }
20 ]

```

3.5 Numerics Section

Controls the numerical solution method and parameters.

```

1 "Numerics": {
2     "AnalysisType": integer,           // 0: Darcy only,
3                                         // 1: Transport only,

```

```

4           "FluxOrder": integer,           // 2: Coupled SFI
5           "DeltaT": float,             // Order of Hdiv space (1 or 2)
6           "NSteps": integer,          // Time step size
7           "Gravity": [gx, gy, gz],    // Number of time steps
8           "IsAxisymmetric": boolean, // Gravity vector
9           "IsLinearTrace": boolean, // Axisymmetric formulation
10          "FourApproxSpaces": boolean, // Linear tracer (no saturation)
11          "NThreadsDarcy": integer, // Use 4-space mixed formulation
12          "MaxIterSFI": integer,    // Threading for Darcy solver
13          "MaxIterSFI": integer,    // Max SFI iterations
14          "TolSFI": float,          // SFI convergence tolerance
15          "MaxIterDarcy": integer, // Max Newton iterations (Darcy)
16          "ResTolDarcy": float,     // Residual tolerance (Darcy)
17          "CorrTolDarcy": float,    // Correction tolerance (Darcy)
18          "MaxIterTransport": integer, // Max iterations (Transport)
19          "ResTolTransport": float,   // Residual tolerance (Transport)
20          "CorrTolTransport": float // Correction tolerance (Transport)
21      }

```

Example:

```

1  "Numerics": {
2      "AnalysisType": 2,
3      "FluxOrder": 1,
4      "DeltaT": 0.001,
5      "NSteps": 100,
6      "Gravity": [0.0, -9.81, 0.0],
7      "IsAxisymmetric": false,
8      "IsLinearTrace": false,
9      "FourApproxSpaces": true,
10     "NThreadsDarcy": 0,
11     "MaxIterSFI": 5,
12     "TolSFI": 1.0e-6,
13     "MaxIterDarcy": 10,
14     "ResTolDarcy": 1.0e-6,
15     "CorrTolDarcy": 1.0e-6,
16     "MaxIterTransport": 10,
17     "ResTolTransport": 1.0e-6,
18     "CorrTolTransport": 1.0e-6
19 }

```

3.6 Fluid Properties Section

Defines fluid characteristics for water and gas phases.

```

1  "FluidProperties": {
2      "WaterDensity": float,           // Reference water density
3      "WaterViscosity": float,        // Water dynamic viscosity
4      "WaterCompressibility": float, // Water compressibility
5      "GasDensity": float,           // Reference gas density
6      "GasViscosity": float,         // Gas dynamic viscosity
7      "GasCompressibility": float,  // Gas compressibility
8      "DensityModel": integer,       // 0: Linear, 1: Exponential
9      "ReferencePressure": float    // Reference pressure for models
10 }

```

Example:

```

1  "FluidProperties": {

```

```

2     "WaterDensity": 1000.0,
3     "WaterViscosity": 0.001,
4     "WaterCompressibility": 0.0,
5     "GasDensity": 1.225,
6     "GasViscosity": 1.81e-5,
7     "GasCompressibility": 0.0,
8     "DensityModel": 0,
9     "ReferencePressure": 101325.0
10 }

```

3.7 Petro-Physics Section

Rock-fluid interaction parameters.

```

1 "PetroPhysics": {
2     "KrModel": integer,      // 0: Linear, 1: Quadratic
3     "Swr": float,           // Residual water saturation
4     "Sgr": float            // Residual gas saturation
5 }

```

Example:

```

1 "PetroPhysics": {
2     "KrModel": 1,
3     "Swr": 0.1,
4     "Sgr": 0.05
5 }

```

3.8 Reservoir Properties Section

Initial conditions for pressure and saturation fields.

```

1 "ReservoirProperties": {
2     "s0": {
3         "functionType": integer, // 0: Constant, 1: Spatial
4         "value": float          // Initial value or parameter
5     },
6     "p0": {
7         "functionType": integer, // 0: Constant, 1: Spatial
8         "value": float          // Initial value or parameter
9     }
10 }

```

Example:

```

1 "ReservoirProperties": {
2     "s0": {
3         "functionType": 0,
4         "value": 0.2
5     },
6     "p0": {
7         "functionType": 0,
8         "value": 0.0
9     }
10 }

```

3.9 Post-Processing Section

Controls output and visualization options.

```
1 "PostProcess": {  
2     "PostProcessFrequency": integer,      // Write output every N steps  
3     "NThreads": integer,                  // Threads for post-processing  
4     "VTKResolution": integer            // VTK mesh refinement level  
5 }
```

Example:

```
1 "PostProcess": {  
2     "PostProcessFrequency": 1,  
3     "NThreads": 0,  
4     "VTKResolution": 0  
5 }
```

4 Analysis Classes

Analysis classes implement the solution procedures for the flow and transport problems and their coupling.

4.1 TSFDarcyAnalysis

Purpose: Solves the Darcy problem for pressure and flux using the $\mathbf{H}(\text{div}) - L^2$ mixed finite element formulation.

Location: `src/TSFDarcyAnalysis.h`

Parent Class: `TPZLinearAnalysis`

Responsibilities:

- Manage the Darcy linear system assembly and solution
- Handle nonlinear iterations when densities depend on pressure
- Apply boundary conditions (Dirichlet and Neumann)
- Track number of iterations and timing information
- Post-process flux and pressure solutions
- Verify element flux conservation (divergence-free properties)

Key Methods:

```

1 // Initialization with problem data
2 void Initialize();
3 void SetProblemData(TSFProblemData *simData);
4 TSFProblemData *GetProblemData();

5
6 // Execute a time step
7 void RunTimeStep(std::ostream &out = std::cout);

8
9 // Post-process and output results
10 void PostProcessTimeStep(int dimToPost = -1, int step = -1);

11
12 // Perform Newton iteration for nonlinear problems
13 void NewtonIteration();

14
15 // System assembly and solving
16 void Assemble() override;
17 void Solve() override;

18
19 // Boundary condition handling
20 void FillNeumannBCMatids(std::set<int> &neumannMatids);
21 void SetInitialBCValue(std::set<int> &neumannMatids);
22 void ApplyEquationFilter(std::set<int> &neumannMatids);

23
24 // Verification
25 void VerifyElementFluxes();

```

Data Members:

```

1 TSFProblemData *fSimData; // Problem configuration
2 int fKiteration; // Current iteration number
3 bool fIsFirstAssemble; // Flag for first assembly

```

4.2 TSFTransportAnalysis

Purpose: Solves the saturation transport problem using a Finite Volume scheme with upwinding.

Location: `src/TSFTransportAnalysis.h`

Parent Class: `TPZLinearAnalysis`

Responsibilities:

- Assemble and solve the transport equation for saturation
- Manage mass matrix (assembled once) and transmissibility matrix (time-dependent)
- Update fluid properties (density) and coefficients based on current solution
- Handle implicit time integration
- Apply Finite Volume upwinding strategies
- Post-process saturation and other transport variables

Key Methods:

```

1 // Initialization
2 void Initialize();
3 void SetProblemData(TSFProblemData *simData);
4 TSFProblemData *GetProblemData();

5
6 // Execute a time step
7 void RunTimeStep(std::ostream &out = std::cout);

8
9 // Post-process results
10 void PostProcessTimeStep(int dimToPost = -1, int step = -1);

11
12 // Matrix assembly
13 void AssembleMass();           // Assemble mass matrix (once)
14 void Assemble() override;     // Assemble transmissibility and RHS
15
16 // Solving
17 void Solve() override;
18
19 // Coefficient updates
20 void UpdateDensityAndCoefficients();

21
22 // Verification
23 void VerifyElementFluxes();

```

Data Members:

```

1 TSFProblemData *fSimData;
2 TSFAlgebraicTransport fAlgebraicTransport;
3 int fKiteration;
4 bool fIsFirstAssemble;
5 // Sparse matrices for efficiency
6 TPZFYSmpMatrix<REAL> *fTransmissibilityMatrix;
7 TPZFMMatrix<REAL> fMassMatrix;

```

4.3 TSFSFIAnalysis

Purpose: Implements the Sequential Fully Implicit (SFI) coupling scheme between Darcy and transport problems.

Location: `src/TSFSFIAnalysis.h`

Parent Class: `TPZLinearAnalysis`

Responsibilities:

- Manage Darcy and transport analysis objects
- Implement data transfer between flow and transport meshes
- Execute SFI iteration loops until convergence
- Control time stepping for coupled simulations
- Transfer flux information from Darcy to transport
- Transfer saturation information from transport to Darcy
- Post-process coupled solutions

Key Methods:

```

1 // Initialization and setup
2 void SetProblemData(TSFProblemData *simData);
3 void Initialize();
4
5 // Run complete simulation
6 void Run(std::ostream &out = std::cout) override;
7
8 // Run single time step with SFI iterations
9 void RunTimeStep(std::ostream &out = std::cout);
10
11 // Data transfer
12 void TransferDarcyToTransport(); // Transfer flux to transport mesh
13 void TransferTransportToDarcy(); // Transfer saturation to Darcy
14
15 // State management
16 void UpdateLastStateVariables();
17
18 // Post-processing
19 void PostProcessTimeStep(const int type, const int dim,
                           int step = -1);
20

```

Data Members:

```

1 TSFProblemData *fSimData;
2 int fKiteration;
3 bool fShouldSolveDarcy; // Solve Darcy only once for linear
                         tracer
4 TSFDarcyAnalysis fDarcyAnalysis;
5 TSFTransportAnalysis fTransportAnalysis;
6 TSFDataTransfer fDataTransfer;
7 TPZFMMatrix<STATE> fDarcySolution;
8 TPZFMMatrix<STATE> fTransportSolution;

```

5 Material Classes

Material classes define the physics of the Darcy and transport equations.

5.1 TSFMixedDarcy

Purpose: Implements the weak form of the mixed Darcy equation for multiphase flow.

Location: `src/TSFMixedDarcy.h`

Parent Class: `TPZMixedDarcyFlow`

Responsibilities:

- Compute volumetric contributions to stiffness matrix and load vector
- Compute boundary condition contributions
- Handle gravity effects
- Support axisymmetric formulations
- Provide post-processing solutions (velocity, pressure gradients)
- Support both 3-space and 4-space mixed formulations

Key Methods:

```

1 // Volumetric contributions to weak form
2 void Contribute(const TPZVec<TPZMaterialDataT<STATE>> &datavec,
3                  REAL weight, TPZMatrix<STATE> &ek,
4                  TPZMatrix<STATE> &ef) override;
5
6 // Additional space contributions (for 4-space formulation)
7 void ContributeFourSpaces(const TPZVec<TPZMaterialDataT<STATE>> &
8                           datavec,
9                           REAL weight, TPZMatrix<STATE> &ek,
10                          TPZMatrix<STATE> &ef);
11
12 // Boundary condition contributions
13 void ContributeBC(const TPZVec<TPZMaterialDataT<STATE>> &datavec,
14                    REAL weight, TPZMatrix<STATE> &ek,
15                    TPZMatrix<STATE> &ef,
16                    TPZBndCondT<STATE> &bc) override;
17
18 // Data requirements specification
19 void FillDataRequirements(
20     TPZVec<TPZMaterialDataT<STATE>> &datavec) const override;
21 void FillBoundaryConditionDataRequirements(
22     int type, TPZVec<TPZMaterialDataT<STATE>> &datavec) const override;
23
24 // Physics control methods
25 void SetAxisymmetry(bool IsAxisymmetric);
26 bool IsAxisymmetric() const;
27 void SetGravity(const TPZFNMatrix<3, REAL> &gravity);
28 const TPZFNMatrix<3, REAL> &GetGravity() const;
29 void SetFourSpaces(bool fourSpaces);
30 bool IsFourSpaces() const;
31
32 // Post-processing
33 void Solution(const TPZVec<TPZMaterialDataT<STATE>> &datavec,
34               int var, TPZVec<REAL> &Solout);

```

5.2 TSFTransportMaterial

Purpose: Implements the weak form of the saturation transport equation with upwinding and gravity effects.

Location: `src/TSFTransportMaterial.h`

Parent Class: Multiple-inheritance material supporting single and interface contributions

Responsibilities:

- Compute volumetric transport equation contributions
- Compute interface (edge/face) flux contributions
- Implement upwinding strategies
- Handle gravity segregation
- Support implicit time integration
- Provide saturation and density post-processing

Key Methods:

```

1 // Data requirements
2 void FillDataRequirements(
3     TPZVec<TPZMaterialDataT<STATE>> &datavec) const override;
4 void FillBoundaryConditionDataRequirements(
5     int type, TPZVec<TPZMaterialDataT<STATE>> &datavec) const override;
6 void FillDataRequirementsInterface(
7     TPZMaterialDataT<STATE> &data) const override;
8
9 // Physics contributions
10 void Contribute(const TPZVec<TPZMaterialDataT<STATE>> &datavec,
11                  REAL weight, TPZFMATRIX<STATE> &ek,
12                  TPZFMATRIX<STATE> &ef) override;
13 void ContributeBC(const TPZVec<TPZMaterialDataT<STATE>> &datavec,
14                     REAL weight, TPZFMATRIX<STATE> &ek,
15                     TPZFMATRIX<STATE> &ef,
16                     TPZBndCondT<STATE> &bc) override;
17
18 // Post-processing
19 int VariableIndex(const std::string &name) const override;
20 int NSolutionVariables(int var) const override;
21 void Solution(const TPZVec<TPZMaterialDataT<STATE>> &datavec,
22               int var, TPZVec<REAL> &Solout) override;
23
24 // Property accessors
25 int Dimension() const;
26 void SetDimension(int dim);
27 int NStateVariables() const;
28 void Print(std::ostream &out) const override;

```

Data Members:

```

1 int m_dimension;           // Problem dimension
2 int m_mat_id;             // Material ID
3 bool m_mass_matrix_Q;    // Mass matrix assembly flag
4 REAL m_dt;                // Time step size
5 REAL m_phi;               // Porosity
6 REAL m_fracture_epsilon; // Fracture parameter

```

6 Data Transfer and Utility Classes

6.1 TSFDataTransfer

Purpose: Manages data transfer between Darcy and transport meshes in the SFI coupling scheme.

Location: `src/TSFDataTransfer.h`

Responsibilities:

- Map and transfer flux from Darcy mesh to transport mesh interfaces
- Map and transfer saturation from transport mesh to Darcy materials
- Maintain mappings between computational meshes and algebraic transport structure
- Handle gather/scatter operations for efficient data movement
- Support interface-volume data associations

Nested Structures:

Manages flux transfer from Darcy to transport including gather/scatter vectors and matrix pointers

Establishes correspondence between algebraic transport cells and Darcy elements

Associates interface elements with their adjacent volume elements

6.2 TSFAlgebraicTransport

Purpose: Provides algebraic representation of the transport problem for efficient numerical operations.

Location: `src/TSFAlgebraicTransport.h`

Responsibilities:

~~• Optimizes element interface~~ data for transport equations

- Store and manage fluid property arrays (density, viscosity)
- Maintain saturation and pressure fields
- Handle relative permeability and fractional flow calculations
- Support efficient matrix-free or matrix-explicit operations

Nested Structures:

Stores all cell-centered data including volume, material ID, pressure, saturation, fluid properties, and transport coefficients

Stores interface flux information including flux coefficients, integral values, and face orientations

6.3 TSFSavable

Purpose: Base class providing serialization capabilities for subFlow objects.

Location: `src/TSFSavable.h`

Responsibilities:

~~TCellDataProvider~~ save/load infrastructure for simulation state

- Support checkpoint and restart capabilities
- Enable long-term data persistence

7 Class Hierarchy Diagram

```
TPZSavable
  +-- TSFSavable
    +-- TSFProblemData

TPZLinearAnalysis
  +-- TSFDarcyAnalysis
  +-- TSFTransportAnalysis
  +-- TSFSFIAnalysis

TPZHDivApproxCreator
  +-- TSFApproxCreator

TPZMixedDarcyFlow
  +-- TSFMixedDarcy

TPZMatBase (with multiple inheritance)
  +-- TSFTransportMaterial

Utility Classes:
  +-- TSFDataTransfer
  +-- TSFAlgebraicTransport
  +-- TPZFastCondensedElement
```

8 Workflow and Data Flow Diagram

This section presents a comprehensive flowchart showing how classes interact during a typical multiphase flow simulation.

8.1 Complete Simulation Workflow

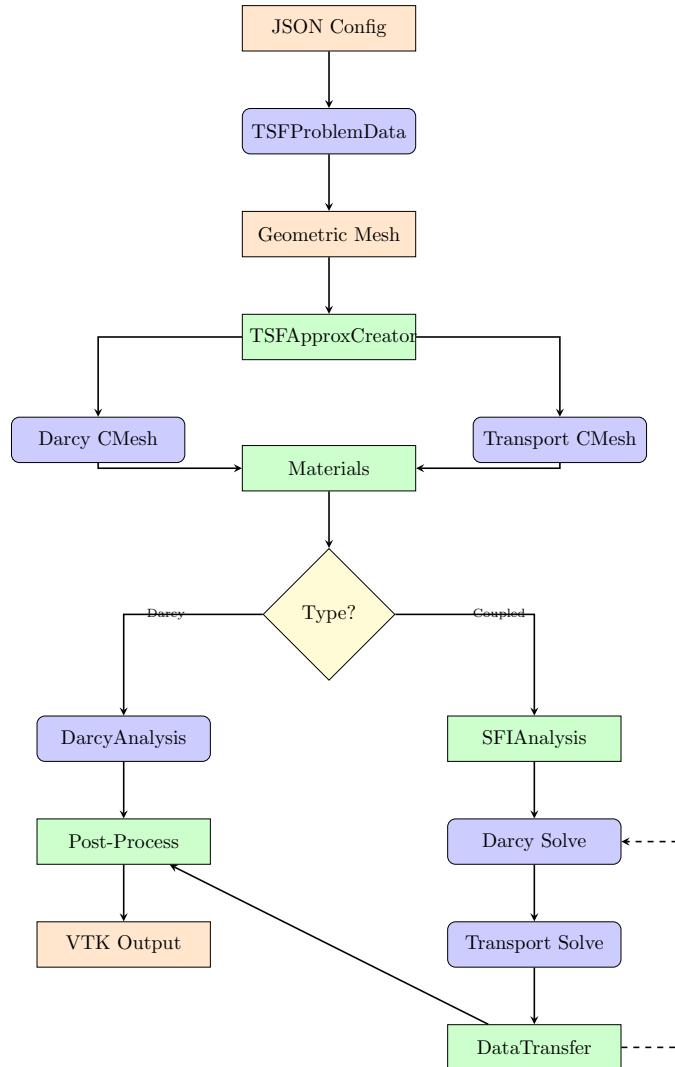


Figure 1: Complete workflow for multiphase flow simulation with subFlow

8.2 Data Transfer and Coupling Mechanism

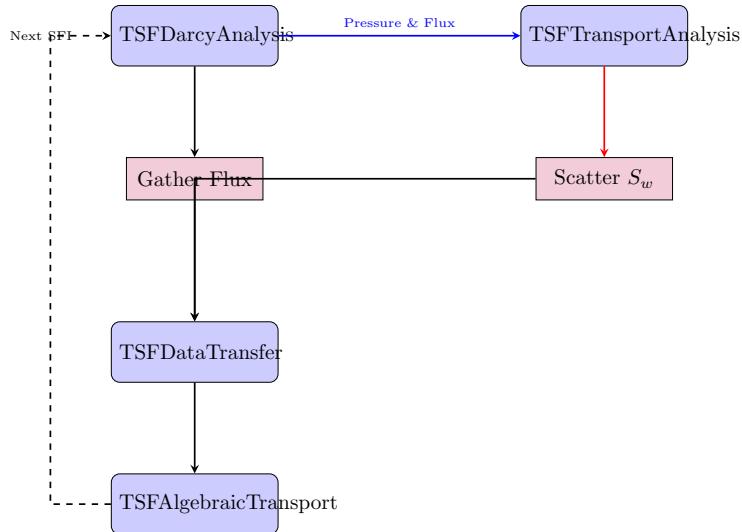


Figure 2: Data transfer mechanism in SFI coupling

9 Typical Usage Workflow

This section demonstrates how to use the subFlow library for a coupled simulation.

9.1 Basic Simulation Steps

```

1 // 1. Load problem data from JSON configuration
2 TSFProblemData simData;
3 simData.ReadJSONFile("simulation_config.json");
4
5 // 2. Build or load geometric mesh
6 TPZGeoMesh *gmesh = new TPZGeoMesh();
7 // ... populate gmesh from GMSH file or generated mesh ...
8
9 // 3. Create approximation spaces
10 TSFAproxCreator approxCreator(gmesh);
11 approxCreator.SetProblemData(&simData);
12 approxCreator.ConfigureDarcySpace();
13 approxCreator.AddDarcyMaterials();
14 TPZMultiphysicsCompMesh *darcy_cmesh =
15     approxCreator.CreateApproximationSpace();
16
17 // 4. Choose analysis type based on configuration
18 if (simData.fTNumerics.fAnalysisType == 0) {
19     // Darcy problem only
20     TSFDarcyAnalysis darcyAnalysis(darcy_cmesh);
21     darcyAnalysis.SetProblemData(&simData);
22     darcyAnalysis.Initialize();
23     darcyAnalysis.RunTimeStep();
24     darcyAnalysis.PostProcessTimeStep(
25         gmesh->Dimension(), 0);
26 } else {
27     // Coupled Darcy-Transport problem
28     approxCreator.BuildTransportCmesh();
29     TPZCompMesh *transport_cmesh =
30         approxCreator.GetTransportCmesh();
31 }
```

```

32 // Create coupled SFI solver
33 TSFSFIAnalysis sfiAnalysis(
34     darcy_cmesh, transport_cmesh);
35 sfiAnalysis.SetProblemData(&simData);
36 sfiAnalysis.Initialize();
37 sfiAnalysis.Run(); // Run complete simulation
38
39 delete transport_cmesh;
40 }
41
42 // 5. Cleanup
43 delete darcy_cmesh;
44 delete gmesh;

```

9.2 Configuration File Example

A minimal JSON configuration for a coupled 2D simulation:

```

1 {
2     "UseGMsh": true,
3     "MshFile": "reservoir_2d.msh",
4     "Dimension": 2,
5     "Domains": [
6         {
7             "name": "reservoir",
8             "matid": 1,
9             "K": 1e-4,
10            "phi": 0.25
11        },
12        "Boundary": [
13            {
14                "name": "inlet",
15                "matid": 10,
16                "type": 0,
17                "value": 100.0,
18                "functionID": 0,
19                "ExternalSaturation": 0.8,
20                "SaturationFunctionID": 0
21            },
22            {
23                "name": "outlet",
24                "matid": 11,
25                "type": 0,
26                "value": 0.0,
27                "functionID": 0,
28                "ExternalSaturation": 0.0,
29                "SaturationFunctionID": 0
30            }
31        ],
32        "Numerics": {
33            "AnalysisType": 2,
34            "FluxOrder": 1,
35            "DeltaT": 0.01,
36            "NSteps": 50,
37            "Gravity": [0.0, -9.81, 0.0],
38            "IsAxisymmetric": false,
39            "IsLinearTrace": false,
40            "FourApproxSpaces": true,
41            "NThreadsDarcy": 0,
42        }
43    }
44 }

```

```
41     "MaxIterSFI": 5,
42     "TolSFI": 1e-6,
43     "MaxIterDarcy": 10,
44     "ResTolDarcy": 1e-6,
45     "CorrTolDarcy": 1e-6,
46     "MaxIterTransport": 10,
47     "ResTolTransport": 1e-6,
48     "CorrTolTransport": 1e-6
49   },
50   "FluidProperties": {
51     "WaterDensity": 1000.0,
52     "WaterViscosity": 1e-3,
53     "WaterCompressibility": 0.0,
54     "GasDensity": 1.0,
55     "GasViscosity": 1e-5,
56     "GasCompressibility": 0.0,
57     "DensityModel": 0,
58     "ReferencePressure": 0.0
59   },
60   "PetroPhysics": {
61     "KrModel": 0,
62     "Swr": 0.0,
63     "Sgr": 0.0
64   },
65   "ReservoirProperties": {
66     "s0": {"functionType": 0, "value": 0.0},
67     "p0": {"functionType": 0, "value": 0.0}
68   },
69   "PostProcess": {
70     "PostProcessFrequency": 1,
71     "NThreads": 0,
72     "VTKResolution": 0
73   }
74 }
```

10 Post-Processing and Output

10.1 VTK Output

The library generates VTK-format output files for visualization in ParaView or other visualization tools.

10.1.1 Available Outputs

- **Geometric Mesh:** `gmesh-*.vtk` - Visualization of computational mesh before and after interface insertion
- **Darcy Solution:** `darcy-cmesh.vtk` - Pressure and flux fields
- **Transport Solution:** `transport-cmesh.vtk` - Saturation fields
- **Time Series:** Multiple files with step number for transient simulations

10.1.2 Post-Processing Frequency

Control output frequency via the JSON configuration:

```

1 "PostProcess": {
2     "PostProcessFrequency": 5,    // Write output every 5 steps
3     "NThreads": 0,
4     "VTKResolution": 0
5 }
```

10.2 Solution Variables

The post-processing methods generate various solution fields depending on the problem type.

10.2.1 Darcy Problem Variables

- Pressure field
- Total flux magnitude and components
- Velocity field
- Permeability

10.2.2 Transport Problem Variables

- Water saturation
- Gas saturation
- Fluid density
- Relative permeabilities (water and gas)
- Fractional flows

11 Advanced Features

11.1 Axisymmetric Formulations

For cylindrical symmetry problems, enable axisymmetric mode:

```

1 "Numerics": {
2     "IsAxisymmetric": true,
3     ...
4 }
```

The Darcy and transport solvers will automatically adjust weak forms and integration for axisymmetric geometry.

11.2 Four-Space Mixed Formulation

The four-space mixed formulation includes an additional space of Lagrange multipliers for improved robustness:

```

1 "Numerics": {
2     "FourApproxSpaces": true,
3     ...
4 }
```

This is enabled via static condensation in `TSFAproxCreator::CondenseElements()`.

11.3 Compressible Fluids

For compressible fluids, set non-zero compressibilities in the JSON:

```

1 "FluidProperties": {
2     "WaterCompressibility": 1e-9,
3     "GasCompressibility": 0.01,
4     "DensityModel": 0, // Linear model
5     "ReferencePressure": 101325.0
6 }
```

Density variations with pressure require Newton iterations in the Darcy solver.

The library supports two compressibility models for fluid density:

Linear Density Model (DensityModel = 0):

$$\rho(p) = \rho_{\text{ref}} [1 + c(p - p_{\text{ref}})]$$

where:

- $\rho(p)$ is the fluid density at pressure p
- ρ_{ref} is the reference density (WaterDensity or GasDensity)
- c is the compressibility coefficient (WaterCompressibility or GasCompressibility)
- p_{ref} is the reference pressure (ReferencePressure)

Exponential Density Model (DensityModel = 1):

$$\rho(p) = \rho_{\text{ref}} \exp [c(p - p_{\text{ref}})]$$

For incompressible fluids, set $c = 0$ to obtain constant density $\rho(p) = \rho_{\text{ref}}$.

11.4 Relative Permeability Models

Control the phase relative permeability computation:

```

1 "PetroPhysics": {
2     "KrModel": 0,      // 0: Linear, 1: Quadratic
3     "Swr": 0.1,        // Residual water saturation
4     "Sgr": 0.05        // Residual gas saturation
5 }
```

The library supports two relative permeability models for both water and gas phases:

Linear Model (KrModel = 0):

Water relative permeability:

$$k_{r,w}(S_w) = \begin{cases} 0 & \text{if } S_w \leq S_{wr} \\ \frac{S_w - S_{wr}}{1 - S_{wr}} & \text{if } S_w > S_{wr} \end{cases}$$

Gas relative permeability:

$$k_{r,g}(S_w) = \begin{cases} 0 & \text{if } S_g \leq S_{gr} \\ \frac{S_g - S_{gr}}{1 - S_{gr}} & \text{if } S_g > S_{gr} \end{cases}$$

where $S_g = 1 - S_w$ is the gas saturation.

Quadratic Model (KrModel = 1):

Water relative permeability:

$$k_{r,w}(S_w) = \begin{cases} 0 & \text{if } S_w \leq S_{wr} \\ \left(\frac{S_w - S_{wr}}{1 - S_{wr}}\right)^2 & \text{if } S_w > S_{wr} \end{cases}$$

Gas relative permeability:

$$k_{r,g}(S_w) = \begin{cases} 0 & \text{if } S_g \leq S_{gr} \\ \left(\frac{S_g - S_{gr}}{1 - S_{gr}}\right)^2 & \text{if } S_g > S_{gr} \end{cases}$$

where:

- $k_{r,w}(S_w)$ is the water relative permeability
- $k_{r,g}(S_w)$ is the gas relative permeability
- S_w is the water saturation
- $S_g = 1 - S_w$ is the gas saturation
- S_{wr} is the residual water saturation (Swr)
- S_{gr} is the residual gas saturation (Sgr)