subFlow Library Documentation

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

Generated Documentation

February 6, 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 $\mathbf{H}(\mathrm{div})$–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 $\mathbf{H}(\mathrm{div})$–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. For convenience, all its atributes are public.

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:

```
// Read problem data from JSON file
void ReadJSONFile(std::string filename);

// Serialization methods for saving/loading state
void Write(TPZStream &buf, int withclassid) const;
void Read(TPZStream &buf, void *context);
```

2.2 JSON File Structure Overview

subFlow uses JSON-based configuration files to define simulation parameters. This section describes the structure and available options. A complete configuration file contains six main sections:

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

2.3 Mesh Configuration

2.3.1 UseGMsh

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

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

2.3.2 MshFile

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

GMSH mesh file name (must be placed in the input directory). Required if `UseGMsh` is `true`.

2.3.3 Dimension

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

Problem spatial dimension (2D or 3D).

2.4 Domains Section

Defines regions within the reservoir with homogeneous properties.

```
"Domains": [
    {
        "name": "string",           // Name identifier
        "matid": integer,           // Unique material ID
        "K": float,                 // Permeability (absolute)
        "phi": float                // Porosity (0 to 1)
    },
    ...
]
```

Example:

```
"Domains": [
    {
        "name": "sandstone",
        "matid": 1,
        "K": 1.83e-5,
        "phi": 0.3
    }
]
```

2.5 Boundary Section

Defines boundary conditions on the domain edges/surfaces.

```
"Boundary": [
    {
        "name": "string",                   // Name identifier
        "matid": integer,                   // Unique material ID
        "type": integer,                    // 0: Dirichlet (pressure),
                                            // 1: Neumann (flux)
        "value": float,                     // BC value
        "functionID": integer,              // 0: Uses value,
```

```
                                                    // other: the ID of a user-
                                                       defined function in
                                                       TSFFunctionsGenerator.h
        "ExternalSaturation": float,                // Saturation for injection
        "SaturationFunctionID": integer             // 0: Uses
           ExternalSaturation,
                                                    // other: the ID of a user-
                                                       defined function in
                                                       TSFFunctionsGenerator.h
    },
    ...
]
```

Example:

```
"Boundary": [
    {
        "name": "inlet",
        "matid": 5,
        "type": 0,
        "value": 10.0,
        "functionID": 0,
        "ExternalSaturation": 1.0,
        "SaturationFunctionID": 0
    },
    {
        "name": "outlet",
        "matid": 3,
        "type": 0,
        "value": 0.0,
        "functionID": 0,
        "ExternalSaturation": 0.0,
        "SaturationFunctionID": 0
    }
]
```

2.6 Numerics Section

Controls the numerical solution method and parameters.

```
"Numerics": {
    "AnalysisType": integer,        // 0: Darcy only,
                                    // 1: Transport only,
                                    // 2: Coupled SFI
    "FluxOrder": integer,           // Order of Hdiv space (1 or 2)
    "DeltaT": float,                // Time step size
    "NSteps": integer,              // Number of time steps
    "Gravity": [gx, gy, gz],        // Gravity vector
    "IsAxisymmetric": boolean,      // Axisymmetric formulation
    "IsLinearTrace": boolean,       // Linear tracer (no saturation)
    "FourApproxSpaces": boolean,    // Use 4-space mixed formulation
    "NThreadsDarcy": integer,       // Threading for Darcy solver
    "MaxIterSFI": integer,          // Max SFI iterations
    "TolSFI": float,                // SFI convergence tolerance
    "MaxIterDarcy": integer,        // Max Newton iterations (Darcy)
    "ResTolDarcy": float,           // Residual tolerance (Darcy)
    "CorrTolDarcy": float,          // Correction tolerance (Darcy)
    "MaxIterTransport": integer,    // Max iterations (Transport)
```

```
    "ResTolTransport": float,      // Residual tolerance (Transport)
    "CorrTolTransport": float      // Correction tolerance (Transport)
}
```

Example:

```
"Numerics": {
    "AnalysisType": 2,
    "FluxOrder": 1,
    "DeltaT": 0.001,
    "NSteps": 100,
    "Gravity": [0.0, -9.81, 0.0],
    "IsAxisymmetric": false,
    "IsLinearTrace": false,
    "FourApproxSpaces": true,
    "NThreadsDarcy": 0,
    "MaxIterSFI": 5,
    "TolSFI": 1.0e-6,
    "MaxIterDarcy": 10,
    "ResTolDarcy": 1.0e-6,
    "CorrTolDarcy": 1.0e-6,
    "MaxIterTransport": 10,
    "ResTolTransport": 1.0e-6,
    "CorrTolTransport": 1.0e-6
}
```

2.7 Fluid Properties Section

Defines fluid characteristics for water and gas phases.

```
"FluidProperties": {
    "WaterDensity": float,          // Reference water density
    "WaterViscosity": float,        // Water dynamic viscosity
    "WaterCompressibility": float,  // Water compressibility
    "GasDensity": float,            // Reference gas density
    "GasViscosity": float,          // Gas dynamic viscosity
    "GasCompressibility": float,    // Gas compressibility
    "DensityModel": integer,        // 0: Linear, 1: Exponential
    "ReferencePressure": float      // Reference pressure for models
}
```

Example:

```
"FluidProperties": {
    "WaterDensity": 1000.0,
    "WaterViscosity": 0.001,
    "WaterCompressibility": 0.0,
    "GasDensity": 1.225,
    "GasViscosity": 1.81e-5,
    "GasCompressibility": 0.0,
    "DensityModel": 0,
    "ReferencePressure": 101325.0
}
```

2.8 Petro-Physics Section

Rock-fluid interaction parameters.

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

Example:

```
"PetroPhysics": {
    "KrModel": 1,
    "Swr": 0.1,
    "Sgr": 0.05
}
```

2.9 Reservoir Properties Section

Initial conditions for pressure and saturation fields.

```
"ReservoirProperties": {
    "s0": {
        "functionType": integer,  // 0: Constant, other: the ID of a
           user-defined function in TSFFunctionsGenerator.h
        "value": float            // Initial value
    },
    "p0": {
        "functionType": integer,  // 0: Constant, other: the ID of a
           user-defined function in TSFFunctionsGenerator.h
        "value": float            // Initial value
    }
}
```

Example:

```
"ReservoirProperties": {
    "s0": {
        "functionType": 0,
        "value": 0.2
    },
    "p0": {
        "functionType": 0,
        "value": 0.0
    }
}
```

2.10 Post-Processing Section

Controls output and visualization options.

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

Example:

```
"PostProcess": {
    "PostProcessFrequency": 1,
```

```
    "NThreads": 0,
    "VTKResolution": 0
}
```

2.11 TSFApproxCreator

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

Location: `src/TSFApproxCreator.h`

Responsibilities:

- Configure the Darcy finite element space ($\mathbf{H}$(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. By default, it uses TPZFastCondensedElement for fast matrix updating, but the user can choose to use standard static condensation by setting `-DUSE_FAST_CONDENSED_COMPEL=false` during CMake configuration.
- Build auxiliary transport mesh for post-processing and interface identification. This mesh is not used to assemble the transport problem.
- Insert interface elements between domain elements and boundary conditions for transport problem.

Key Methods:

```
// Set problem data reference
void SetProblemData(TSFProblemData *simData);

// Configure Darcy approximation space
void ConfigureDarcySpace();

// Add Darcy materials to the mesh
void AddDarcyMaterials();

// Create complete approximation space (main driver)
TPZMultiphysicsCompMesh *CreateApproximationSpace();

// Build auxiliary transport mesh
void BuildTransportCmesh();
```

2.12 TPZFastCondensedElement

Purpose: Implements static condensation of high-order element degrees of freedom to reduce the system size. The assembly is performed only once, and the matrix updating is done purely in an algebraic way.

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
- Update condensed matrices algebraically during nonlinear iterations.

Key Methods:

```
// Computes the element stifness matrix and right hand side
void CalcStiff(TPZElementMatrixT<STATE> &ek,TPZElementMatrixT<STATE> &
    ef);
```

Implements an efficient algebraic matrix update strategy for nonlinear multiphase flow problems. Instead of reassembling the element matrix from scratch at each Newton iteration, it reuses a precomputed reference matrix and applies selective algebraic coefficient modifications. This achieves $O(1)$ complexity per iteration for matrix assembly.

Reference Matrix Caching (First Assembly) On the first call (when `fMatrixComputed == false`), the method:

- Assembles the full condensed element stiffness matrix via the parent class: `TPZCondensedCompElT::CalcS fEF)`.
- Stores the reference matrix and RHS: `fEK`, `fEF`.
- Computes and caches body force contributions for pressure equations.
- Sets `fMatrixComputed = true` to skip reassembly in subsequent calls.

Selective Coefficient Scaling The method copies the reference matrices: `ek = fEK`, `ef = fEF`, and applies selective scaling to the stiffness matrix to account for changes in fluid properties:

- Interior (flux) rows/columns: multiplied by $\frac{1}{\lambda}$ (reducing coupling strength when $\lambda > 1$).
- Last row/column (pressure equation): multiplied by λ (maintaining coupling consistency).

This asymmetric scaling keeps the mixed system well-posed while controlling the pressure–flux coupling strength through λ (stored in `fLambda`).

Solution Vector Extraction The current solution vector is extracted from the computational mesh via `GetSolutionVector()`, which gathers values from all connected pressure and flux degrees of freedom.

RHS Modification for Density Variation The RHS is scaled by the mixture density factor:

$$\text{ef} \leftarrow \text{ef} \times G_\lambda$$

(stored in `fMixedDensity`). This accounts for changes in the effective density:

$$G_\lambda = \rho_w f_w + \rho_g f_g$$

as water and gas saturation vary. A compressibility contribution is added to the average pressure equation (last row):

$$\text{ef}[nrows - 1] \mathrel{+}= \texttt{fCompressibilityRhsTerm}.$$

where `fCompressibilityRhsTerm` stores $\int_{\Omega_e} \frac{\phi}{\Delta t} \left[(\frac{s_w^{n+1}}{b_w^{n+1}} + \frac{s_g^{n+1}}{b_g^{n+1}}) - (\frac{s_w^n}{b_w^n} + \frac{s_g^n}{b_g^n}) \right] d\Omega_e$.

Residual Computation (Nonlinear Effects) The residual contribution from the current solution is computed as:

$$\text{ef} \leftarrow \text{ef} - \mathbf{K} \cdot \vec{u}$$

This residual incorporates nonlinear effects of the current pressure and flux state, which is essential for Newton iteration convergence.

Compressibility Matrix Term Finally, the last diagonal entry is set:

$$\mathbf{K}[nrows - 1, ncols - 1] = \texttt{fCompressibilityMatrixTerm}.$$

where $\texttt{fCompressibilityMatrixTerm} = \int_{\Omega_e} \frac{\phi}{\Delta t} \left(\frac{s_w^{n+1}}{\frac{d(b_w^{n+1})}{dp}} + \frac{s_g^{n+1}}{\frac{d(b_g^{n+1})}{dp}} \right) d\Omega_e$, with

$$\frac{d(b_\alpha)}{dp} = c_\alpha \rho_\alpha^{\text{ref}},$$

where c_α is the compressibility modulus of phase α.

3 Analysis Classes

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

3.1 TSFDarcyAnalysis

Purpose: Solves the Darcy problem for pressure and flux using the $\mathbf{H}(\mathrm{div})$–L^2 mixed finite element formulation. By default, it uses `TPZSSpStructMatrix` for efficient sparse matrix assembly and `Intel Pardiso` direct solver if Neopz was compiled with `-DUSE_MKL`. If MKL is not available, we are in the process of making `MUMPS` an option.

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). The first one is applied in a weak manner in the rhs, while the second one is applied via `TPZEquationFilter`, where the Neumann equations are filtered out and the corresponding flux values are set in the solution vector.
- Track number of iterations and timing information
- Post-process flux and pressure solutions

Key Methods:

```
// Initialization with problem data
void Initialize();
void SetProblemData(TSFProblemData *simData);
TSFProblemData *GetProblemData();

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

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

// System assembly and solving
void Assemble() override;
void Solve() override;
```

3.2 TSFTransportAnalysis

Purpose: Solves the saturation transport problem using a Finite Volume scheme with upwinding. This can be seen as an "Interface" class in the sense that it performs the nonlinear iterations, but the assembly of the transport problem is done in the class `TSFAlgebraicTransport`, which is called inside the method `Assemble()` of this class.

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:

```
// Initialization
void Initialize();
void SetProblemData(TSFProblemData *simData);
TSFProblemData *GetProblemData();

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

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

// Matrix assembly
void AssembleMass();         // Assemble mass matrix (once)
void Assemble() override;    // Assemble transmissibility and RHS

// Solving
void Solve() override;

// Coefficient updates
void UpdateDensityAndCoefficients();

// Verification
void VerifyElementFluxes();
```

Data Members:

```
TSFProblemData *fSimData;
TSFAlgebraicTransport fAlgebraicTransport;
int fKiteration;
bool fIsFirstAssemble;
// Sparse matrices for efficiency
TPZFYsmpMatrix<REAL> *fTransmissibilityMatrix;
TPZFMatrix<REAL> fMassMatrix;
```

3.3 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:

```
// Initialization and setup
void SetProblemData(TSFProblemData *simData);
void Initialize();

// Run complete simulation
void Run(std::ostream &out = std::cout) override;

// Run single time step with SFI iterations
void RunTimeStep(std::ostream &out = std::cout);

// Data transfer
void TransferDarcyToTransport();   // Transfer flux to transport mesh
void TransferTransportToDarcy();   // Transfer saturation to Darcy

// State management
void UpdateLastStateVariables();

// Post-processing
void PostProcessTimeStep(const int type, const int dim,
                         int step = -1);
```

Data Members:

```
TSFProblemData *fSimData;
int fKiteration;
bool fShouldSolveDarcy;        // Solve Darcy only once for linear
   tracer
TSFDarcyAnalysis fDarcyAnalysis;
TSFTransportAnalysis fTransportAnalysis;
TSFDataTransfer fDataTransfer;
TPZFMatrix<STATE> fDarcySolution;
TPZFMatrix<STATE> fTransportSolution;
```

4 Material Classes

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

4.1 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:

```
// Volumetric contributions to weak form
void Contribute(const TPZVec<TPZMaterialDataT<STATE>> &datavec,
                REAL weight, TPZFMatrix<STATE> &ek,
                TPZFMatrix<STATE> &ef) override;

// Additional space contributions (for 4-space formulation)
void ContributeFourSpaces(const TPZVec<TPZMaterialDataT<STATE>> &
    datavec,
                          REAL weight, TPZFMatrix<STATE> &ek,
                          TPZFMatrix<STATE> &ef);

// Boundary condition contributions
void ContributeBC(const TPZVec<TPZMaterialDataT<STATE>> &datavec,
                  REAL weight, TPZFMatrix<STATE> &ek,
                  TPZFMatrix<STATE> &ef,
                  TPZBndCondT<STATE> &bc) override;

// Data requirements specification
void FillDataRequirements(
    TPZVec<TPZMaterialDataT<STATE>> &datavec) const override;
void FillBoundaryConditionDataRequirements(
    int type, TPZVec<TPZMaterialDataT<STATE>> &datavec) const override;

// Physics control methods
void SetAxisymmetry(bool IsAxisymmetric);
bool IsAxisymmetric() const;
void SetGravity(const TPZFNMatrix<3, REAL> &gravity);
const TPZFNMatrix<3, REAL> &GetGravity() const;
void SetFourSpaces(bool fourSpaces);
bool IsFourSpaces() const;

// Post-processing
void Solution(const TPZVec<TPZMaterialDataT<STATE>> &datavec,
              int var, TPZVec<REAL> &Solout);
```

4.2 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:

```
// Data requirements
void FillDataRequirements(
    TPZVec<TPZMaterialDataT<STATE>> &datavec) const override;
void FillBoundaryConditionDataRequirements(
    int type, TPZVec<TPZMaterialDataT<STATE>> &datavec) const override;
void FillDataRequirementsInterface(
    TPZMaterialDataT<STATE> &data) const override;

// Physics contributions
void Contribute(const TPZVec<TPZMaterialDataT<STATE>> &datavec,
                REAL weight, TPZFMatrix<STATE> &ek,
                TPZFMatrix<STATE> &ef) override;
void ContributeBC(const TPZVec<TPZMaterialDataT<STATE>> &datavec,
                  REAL weight, TPZFMatrix<STATE> &ek,
                  TPZFMatrix<STATE> &ef,
                  TPZBndCondT<STATE> &bc) override;

// Post-processing
int VariableIndex(const std::string &name) const override;
int NSolutionVariables(int var) const override;
void Solution(const TPZVec<TPZMaterialDataT<STATE>> &datavec,
              int var, TPZVec<REAL> &Solout) override;

// Property accessors
int Dimension() const;
void SetDimension(int dim);
int NStateVariables() const;
void Print(std::ostream &out) const override;
```

Data Members:

```
int m_dimension;            // Problem dimension
int m_mat_id;               // Material ID
bool m_mass_matrix_Q;       // Mass matrix assembly flag
REAL m_dt;                  // Time step size
REAL m_phi;                 // Porosity
REAL m_fracture_epsilon;    // Fracture parameter
```

5 Data Transfer and Utility Classes

5.1 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

5.2 TSFAlgebraicTransport

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

Location: `src/TSFAlgebraicTransport.h`

Responsibilities:

`TFromDarcyToTransport` `TFromDarcyToTransport` `TFromDarcyToTransport`

- 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

5.3 TSFSavable

Purpose: Base class providing serialization capabilities for subFlow objects.

Location: `src/TSFSavable.h`

Responsibilities:

`TCellDataProvider`

- Provide save/load infrastructure for simulation state
- Support checkpoint and restart capabilities
- Enable long-term data persistence

6 Class Hierarchy Diagram

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

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

TPZHDivApproxCreator
    +-- TSFApproxCreator

TPZMixedDarcyFlow
    +-- TSFMixedDarcy

TPZMatBase (with multiple inheritance)
    +-- TSFTransportMaterial

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

7 Workflow and Data Flow Diagram

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

7.1 Complete Simulation Workflow

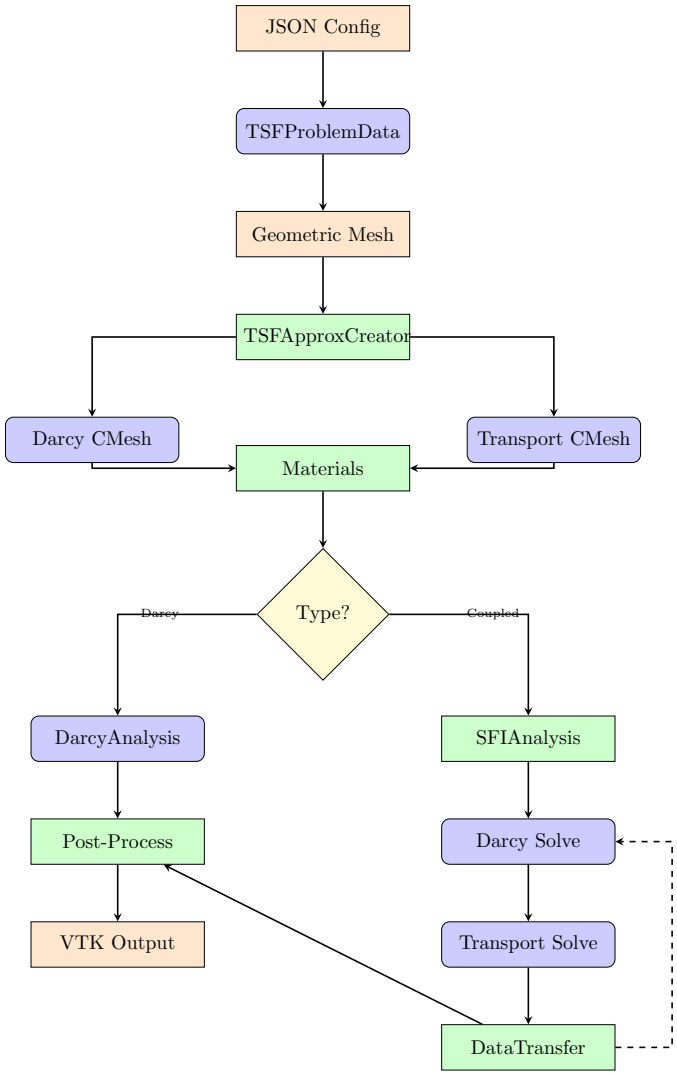

Figure 1: Complete workflow for multiphase flow simulation with subFlow

7.2 Data Transfer and Coupling Mechanism

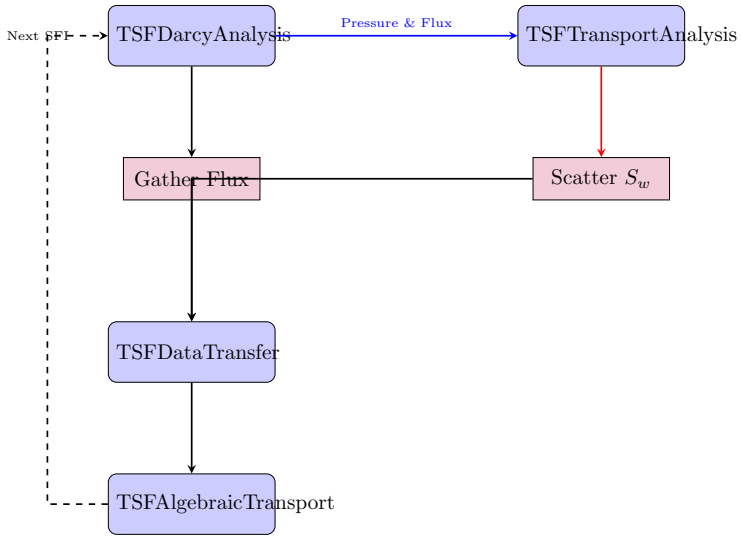

Figure 2: Data transfer mechanism in SFI coupling

8 Typical Usage Workflow

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

8.1 Basic Simulation Steps

```
// 1. Load problem data from JSON configuration
TSFProblemData simData;
simData.ReadJSONFile("simulation_config.json");

// 2. Build or load geometric mesh
TPZGeoMesh *gmesh = new TPZGeoMesh();
// ... populate gmesh from GMSH file or generated mesh ...

// 3. Create approximation spaces
TSFApproxCreator approxCreator(gmesh);
approxCreator.SetProblemData(&simData);
approxCreator.ConfigureDarcySpace();
approxCreator.AddDarcyMaterials();
TPZMultiphysicsCompMesh *darcy_cmesh =
    approxCreator.CreateApproximationSpace();

// 4. Choose analysis type based on configuration
if (simData.fTNumerics.fAnalysisType == 0) {
    // Darcy problem only
    TSFDarcyAnalysis darcyAnalysis(darcy_cmesh);
    darcyAnalysis.SetProblemData(&simData);
    darcyAnalysis.Initialize();
    darcyAnalysis.RunTimeStep();
    darcyAnalysis.PostProcessTimeStep(
        gmesh->Dimension(), 0);
} else {
    // Coupled Darcy-Transport problem
    approxCreator.BuildTransportCmesh();
    TPZCompMesh *transport_cmesh =
        approxCreator.GetTransportCmesh();

```

```
    // Create coupled SFI solver
    TSFSFIAnalysis sfiAnalysis(
        darcy_cmesh, transport_cmesh);
    sfiAnalysis.SetProblemData(&simData);
    sfiAnalysis.Initialize();
    sfiAnalysis.Run();  // Run complete simulation

    delete transport_cmesh;
}

// 5. Cleanup
delete darcy_cmesh;
delete gmesh;
```

8.2 Configuration File Example

A minimal JSON configuration for a coupled 2D simulation:

```
{
    "UseGMsh": true,
    "MshFile": "reservoir_2d.msh",
    "Dimension": 2,
    "Domains": [{
        "name": "reservoir",
        "matid": 1,
        "K": 1e-4,
        "phi": 0.25
    }],
    "Boundary": [
        {
            "name": "inlet",
            "matid": 10,
            "type": 0,
            "value": 100.0,
            "functionID": 0,
            "ExternalSaturation": 0.8,
            "SaturationFunctionID": 0
        },
        {
            "name": "outlet",
            "matid": 11,
            "type": 0,
            "value": 0.0,
            "functionID": 0,
            "ExternalSaturation": 0.0,
            "SaturationFunctionID": 0
        }
    ],
    "Numerics": {
        "AnalysisType": 2,
        "FluxOrder": 1,
        "DeltaT": 0.01,
        "NSteps": 50,
        "Gravity": [0.0, -9.81, 0.0],
        "IsAxisymmetric": false,
        "IsLinearTrace": false,
        "FourApproxSpaces": true,
        "NThreadsDarcy": 0,
```

```
        "MaxIterSFI": 5,
        "TolSFI": 1e-6,
        "MaxIterDarcy": 10,
        "ResTolDarcy": 1e-6,
        "CorrTolDarcy": 1e-6,
        "MaxIterTransport": 10,
        "ResTolTransport": 1e-6,
        "CorrTolTransport": 1e-6
    },
    "FluidProperties": {
        "WaterDensity": 1000.0,
        "WaterViscosity": 1e-3,
        "WaterCompressibility": 0.0,
        "GasDensity": 1.0,
        "GasViscosity": 1e-5,
        "GasCompressibility": 0.0,
        "DensityModel": 0,
        "ReferencePressure": 0.0
    },
    "PetroPhysics": {
        "KrModel": 0,
        "Swr": 0.0,
        "Sgr": 0.0
    },
    "ReservoirProperties": {
        "s0": {"functionType": 0, "value": 0.0},
        "p0": {"functionType": 0, "value": 0.0}
    },
    "PostProcess": {
        "PostProcessFrequency": 1,
        "NThreads": 0,
        "VTKResolution": 0
    }
}
```

9 Post-Processing and Output

9.1 VTK Output

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

9.1.1 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

9.1.2 Post-Processing Frequency

Control output frequency via the JSON configuration:

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

9.2 Solution Variables

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

9.2.1 Darcy Problem Variables

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

9.2.2 Transport Problem Variables

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

10 Advanced Features

10.1 Axisymmetric Formulations

For cylindrical symmetry problems, enable axisymmetric mode:

```
"Numerics": {
    "IsAxisymmetric": true,
    ...
}
```

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

10.2 Four-Space Mixed Formulation

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

```
"Numerics": {
    "FourApproxSpaces": true,
    ...
}
```

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

10.3 Compressible Fluids

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

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

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):

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

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):

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

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

10.4 Relative Permeability Models

Control the phase relative permeability computation:

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

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)