Instructional workshop on OpenFOAM programming
LECTURE # 5

Pavanakumar Mohanamuraly

April 28, 2014

Outline

Recap

- *fvMatrix* class and *boundaryCoeffs* and *internalCoeffs*
- Typos in Day 2 slides
  - *FVPatch* should have been *FvPatch*
  - *Make/files* - missed an *'s'* in *RobinFvPatchFields.C*

Implementing Robins BC

- No need for two versions to be implemented *fvm* and *fvc*
- Need to read in three extra parameter ϕ', *a* and *b*

$$a\phi(0) + b\phi'(0) \; and/or \; a\phi(L) + b\phi'(L) \tag{1}$$

- This will introduce one extra *RHS* source term to the Dirichlet BC
- Makes sense to use the Dirichlet BC and modify it for Robin

Hands on - Setting up

- Copy the contents of *FOAM_SRC/finiteVolume/fields/fvPatchFields/basic/fixedValue* to a folder named *MY_FOLDER/RobinBC*
- Rename all files having prefix *fixedValueFvPatchField* to *RobinFvPatchField*

```
for name in fixedValueFvPatchField*
do
    newname=RobinFvPatchField"$(echo "$name" | cut -
        c23-)"
    mv "$name" "$newname"
done
```

- Find and replace text *fixedValueFvPatchField* to *RobinFvPatchField* in all files

```
sed -i 's/fixedValueFvPatchField/RobinFvPatchField/g
    ' RobinFvPatchField*
```

Hands on - Make changes

- Replace all *fixedValue* fields with *Robin*

```
sed -i 's/fixedValue/Robin/g' RobinFvPatchField*
```

- Runtime type information

```
TypeName("Robin");
```

- Runtime object selection

```
RobinFvPatchFields.C:37:makePatchFields(Robin);
RobinFvPatchFields.H:39:makePatchTypeFieldTypedefs(
    Robin)
RobinFvPatchFieldsFwd.H:40:
    makePatchTypeFieldTypedefs(Robin)
```

- *wclean* and
- *wmake libso* to create library *libRobinBC.so*

Hands on - Compile code

- Create the *Make* folder with *files* and *options* as follows

files

```
RobinFvPatchFields.C

LIB = libRobinBC
```

options

```
EXE_INC = \
    -I$(LIB_SRC)/finiteVolume/lnInclude -g
EXE_LIBS = -lfiniteVolume
```

- *wmake libso* to create library *libRobinBC.so*

Hands on - More changes

- Make the *fixesValue*() boolean function return *false* in file *RobinFvPatchField.H*

```
virtual bool fixesValue() const
{
  return false;
}
```

Hands on - Preliminary testing I

- Go to the $1d$ case folder and add the following to *system/controlDict*

```
libs ("libRobinBC.so");
```

- Set the library environment search path to the *Make/linux* $*****$ folder (where the *libRobinBC.so* is created)
- Run the previous hands on example and check if you get errors
- If you get a warning as shown below

```
From function dlLibraryTable::open(const fileName&
    functionLibName)
in file db/dlLibraryTable/dlLibraryTable.C at line
    85
could not load dlopen(libRobinBC.so, 9): image not
    found
```

- Check your library path and see if the lib file exists

Hands on - Preliminary testing II

- In the fields file replace all *fixedValue* types to *Robin*
- Re-run the code and it should give the same results as run using *fixedValue*
- This ensures that the BC is compiled, loaded and setup correctly

Getting inputs - ϕ', a and b

- Declare variables (RobinFvPatchField.H)

```
template<class Type>
class RobinFvPatchField
: public fvPatchField<Type>
{
  //- The $\phi'$ value
  Field<Type> gradPhiBoundary_;
  //- The const parameter a and b
  scalar a_, b_;
```

- *gradPhiBoundary_* is of type "*Field* $<$ *Type* $>$" (*scalar*, *vector* or *tensor*)

Getting inputs - ϕ', a and b

- Constructor - 1 (RobinFvPatchField.C)

```
//- Construct from patch and internal field
template<class Type>
RobinFvPatchField<Type>::RobinFvPatchField
(
    const fvPatch& p,
    const DimensionedField<Type, volMesh>& iF
)
:
fvPatchField<Type>( p, iF ),
gradPhiBoundary_( p.size(), pTraits<Type>::zero ),
a_(), b_()
{}
```

Getting inputs - ϕ', a and b

- Constructor - 2 (RobinFvPatchField.C)

```
template<class Type>
RobinFvPatchField<Type>::RobinFvPatchField
(
    const fvPatch& p,
    const DimensionedField<Type, volMesh>& iF,
    const dictionary& dict
)
:
fvPatchField<Type>( p, iF, dict, true ),
gradPhiBoundary_( "gradient", dict, p.size() ),
a_(dict.lookupOrDefault<scalar>( "a", scalar(1.0))),
b_(dict.lookupOrDefault<scalar>( "b", scalar(0.0)))
{ }
```

Getting inputs - ϕ', *a* and *b*

- Constructor - 3 (RobinFvPatchField.C)

```
template<class Type>
RobinFvPatchField<Type>::RobinFvPatchField
(
    const RobinFvPatchField<Type>& ptf,
    const fvPatch& p,
    const DimensionedField<Type, volMesh>& iF,
    const fvPatchFieldMapper& mapper
)
:
fvPatchField<Type>( ptf, p, iF, mapper ),
gradPhiBoundary_( ptf.gradPhiBoundary_ ),
a_( ptf.a_ ), b_( ptf.b_ )
{}
```

Getting inputs - ϕ', a and b

- Constructor - 4 (RobinFvPatchField.C)

```
template<class Type>
RobinFvPatchField<Type>::RobinFvPatchField
(
    const RobinFvPatchField<Type>& ptf
)
:
fvPatchField<Type>( ptf ),
gradPhiBoundary_( ptf.gradPhiBoundary_ ),
a_( ptf.a_ ), b_( ptf.b_ )
{}
```

Getting inputs - ϕ', a and b

- Constructor - 5 (RobinFvPatchField.C)

```
template<class Type>
RobinFvPatchField<Type>::RobinFvPatchField
(
    const RobinFvPatchField<Type>& ptf,
    const DimensionedField<Type, volMesh>& iF
)
:
fvPatchField<Type>( ptf, iF ),
gradPhiBoundary_( ptf.gradPhiBoundary_ ),
a_( ptf.a_ ), b_( ptf.b_ )
{}
```

Getting inputs - ϕ', a and b

- Printing the boundary patch information (RobinFvPatchField.C)

```
template<class Type>
void RobinFvPatchField<Type>::write(Ostream& os)
    const
{
    fvPatchField<Type>::write(os);
    this->writeEntry( "value", os);
    gradPhiBoundary_.writeEntry( "gradient", os);
    os.writeKeyword("a")
        << a_ << token::END_STATEMENT << nl;
    os.writeKeyword("b")
        << b_ << token::END_STATEMENT << nl;
}
```

- $token :: END\_STATEMENT = ";"$ (line delimiter)

Hands on - Testing BC input

- Compile code again to create *libRobinBC.so*
- Modify example (laplacian solver) - loop over boundaryFields and print field

```
Info << x.boundaryField()[ipatch];
```

- Edit the *boundaryField* entry of the 1*d* case

```
...
boundaryField
{
  left
  {
    type Robin;
    value uniform 10.0;
    gradient uniform 20.0;
    a 0.5;
    b 0.5;
  }
...
```

- Run the code and check values

Some concepts - *deltaCoeffs*()

- *deltaCoeffs*() is the reciprocal distance
  - Between owner and neighbour cell centroids of an internal face

surfaceInterpolation.C : lines (234 - 242)

```
// Set local references to mesh data
const volVectorField& C = mesh_.C();
const labelUList& owner = mesh_.owner();
const labelUList& neighbour = mesh_.neighbour();

forAll(owner, facei)
{
  DeltaCoeffs[facei] = 1.0/mag(C[neighbour[facei]] -
        C[owner[facei]]);
}
```

Some concepts - *deltaCoeffs*()

- *deltaCoeffs*() is the reciprocal distance
  - Between owner and boundary face centroid of patch face

surfaceInterpolation.C : lines (244 - 248)

```
forAll(DeltaCoeffs.boundaryField(), patchi)
{
  DeltaCoeffs.boundaryField()[patchi] =
    1.0/mag(mesh_.boundary()[patchi].delta());
}
```

fvPatch.C : lines (141 - 143)

```
Foam::tmp<Foam::vectorField> Foam::fvPatch::delta()
    const
{
  return Cf() - Cn();
}
```

Some concepts - Patch *gradient* and *value* coefficients

Patch object should return the following to evaluate patch BC for operators

- *valueInternalCoeffs*()
  - Return the matrix diagonal coefficients corresponding to the evaluation of the value of the patchField with given weights
- *valueBoundaryCoeffs*()
  - Return the matrix source coefficients corresponding to the evaluation of the value of the patchField with given weight
- *gradientInternalCoeffs*()
  - Return the matrix diagonal coefficients corresponding to the evaluation of the gradient of the patchField
- *gradientBoundaryCoeffs*()
  - Return the matrix source coefficients corresponding to the evaluation of the gradient of this patchField

Some concepts - Patch *value* coefficients example

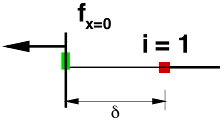
Using the value internal/boundary coefficients to get fvMatrix entries (Gauss-Gradient scheme)

gaussConvectionScheme.C (::fvmDiv function)

```
forAll(fvm.psi().boundaryField(), patchI)
{
  const fvPatchField<Type>& psf =
      fvm.psi().boundaryField()[patchI];
  const fvsPatchScalarField& patchFlux =
      faceFlux.boundaryField()[patchI];
  const fvsPatchScalarField& pw =
      weights.boundaryField()[patchI];

  fvm.internalCoeffs()[patchI] =
     patchFlux*psf.valueInternalCoeffs(pw);
  fvm.boundaryCoeffs()[patchI] =
     -patchFlux*psf.valueBoundaryCoeffs(pw);
}
```

Rationale behind value coefficients (face value interpolation)



- Dirichlet type

$$\phi_f = \underbrace{\phi_0}_{RHS\ \ source} + \underbrace{0}_{LHS\ \ coeff}$$

- Neumann boundary conditions

$$\int_0^\delta \mathbf{f}_{x=0} dx = \int_0^\delta \frac{\phi_f - \phi_1}{\delta} dx = \phi_f - \phi_1$$

$$\phi_f = \underbrace{1}_{LHS\ \ coeff} \times \phi_1 + \underbrace{\mathbf{f}_{x=0}\delta}_{RHS\ \ source} \quad (2)$$

Some concepts - *fixedValue* patch coefficients

fixedValueFvPatchField.C

```
template<class Type>
tmp<Field<Type> > fixedValueFvPatchField<Type>::
    valueInternalCoeffs
( const tmp<scalarField>& ) const
{
  return tmp<Field<Type> >
  ( new Field<Type>(this->size(), pTraits<Type>::zero) );
}
```

fixedValueFvPatchField.C

```
template<class Type>
tmp<Field<Type> > fixedValueFvPatchField<Type>::
    valueBoundaryCoeffs
( const tmp<scalarField>& ) const
{ return *this; } /// this has base class Field<Type>
```

Some concepts - *fixedGradient* patch coefficients

fixedGradientFvPatchField.C

```
template<class Type>
tmp<Field<Type> > fixedGradientFvPatchField<Type>::
    valueInternalCoeffs
( const tmp<scalarField>& ) const
{
    return tmp<Field<Type> >
      (new Field<Type>(this->size(), pTraits<Type>::one));
}
```

fixedGradientFvPatchField.C

```
template<class Type>
tmp<Field<Type> > fixedGradientFvPatchField<Type>::
    valueBoundaryCoeffs
( const tmp<scalarField>& ) const
{ return gradient()/this->patch().deltaCoeffs(); }
```

Some concepts - Patch *gradient* coefficients example

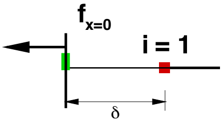
Using the value internal/boundary coefficients to get fvMatrix entries (Gauss-Laplacian scheme)

gaussLaplacianScheme.C (::fvmLaplacianUncorrected function)

```
forAll(fvm.psi().boundaryField(), patchI)
{
  const fvPatchField<Type>& psf = fvm.psi().boundaryField()[
      patchI];
  const fvsPatchScalarField& patchGamma =
    gammaMagSf.boundaryField()[patchI];

  fvm.internalCoeffs()[patchI] = patchGamma*psf.
      gradientInternalCoeffs();
  fvm.boundaryCoeffs()[patchI] = -patchGamma*psf.
      gradientBoundaryCoeffs();
}
```

Rationale behind gradient coefficients (one-sided face gradient)



- Dirichlet type

$$\mathbf{f}_{x=0} = \frac{\phi_f}{dx} = \frac{\phi_f - \phi_1}{\delta} = \underbrace{\frac{\phi_f}{\delta}}_{RHS\ \ source} - \underbrace{\frac{1}{\delta}}_{LHS\ \ coeff} \phi_1$$

- Neumann boundary conditions

$$\mathbf{f}_{x=0} = \underbrace{\phi'_{x=0}}_{RHS\ \ source} + \underbrace{0}_{LHS\ \ coeff}$$

Some concepts - *fixedValue* patch coefficients

fixedValueFvPatchField.C

```
template<class Type>
tmp<Field<Type> > fixedValueFvPatchField<Type>::
gradientInternalCoeffs() const
{
  return -pTraits<Type>::one*this->patch().deltaCoeffs();
}
```

fixedValueFvPatchField.C

```
template<class Type>
tmp<Field<Type> > fixedValueFvPatchField<Type>::
gradientBoundaryCoeffs() const
{
  return this->patch().deltaCoeffs()*(*this);
}
```

Some concepts - *fixedGradient* patch coefficients

fixedGradientFvPatchField.C

```
template<class Type>
tmp<Field<Type> > fixedGradientFvPatchField<Type>::
gradientInternalCoeffs() const
{
  return tmp<Field<Type> >
  ( new Field<Type>(this->size(), pTraits<Type>::zero) );
}
```

fixedGradientFvPatchField.C

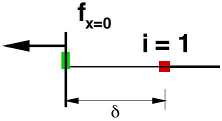
```
template<class Type>
tmp<Field<Type> > fixedGradientFvPatchField<Type>::
gradientBoundaryCoeffs() const
{
  return gradient();
}
```

Hands on - RobinBC implementation

- Use the value and gradient coefficients shown in previous slides and implement the following Robin BC
- At $x = 0$
$$a\phi(0) + b\phi'(0) \tag{3}$$
- At $x = L$
$$a\phi(L) + b\phi'(L) \tag{4}$$
- Compile the *libRobinBC.so* and test the code

Custom BC done !

Robin BC



- Value Coefficients

$$a\phi_f^{dir} + b\phi_f^{neu} = \underbrace{a}_{LHS} \times \phi_1 + \underbrace{b\mathbf{f}_{x=0}\delta + a\phi_0}_{RHS}$$

- Gradient Coefficients

$$a\phi_f'^{dir} + b\phi_f'^{neu} = \underbrace{-\frac{a}{\delta}}_{LHS} \times \phi_1 + \underbrace{\frac{a\phi_f}{\delta} + b\phi'_{x=0}}_{RHS} \tag{5}$$

- Remember a and b are between $[0, 1]$ $0 \leq a, b \leq 1$

End of Week 2 Day 3