Introdution to LifeV

Davide Baroli
LifeV-admin: Luca Paglieri

Politecnico di Milano
Laboratory for Modeling and Scientific Computing

29-01-2015

LifeV in Action

- Why LifeV?
- Distribuite Communication
- Mesh
- Linear Algebra
  Matrix, Vector, distributor of elements
- Finite Element Space
- ETA: Expression Template Assembler
- Linear Solver
- Post-Processing: Exporter/Importer

LifeV Project

LifeV is a open source Finite Element Library for the solution of PDEs developed at MOX, CMCS (EPFL), E(CM)^2 (Emory) and Estime (INRIA).

- Object oriented, c++ code.
- Trilinos Epetra backend for distribuite parallel computation.
- Research code oriented to the development novel numerical methods and algorithms.
- Aim: Solve large-scale, complex multi-physics engineering and scientific problems.
- Scalable library on HPC.
- Licensed under the GNU PBS.
- Release Versions are hosted at public repository `https://github.com/lifev/lifev`
- Google groups for mailing lists (lifev-user and lifev-dev)

Why LifeV?

Original features of LifeV:

- Type of Mesh: Hexaedra, Tetraedra (3D), Quad and Triangle (2D), Line (1D)
- Internal mesh generator for structured hypercube of dimension 1,2,3.
- Supporting external mesh format: .mesh (MEDIT), .msh (GMSH), .vol (NETGEN), m++
- Different Mesh Partitioner: Parmetis,Zoltan (Trilinos' package); offline-online partitioner to speed up the computation on HPC.
- Parallel I/O based on hdf5 format: internal wrapper of HDF5 library and EpetraExt Trilinos' HDF5 interface.
- Parser: GetPot for datafile and Teuchos (Trilinos) for XML.

- Construction of finite element assembly based on Epetra_FECrsMatrix, Epetra_FEVector and Epetra_Map
- Assembly of Finite Element based on DSEL principle: ETA Expression Template Assembly
- Internal Extension of Epetra Linear Algebra to assembly block matrix and block vector: "MatrixEpetraStructured" and "VectorEpetraStructured"
- Matrix-free methods are derived from Epetra_Operator
  1. LinearOperator
  2. SolverOperator(interface of an Invertible Linear Operator)
  3. ConfinedOperator (interface for restriction to a block part of Linear Block Operator)
- Preconditioner and Linear Solver:
  1. Internal interface for block preconditioner and multiplicative composition of preconditioner: SIMPLE,PCD and Yoshida.
  2. Wrapper of Ifpack (one-level Additive Schwarz Preconditioner), ML (Algebraic Multigrid),Teko (Block Preconditioner- LSC),
  3. Wrapper of Aztec00 (GMRES, CG,BICGSTAB) and Belos (Block GMRES) (Iterative solver)
  4. Wrapper of Anasazi (Eigenvalue) and Amesos (direct linear solver).

Algorithms and Modelling:

1. Geometric Multiscale Method
2. Level-Set
3. Monolithic and segregated algorithms for Fluid-Structure Interaction
4. Elettrophysiology for cardiovascular system
5. Structure: Linear and NonLinear Elasticity
6. Fluid: Navier-Stokes
7. Porous Media: Darcy, Non-Darcy and Hyperbolic Problem

Ongoing Research about novel discretizations, methods and models hosted at `https://cmcsforge.epfl.ch/projects/lifev`

1. Variational MultiScale
2. eXtended Finite Element Method
3. IsoGeometric Analysis on Surface (interface LifeV-LibIGA) and Structural problem (interface FEAP-LifeV)
4. Hierarchical Model reduction
5. Data Assimilation
6. Turbolence Models

Type of LifeV

Firstly in a own user/developed code in LifeV library, we need to include in header

```
#include <lifev/core/LifeV.hpp>
```

"LifeV.hpp" defines a number of types that are used in the library. The most used (in the tutorial) are

```
//uint32_type a 32 bit unsigned integer
typedef uint32_type UInt;
//! IDs
typedef uint32_type ID;
typedef double Real;
```

The Epetra Communicator Objects which encapsulates the general information and services needed to run on serial or parallel our computation.

```
#include <Epetra_ConfigDefs.h>
#ifdef EPETRA_MPI
#include <mpi.h>
#include <Epetra_MpiComm.h>
#else
#include <Epetra_SerialComm.h>
#endif
```

Listing 1: Header

```
int main ( int argc, char** argv )
{
#ifdef HAVE_MPI
    MPI_Init (&argc, &argv);
//For MPI distributed memory executions
    boost::shared_ptr<Epetra_Comm> Comm (new Epetra_MpiComm (MPI_COMM_WORLD) );
#else
//For serial executions
    boost::shared_ptr<Epetra_Comm> Comm (new Epetra_SerialComm);
#endif
  Int NumProc = Comm->NumProc();
  Int MyPID   = Comm->MyPID();
// ... other code follows ...
#ifdef HAVE_MPI
    MPI_Finalize();
#endif
return 0;
}
```

Mesh: 1D

Create Structured Mesh

```
#include<lifev/core/mesh/RegionMesh1DStructured.hpp>
 typedef RegionMesh<LinearLine> mesh_Type;
 meshPtr_Type FullMeshPtr(new mesh_Type(Comm));
// MarkerID=0; Domain=[-1,2] with 10 Elements along x
 regularMesh1D (*FullMeshPtr, 0, 8, 3, -1);
```

Mesh: 2D

Create Structured Mesh

```
#include <lifev/core/mesh/RegionMesh2DStructured.hpp>
 typedef RegionMesh<LinearTriangle> mesh_Type;
    meshPtr_Type FullMeshPtr ( new mesh_Type ( Comm ) );
// MarkerID=0; Domain=[-1,2]x[-1,3] with 8 Elements along x
    and 11 Elements along y
    regularMesh2D ( *FullMeshPtr, 0, 8, 11, 2.,3.,-1.,-1. );
```

Mesh: 3D

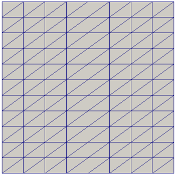
Create Structured Mesh

```
#include <lifev/core/mesh/RegionMesh3DStructured.hpp>
typedef RegionMesh<LinearTetra> mesh_Type;
    meshPtr_Type FullMeshPtr ( new mesh_Type ( Comm ) );
// MarkerID=0; Domain=[-1,1]x[-1,1]x[-1,1] with 10 Elements
    along x; 11 Elements along y;  12 Elements along y
    regularMesh3D ( *FullMeshPtr, 0, 10, 11,12, 1.,1.,1.,
    -1.,-1.,-1 );
```

Declaration of the Method

```cpp
template <typename GeoShape, typename MC >
void regularMesh3D ( RegionMesh<GeoShape,MC >&
    mesh,markerID_Type regionFlag,
                    const UInt& m_x,
                    const UInt& m_y,
                    const UInt& m_z,
                    bool verbose = false,
                    const Real& l_x = 1.0,
                    const Real& l_y = 1.0,
                    const Real& l_z = 1.0,
                    const Real& t_x = 0.0,
                    const Real& t_y = 0.0,
                    const Real& t_z = 0.0
                )
```

Geoshape: LinearTriangle, LinearTetra (see ElementShape.hpp)
MC define the policy for the entityFlag treatment



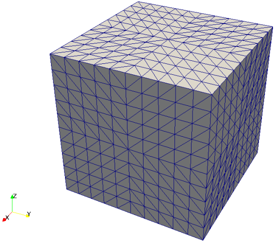


Figure: Triangular Mesh(left) and Tetrahedra Mesh (right)

For 2d Mesh, the Marker(Label) are

- for Edge are: 1 (BOTTOM), 2(LEFT), 3(TOP),4(RIGHT)
- for Corner are: 5(BOTTOM_RIGHT), 6(TOP_RIGHT),7(TOP_LEFT), 8(BOTTOM_LEFT)

For 3d Mesh, see "RegionMesh3DStructured.hpp".

Mesh Handler in LifeV

The mesh is build up over the following base properties:

- Markers: label associated to any entity and has a partial order relation. The markers are: point_marker,edge_marker,face_marker,volume_marker.
- Geometric Entities (GeoShape): Line, Triangle, Quad; their properties (member) are S_dimension, S_numFaces,S_numEdges,S_numVertices,S_shape, S_geometry
- Geometric Entities to define Finite Element: LinearTriangle, QuadTriangle whose properties are: S_numPoints (number of DOF),S_numPointsPerVertex ...
- Basic Entities of Mesh ("MeshEntity.hpp“), whose properties are: identifiers (localId and globalId) and flags to specify geometrical properties (PHYSICAL_BOUNDARY, SUBDOMAIN_INTERFACE, INTERNAL_INTERFACE)

In addition it is provided operation over the Mesh ("meshElementBare.hpp”) , mesh Utility ("MeshUtility.hpp“) and MeshCheck.hpp, which it is useful when we read external mesh. Finally it also defined the Container ("MeshEntityContainer.hpp”) for mesh entity.

Change Marker ID according to a function.

```
#include <lifev/core/mesh/MeshUtility.hpp>
#include <lifev/core/array/VectorSmall.hpp>
//Defining a function (outside main)
UInt colour_fun ( const VectorSmall<3>& barycentre )
{
    if ( barycentre[0] < 0.5 && barycentre[1] < 0.5 )
    return 2;
    return 3;
}
MeshUtility::assignRegionMarkerID ( *FullMeshPtr,
    colour_fun );
// How many element have markerID =2 ?
const UInt ElementID2=
    FullMeshPtr->elementList().countElementsWithMarkerID (
    2, std::equal_to<markerID_Type>() );
```

Extract Extract elements and facets from a mesh based on a functor.

```
#include <lifev/core/mesh/MeshEntityContainer.hpp>
```

We need to define the "Predicates“ (functors) whose templates are:

1. MeshEntity
2. Comparison Policy which must be a function able to compare two flag_Type.

The main method of Predicate is

```
 bool operator()(const MeshEntity & entity)const
```

Example is provided in core/testsuite/mesh ” entity_selection.cpp“.

The format supported by LifeV are: Mesh++, INRIA mesh, GMSH (.mesh), NETGEN (.vol) and MEDIT (.msh).

```
#include <lifev/core/filter/GetPot.hpp>
#include <lifev/core/mesh/MeshData.hpp>
// Read From dataFile
   GetPot command_line (argc, argv);
    const std::string data_file_name =
    command_line.follow ("data", 2, "-f", "--file");
    GetPot dataFile (data_file_name);
 const std::string discretization_section = "mesh";
 MeshData meshData (dataFile,
    (discretization_section).c_str() );
    readMesh (*fullMeshPtr, meshData);
```

Online Mesh Partitioner

```cpp
#include <lifev/core/mesh/MeshPartitioner.hpp>
    boost::shared_ptr<mesh_Type> meshPtr;
    meshPtr localMesh;
    {
        MeshPartitioner< mesh_Type >    meshPart;
        meshPart.setPartitionOverlap ( numOverlap );
        meshPart.doPartition ( fullMeshPtr, Comm );
        localMeshPtr = meshPart.meshPartition();
    }
    fullMeshPtr.reset();
```

Offline-Online Mesh Partitioner

Using HDF5IO.hpp and MeshPartitionerTool.hpp it is possible to divide-et-impera the partitioning in offline(rootprocessor-writer)-online(parallel-read). It is possible to choose:

- MeshPartioner
- Parmetis
- Zoltan

Detail TestCase can be found in "core/testsuite/offline_partition_io".

LifeV Linear Algebra: EpetraMap

EpetraMap manage the distribution of elements of matrices or vectors on a parallel machine (wrapper of Map_Epetra of Trilinos).

```cpp
#include <lifev/core/array/MapEpetra.hpp>
MapEpetra ( Int  numGlobalElements, // Num global Elements
            Int  numMyElements,    // Num local Elements
            Int* myGlobalElements, // Array of Id of local element
            const comm_ptrtype& commPtr ); // Pointer to Communicator
 MapEpetra ( mapData_Type,  // LifeV::Unique or LifeV::Repeated
 const comm_ptrtype& commPtr ); // Pointer to Communicator
```

Repeated is used during assembly, while Unique structured is used for post-processing

Operations: Addition, Assign and Juxtaposition for MapVector

LifeV Linear Algebra:EpetraMatrix

EpetraMatrix manage the Matrix (wrapper of Epetra_FECrsMatrix)

```
#include <lifev/core/array/MatrixEpetra.hpp>
MatrixEpetra ( const MapEpetra& domainmap,
               const MapEpetra& rangemap,
               Int numEntries = 50,
               bool ignoreNonLocalValues = false );
MatrixEpetra ( const MapEpetra& map, const
    Epetra_FECrsGraph& graph, bool ignoreNonLocalValues =
    false); // Graph contains the non-zeros pattern
```

Operations: Addition, Subtraction, vector-matrix multiplication
Spy (open in matlab),save in hdf5
To fill-in matrix:setCoefficient, addToCoefficients and sumIntoCoefficients
Close and re-open matrix:openCrsMatrix(), globalAssemble()

LifeV Linear Algebra:EpetraVector

EpetraVector manage the Matrix (wrapper of Epetra_FEVector)

```
#include <lifev/core/array/VectorEpetra.hpp>
explicit VectorEpetra ( const MapEpetra& map,
                        const MapEpetraType& mapType=Unique,
                        const combineMode_Type combineMode = Add );
VectorEpetra& subset ( const VectorEpetra& vector,
                       const MapEpetra& map,
                       const UInt offset1,
                       const UInt offset2 );
```

Operations: Addition, Subtraction, division, comparison
To fill-in vector: setCoefficient, add, replace, sumIntoGlobalValues
Close vector, globalAssemble(), and calculating norm norm1(),. . .

ETFESpace defines a templete version of Finite Element Space.

```
template<typename MeshType, typename MapType, UInt
    SpaceDim, UInt FieldDim>
ETFESpace<MeshType, MapType, SpaceDim, FieldDim>
(const meshPtr_Type& mesh, const ReferenceFE* refFE,const
    GeometricMap* geoMap, commPtr_Type& commptr)
```

If GeometricMap is guessed in the constructor, it is used the shape of the element of the mesh. Any information about the quadrature can be recover using QuadratureRuleProvider.hpp or "FESpace.hpp' (using method "qr()")

```
QuadratureRuleProvider::provideExactness (TETRA, 0)
```

Finite Element Space: Scalar and Vector Field

```
#include <lifev/eta/fem/ETFESpace.hpp>
```

```
boost::shared_ptr<ETFESpace< mesh_Type, MapEpetra, 3, 1 > >
    ETuSpace
( new ETFESpace< mesh_Type, MapEpetra, 3, 1 >
    (localMeshPtr, &feTetraP1, Comm) );
```

```
boost::shared_ptr<ETFESpace< mesh_Type, MapEpetra, 3, 3 > >
    ETuSpace
    ( new ETFESpace< mesh_Type, MapEpetra, 3,3  >
    (localMeshPtr, &feTetraP1, Comm) );
```

Defining before an FESpace:

```
#include <lifev/core/fem/FESpace.hpp>
 std::string uOrder ("P1");
boost::shared_ptr<FESpace< mesh_Type, MapEpetra > > uSpace
 ( new FESpace< mesh_Type, MapEpetra > (meshPtr, uOrder, 3, Comm) );
 boost::shared_ptr<ETFESpace< mesh_Type, MapEpetra, 3, 3 > > ETuSpace
 ( new ETFESpace< mesh_Type, MapEpetra, 3, 3 > (meshPtr, & (uSpace->refFE() ), & (uSpace->fe().geoMap() ), Comm) );
std::cout<<"Total Dof"<<ETuSpace->dof().numTotalDof()<<std::endl;

```

System Matrix

```
#include <lifev/core/array/MatrixEpetra.hpp>
typedef MatrixEpetra<Real> matrix_Type;
boost::shared_ptr<matrix_Type> systemMatrix (new
    matrix_Type ( ETuSpace->map() ) );
*systemMatrix *= 0.0;
{" fill in the matrix by Expression Template Assembler"}
systemMatrix->globalAssemble();
```

Let a domain Ω with a partion of $\partial\Omega = \Gamma_D \cup \Gamma_N \cup \Gamma_R$.

$$
\begin{aligned}
-\nabla \cdot (K \nabla u) + a_R u &= f \text{in } \Omega \\
u &= g_D \text{on } \Gamma_D \\
\frac{\partial u}{\partial n} &= g_N \text{on } \Gamma_N \\
\alpha u + \frac{\partial u}{\partial n} &= g_R \text{on } \Gamma_R
\end{aligned}
\tag{1}
$$

Find $u \in H^1_{\Gamma_D}(\Omega)$ such that

$$
\int_\Omega K \nabla u \nabla v + \int_\Omega a_R u + \alpha \int_{\Gamma_R} uv = \int_\Omega fv + \int_{\Gamma_N} g_N v + \int_{\Gamma_R} g_R v \quad \forall v \in H^1_0(\Omega)
\tag{2}
$$

See “lifev/eta/examples/diffusionReaction/main.cpp”

Parameter of Problem

```
Real scalardiffusion ( const Real& /*t*/, const Real& x , const Real& y, const Real& z , const ID& /*i*/)
{
    return 1.; // std::sin (2* pi / y ) * std::cos ( 2*pi / x ) * std::exp ( z ) ;
}
```

```
class scalarDiffusionFunctor
{
public:
    typedef Real return_Type;

    return_Type operator() ( const VectorSmall<3> spaceCoordinates )
    {
        return scalardiffusion (0, spaceCoordinates[0], spaceCoordinates[1], spaceCoordinates[2] , 0 ) ;
    }

    scalarDiffusionFunctor() {}
    scalarDiffusionFunctor (const scalarDiffusionFunctor&) {}
    ~scalarDiffusionFunctor() {}
};
```

```
boost::shared_ptr<scalarDiffusionFunctor>
    scalarDiffusionFct ( new scalarDiffusionFunctor );
 {using namespace ExpressionAssembly;
integrate (  elements (ETuFESpace->mesh() ),
                    quadRuleTetra4pt,
                    ETuFESpace,
                    ETuFESpace,
                    eval ( scalarDiffusionFct, X) * dot (
    grad (phi_j) , grad (phi_i) )
                   - eval ( scalarReactionFct, X) *
    phi_j * phi_i

                 )
              >> *SystemMatrix;
 }
```

```
{ using namespace ExpressionAssembly;
        integrate ( elements (ETuFESpace->mesh() ), // Mesh


                    uFESpace->qr(), // QR

                    ETuFESpace,

        eval ( ScalarFctRhs, X ) * phi_i

                 )
              >> uRhs;
}
```

Impose Robin Boundary Condition:LHS

```
QuadratureBoundary myBDQR (buildTetraBDQR (quadRuleTria4pt) );
integrate ( boundary (ETuFESpace->mesh(), BCFlags::BOTTOMWALL ),
            myBDQR,
            ETuFESpace,
            ETuFESpace,
            value ( alpha ) * phi_j * phi_i

          )
          >> *SystemMatrix;
```

Impose Robin Boundary Condition:RHS

```
integrate ( boundary (ETuFESpace->mesh(),
    BCFlags::BOTTOMWALL), // Mesh

                       myBDQR, // QR

                       ETuFESpace,

                       eval ( ScalarFctR , X ) * phi_i

                   )
               >> uRhs;
```

```
BCHandler bcHandler;
 BCFunctionBase dirichletBCFct ( dirichlet );
 bcHandler.addBC ("Left", BCFlags::LEFTWALL, Essential,
 Full, dirichletBCFct, 1);
 bcHandler.addBC ("Right", BCFlags::RIGHTWALL,
 Essential, Full , dirichletBCFct, 1);
 bcHandler.addBC ("Back", BCFlags::BACKWALL, Essential,
 Full, dirichletBCFct, 1);
 bcHandler.bcUpdate ( *meshPtr, uFESpace->feBd(),
 uFESpace->dof() );
 bcManage (*SystemMatrix, *uRhsUnique,
           *uFESpace->mesh(), uFESpace->dof(),
           bcHandler, uFESpace->feBd(), 1.0, Real (0.0)
 );
```

Setting the Algebraic System Solver

```
prec_Type* precRawPtr;
basePrecPtr_Type precPtr;
precRawPtr = new prec_Type;
precRawPtr->setDataFromGetPot ( dataFile, "prec" );
precPtr.reset ( precRawPtr );
Teuchos::RCP< Teuchos::ParameterList > solverList = Teuchos::rcp ( new Teuchos::ParameterList );
const std::string solverParam = dataFile("solver/listName", "SolverParamListBelos.xml");
solverList = Teuchos::getParametersFromXmlFile(solverParam );
LinearSolver linearSolver;
linearSolver.setCommunicator ( Comm );
linearSolver.setParameters ( *solverList );
linearSolver.setPreconditioner ( precPtr );
```

```
vectorPtr_Type uSolution ( new vector_Type ( ETuFESpace->map() , Unique) );
linearSolver.setOperator ( SystemMatrix );
linearSolver.setRightHandSide ( uRhsUnique );
linearSolver.solve ( uSolution );
```

Exporter in HDF5

```
std::string const exporterFileName    =  dataFile (
"exporter/filename", "cube");
ExporterHDF5<mesh_Type> exporter ( dataFile, meshPtr,
exporterFileName, Comm->MyPID() );
exporter.setMultimesh (false);
boost::shared_ptr<vector_Type> uExported ( new
vector_Type (ETuFESpace->map(), exporter.mapType() ) );
exporter.addVariable (
ExporterData<mesh_Type>::ScalarField, "u", uFESpace,
uSolution, UInt (0) );
exporter.postProcess ( 1.0 );
exporter.closeFile();
```

Time Advance : BDF 1,2,3

```
#include <lifev/core/fem/TimeAdvanceBDF.hpp>
 const Real initialTime     = 0.0;
 const Real endTime         = 100.0;
 const Real timestep        = 1e-1;
 UInt BDFOrder = 2;
 TimeAdvanceBDF<vector_Type> bdf;
 bdf.setup (BDFOrder);
 Real currentTime = initialTime - timestep * BDFOrder;
 bdf.setInitialCondition ( *usolution );
```

Time Advance : BDF 1,2,3

```
bdf.updateRHSContribution ( timestep );
*uRhsUnique = *MassMatrix *
  bdf.rhsContributionFirstDerivative();
SystemMatrix.reset (new matrix_Type ( ETuFESpace->map() )
  );
double alphaTime = bdf.coefficientFirstDerivative ( 0 ) /
  timestep;
*SystemMatrix += *MassMatrix * alphaTime;
*SystemMatrix += *BaseMatrix;
bcManage (*SystemMatrix, *uRhsUnique,
          *uFESpace->mesh(), uFESpace->dof(),
             bcHandler, uFESpace->feBd(), 1.0, Real (0.0)
  );
SystemMatrix->globalAssemble();
```

Pacs-LifeV Project: Efficient and Fast Expression Template Assembler for PDEs discretization

Purpose: Educational Use of LifeV, programming Expression Template, c++11 standard.
Use LifeV Assembler: ETA

1. Mixed-Hybrid and low-Raviart-Thomas formulation for Darcy Problem .
2. Hyperbolic discretization by Godunov Flux for Saturation Equation .
3. Laplace-Beltrami problem applied to porous media flow .
4. Multigrid Space-Time preconditioner for solving Navier-Stokes system.
5. Turbolence Model: Reynolds-averaged Navier-Stokes equations (RANS) .