Pseudo-code for CG routines and algorithms

Bill Henshaw

Centre for Applied Scientific Computing,
Lawrence Livermore National Laboratory,
henshaw@llnl.gov

February 5, 2016

Contents

1 Time-stepping pseudo code **2**
1.1 DomainSolver::advanceAdamsPredictorCorrector . . . 2

2 Moving grid pseudo code **3**
2.1 DomainSolver::moveGrids . . . 3
2.2 MovingGrids::moveGrids . . . 3
2.3 MovingGrids::moveDeformingBodies . . . 3

3 DeformingBodyMotion pseudo code **4**
3.1 DeformingBodyMotion::integrate . . . 4
3.2 DeformingBodyMotion::regenerateComponentGrids . . . 4
3.3 DeformingBodyMotion::correct . . . 4

4 Cgmp **5**
4.1 Cgmp::solve . . . 5
4.2 Cgmp::multiDomainAdvance . . . 6
4.3 Cgmp::multiDomainAdvanceNew . . . 7

5 Interfaces **8**
5.1 Cgmp::assignInterfaceRightHandSide . . . 8
5.2 DomainSolver::interfaceRightHandSide . . . 9

1 Time-stepping pseudo code

1.1 DomainSolver::advanceAdamsPredictorCorrector

Here is an overview of the `DomainSolver::advanceAdamsPredictorCorrector` function (cg/common/src/advancePC.bC) This is an explicit Adams predictor-corrector time stepper.

```
DomainSolver::advanceAdamsPredictorCorrector( ..., numberOfSubSteps, ..)
{
  initialize
  for( int mst=1; mst≤numberOfSubSteps; mst++ )  take time steps
    if adapt grids
      adaptGrids( ...  )
    if move grids
      moveGrids( ...  ); (Sec. 2.1)
      exposedPoints.interpolate(...)

     predictor step:
    getUt( ...  )
    interpolateAndApplyBoundaryConditions( ...  );
    solveForTimeIndependentVariables( ...  )
    correctMovingGrids( ...  )

    for( int correction=0; correction<numberOfCorrections; correction++ )
       corrector step:
      getUt( ...  )
      solveForTimeIndependentVariables( ...  )
      interpolateAndApplyBoundaryConditions( ...  );
      correctMovingGrids( ...  )
}
```

Figure 1: Pseudo-code outline of the advanceAdamsPredictorCorrector function.

2 Moving grid pseudo code

2.1 DomainSolver::moveGrids

Pseudo-code for `DomainSolver::moveGrids` (cg/common/src/move.C)

```
DomainSolver::moveGrids( t1,t2,t3,dt0, cgf1,cgf2,cgf3 )
{
  setInterfacesAtPastTimes( t1,t2,t3,dt0,cgf1,cgf2,cgf3 ); initialize interfaces
  parameters.dbase.get<MovingGrids>("movingGrids").moveGrids(t1,t2,t3,dt0,cgf1,cgf2,cgf3 );
  gridGenerator->updateOverlap( cg, mapInfo );  regenerate the grid with Ogen
}
```

2.2 MovingGrids::moveGrids

Pseudo-code for `MovingGrids::moveGrids` (cg/common/moving/src/MovingGrids.C)

```
MovingGrids::moveGrids( t1,t2,t3,dt0, cgf1,cgf2,cgf3 )
{
  First move the bodies (but not the grids):
  detectCollisions(cgf1);
  rigidBodyMotion( t1,t2,t3,dt0,cgf1,cgf2,cgf3 );
  moveDeformingBodies( t1,t2,t3,dt0,cgf1,cgf2,cgf3 );
  userDefinedMotion( t1,t2,t3,dt0,cgf1,cgf2,cgf3 );
  Apply any matrix motions:  rotate, shift, scale

  getGridVelocity( cgf2,t2 );
  Now move the grids:
  for( grid=0; grid<numberOfBaseGrids; grid++ )
    MatrixTransform & transform = *cgf3.transform[grid];
    if( moveOption(grid)==matrixMotion )
      apply specified rotation and/or shift
      transform.rotate(...)
    else if( moveOption(grid)==rigidBody )
      rotate and shift the rigid body
      transform.rotate(...)
  for( int b=0; b<numberOfDeformingBodies; b++ )
    deformingBodyList[b]->regenerateComponentGrids( newT, cgf3.cg );

  getGridVelocity( cgf3,t3 );
}
```

2.3 MovingGrids::moveDeformingBodies

Pseudo-code for `MovingGrids::moveDeformingBodies` (cg/common/moving/src/MovingGrids.C)

```
MovingGrids::moveDeformingBodies( t1,t2,t3,dt0, cgf1,cgf2,cgf3 )
{
  for( int b=0; b<numberOfDeformingBodies; b++ )
    deformingBodyList[b]->integrate( t1,t2,t3,cgf1,cgf2,cgf3, stress);
}
```

3 DeformingBodyMotion pseudo code

The `DeformingBodyMotion` class handles deforming bodies.

3.1 DeformingBodyMotion::integrate

Pseudo-code for `DeformingBodyMotion::integrate` (cg/common/moving/src/DeformingBodyMotion.C)
This function is called by `MovingGrids::movingGrids` to move the deforming body (but not the grid associated with the deforming body).

```
MovingGrids::integrate( t1,t2,t3,dt0, cgf1,cgf2,cgf3, stress )
{
  if( elasticShell )
    advanceElasticShell(t1,t2,t3,cgf1,cgf2,cgf3,stress,option);
  else if( ...  )

  for( int face=0; face<numberOfFaces; face++ )
    if( ...  )
    else if( userDefinedDeformingBodyMotionOption==interfaceDeform )
      The deformed surface is obtained from the boundaryData array:
      RealArray & bd = parameters.getBoundaryData(side,axis,grid,cg[grid]);
      x0 = bd;

}
```

3.2 DeformingBodyMotion::regenerateComponentGrids

Pseudo-code for `DeformingBodyMotion::regenerateComponentGrids` (cg/common/moving/src/DeformingBodyMoti
This function is called by `MovingGrids::movingGrids` (after calling `DeformingBodyMotion:;integrate`)
to actually generate the grid associated with the deforming body.

```
DeformingBodyMotion::regenerateComponentGrids( const real newT, CompositeGrid & cg)
{
  for( int face=0; face<numberOfFaces; face++ )
    hyp.generate(); Call the hyperbolic grid generator.
    Save the grid in the GridEvolution list:
    gridEvolution[face]->addGrid(dpm.getDataPoints(),newT);
}
```

3.3 DeformingBodyMotion::correct

Pseudo-code for `DeformingBodyMotion::correct` (cg/common/moving/src/DeformingBodyMotion.C)
This function is called by `MovingGrids::correctGrids`.

```
DeformingBodyMotion::correct( t1, t2, GridFunction & cgf1,GridFunction & cgf2 )
{
  This function currently does nothing.
}
```

4 Cgmp

4.1 Cgmp::solve

Pseudo-code for `Cgmp::solve` (cg/mp/src/solve.C)

```
Cgmp::solve()
{
  cycleZero();
  buildRunTimeDialog();
  for( int step=0; step<maximumNumberOfSteps && !finish; )
    if( t≥ nextTimeToPrint )
      printTimeStepInfo(step,t,cpuTime);
      saveShow( gf[current] );
      finish=plot(t, optionIn, tFinal);
      if( finish ) break;
    dtNew = getTimeStep( gf[current] ); choose time step
    computeNumberOfStepsAndAdjustTheTimeStep(t,tFinal,nextTimeToPrint,numberOfSubSteps,dtNew);

    advance(tFinal); advance to t=nextTimeToPrint

}
```

4.2 Cgmp::multiDomainAdvance

Pseudo-code for `Cgmp::multiDomainAdvance` (cg/mp/src/multiDomainAdvance.C)

```
Cgmp::multiDomainAdvance( real &t, real & tFinal )
{
  if( initialize )
    initializeInterfaceBoundaryConditions( t,dt,gfIndex );
    ForDomain( d ) assignInterfaceRightHandSide( d, t, dt, correct, gfIndex );
    ForDomain( d ) domainSolver[d]->initializeTimeStepping( t,dt );
  Take some time steps:
  for( int i=0; i<numberOfSubSteps; i++ )
    ForDomain( d )
      domainSolver[d]->startTimeStep( t,dt,...  );
      numberOfRequiredCorrectorSteps=...; gridHasChanged=...;
    if( gridHasChanged )
      initializeInterfaces(gfIndex); initializeInterfaceBoundaryConditions(...);

    for( int correct=0; correct<=numberOfCorrectorSteps; correct++ )
      ForDomain( d )
        assignInterfaceRightHandSide( d, t+dt, dt, correct, gfIndex ); (Sec. 5.1)
        domainSolver[d]->takeTimeStep( t,dt,correct,advanceOptions[d] );
      if( hasConverged = checkInterfaceForConvergence( ..  )  )  break;

    ForDomain( d ) domainSolver[d]->endTimeStep( td,dt,advanceOptions[d] );
    t+=dt;
}
```

4.3 Cgmp::multiDomainAdvanceNew

Pseudo-code for `Cgmp::multiDomainAdvanceNew` (cg/mp/src/multiDomainAdvanceNew.bC). This is the new versio of the multi-domain advance routine that supports more general time stepping and the use of AMR.

```
Cgmp::multiDomainAdvanceNew( real &t, real & tFinal )
{
  if( initialize )
    initializeInterfaceBoundaryConditions( t,dt,gfIndex );
    ForDomain( d ) assignInterfaceRightHandSide( d, t, dt, correct, gfIndex );
    ForDomain( d ) domainSolver[d]->initializeTimeStepping( t,dt );
  Take some time steps:
  for( int i=0; i<numberOfSubSteps; i++ )
    ForDomain( d )
      domainSolver[d]->startTimeStep( t,dt,...  );
      numberOfRequiredCorrectorSteps=...; gridHasChanged=...;
    if( gridHasChanged )
      initializeInterfaces(gfIndex); initializeInterfaceBoundaryConditions(...);
    Get current interface residual and save current interface values :
    getInterfaceResiduals( t, dt, gfIndex, maxResidual, saveInterfaceTimeHistoryValues );

      Stage I: advance the solution but do not apply BC's:
      ForDomain( d )
        assignInterfaceRightHandSide( d, t+dt, dt, correct, gfIndex );
        domainSolver[d]->takeTimeStep( t,dt,correct,step but no BC's );

      Stage II: Project interface values part 1:
      interfaceProjection( t+dt, dt, correct, gfIndex,set interface values );

      Stage III: evaluate the interface conditions and apply the boundary conditions:
      ForDomain( d )
        assignInterfaceRightHandSide( d, t+dt, dt, correct, gfIndex ); (Sec. 5.1)
        domainSolver[d]->takeTimeStep( t,dt,correct,apply BC's );

      Stage IV: Project interface values part 2:
      interfaceProjection( t+dt, dt, correct, gfIndex,set interface ghost values );

      if( hasConverged = checkInterfaceForConvergence( ..  )  )  break;

    ForDomain( d ) domainSolver[d]->endTimeStep( td,dt,advanceOptions[d] );
    t+=dt;
}
```

5 Interfaces

5.1 Cgmp::assignInterfaceRightHandSide

The `Cgmp::assignInterfaceRightHandSide` function is used to get interface values from a source domain and set interface values on a target domain. It is used in the Cgmp::multiDomainAdvance routine 4.2.

Here is `Cgmp::assignInterfaceRightHandSide` (cg/mp/src/assignInterfaceBoundaryConditions.C)

```
int Cgmp::assignInterfaceRightHandSide( int d, real t, real dt, int correct, std::vector<int> &
gfIndex )
d :  target domain, assign the interface RHS for this domain.
{
  if( interfaceList.size()==0 )
    initializeInterfaces(gfIndex); Create the list of interfaces.

  for( each interface on domain d )
    InterfaceDescriptor & interfaceDescriptor = interfaceList[inter];
    Target and source grid functions:
    GridFunction & gfTarget = domainSolver[domainTarget]->gf[gfIndex[domainTarget]];
    GridFunction & gfSource = domainSolver[domainSource]->gf[gfIndex[domainSource]];

    Get source data:
    for( int face=0; face<gridListSource.size(); face++ )
      domainSolver[domainSource]->interfaceRightHandSide( getInterfaceRightHandSide, ...  );

    Transfer the source arrays to the target arrays:
    interfaceTransfer.transferData(...  );

    Adjust the target data before assigning:
    for( int face=0; face<gridListTarget.size(); face++ )
      Extrapolate the initial guess.
      Under-relaxed iteration.

    Assign the target data:
    for( int face=0; face<gridListTarget.size(); face++ )
      domainSolver[domainTarget]->interfaceRightHandSide( setInterfaceRightHandSide,...);

}
```

5.2 DomainSolver::interfaceRightHandSide

The `DomainSolver::interfaceRightHandSide` function is used to get or set interface values. Each DomainSolver (cgad, cgcns, cgins, cgsm,...) has a version of this routine. The generic version appears in cg/common/src/interfaceBoundaryConditions.C.

Here is `Cgcns::interfaceRightHandSide` (cg/cns/src/interface.bC)

```
Cgcns::interfaceRightHandSide( InterfaceOptionsEnum option, int interfaceDataOptions,
      GridFaceDescriptor & info, GridFaceDescriptor & gfd, int gfIndex, real t )
{
  RealArray & bd = parameters.getBoundaryData(side,axis,grid,mg); Interface data on this domain.
  RealArray & f = *info.u; Interface data from another domain.

  if( interfaceType(side,axis,grid)==Parameters::heatFluxInterface )
    if( option==setInterfaceRightHandSide )
      bd(I1,I2,I3,tc)=f(I1,I2,I3,tc); Set T (using values from another domain).
    else if( option==getInterfaceRightHandSide )
      Evaluate a0*T + a1*Tn (to send to another domain).
      f(I1,I2,I3,tc) = a[0]*uLocal(I1,I2,I3,tc) + a[1]*( normal(I1,I2,I2,0)*ux + ...  );

  else if( interfaceType(side,axis,grid)==Parameters::tractionInterface )
    if( option==setInterfaceRightHandSide )
      bd(I1,I2,I3,V)=f(I1,I2,I3,V); Set positions of the interface.
    else if( option==getInterfaceRightHandSide )
      parameters.getNormalForce( gf[gfIndex].u,traction,ipar,rpar );
      f(I1,I2,I3,V)=traction(I1,I2,I3,D);
      InterfaceDataHistory & idh = gfd.interfaceDataHistory; Holds interface history.
      if( interfaceDataOptions & Parameters::tractionRateInterfaceData )
        RealArray & f0 = idh.interfaceDataList[prev].f;
        f(I1,I2,I3,Vt)= (f(I1,I2,I3,V) - f0(I1,I2,I3,V))/dt; Time derivative of the traction.

}
```