ULaMBAtoR

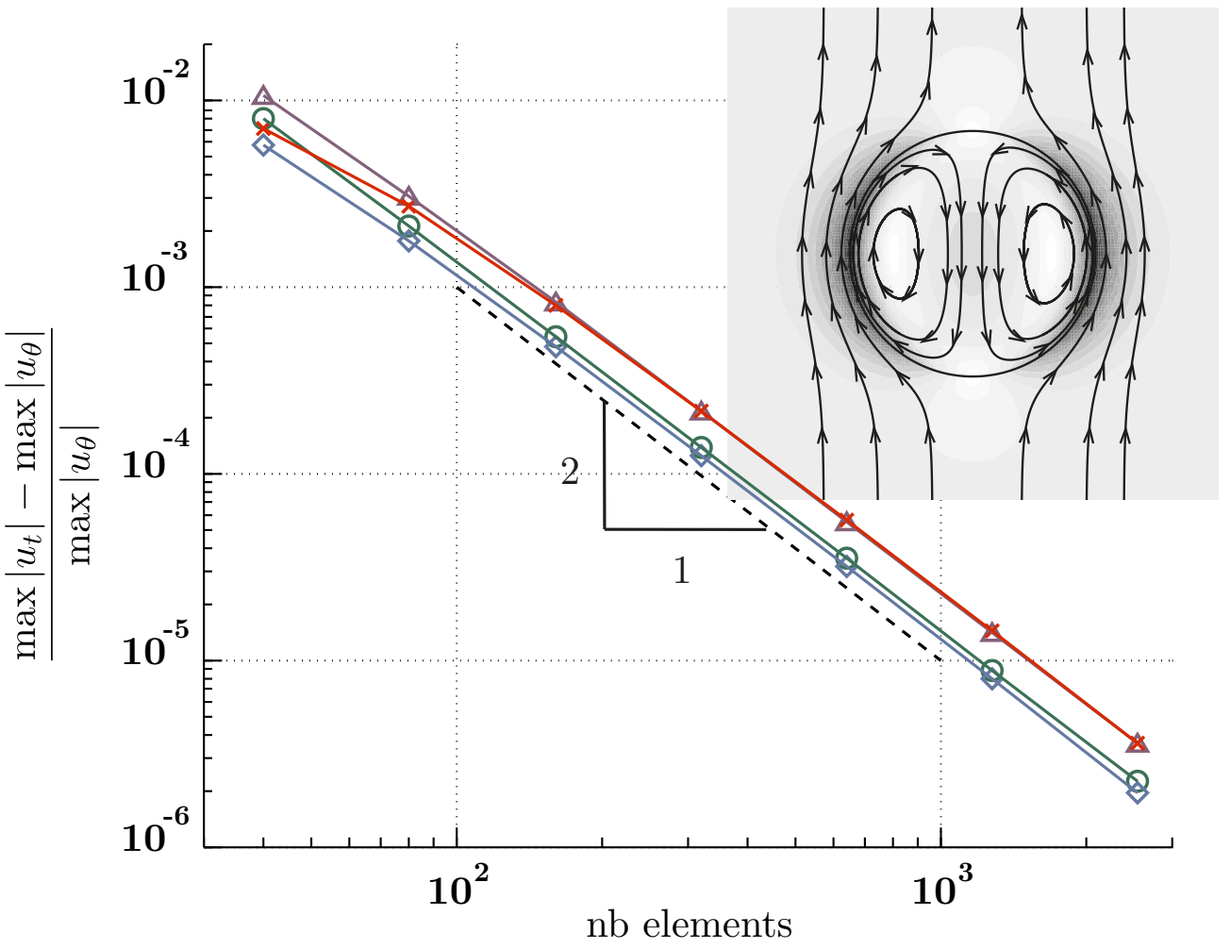
1. Workshop (part III)

December 11th 2015
TIPs Lab

ULB

Empirical recommendations

- Discretization, spatial point density:

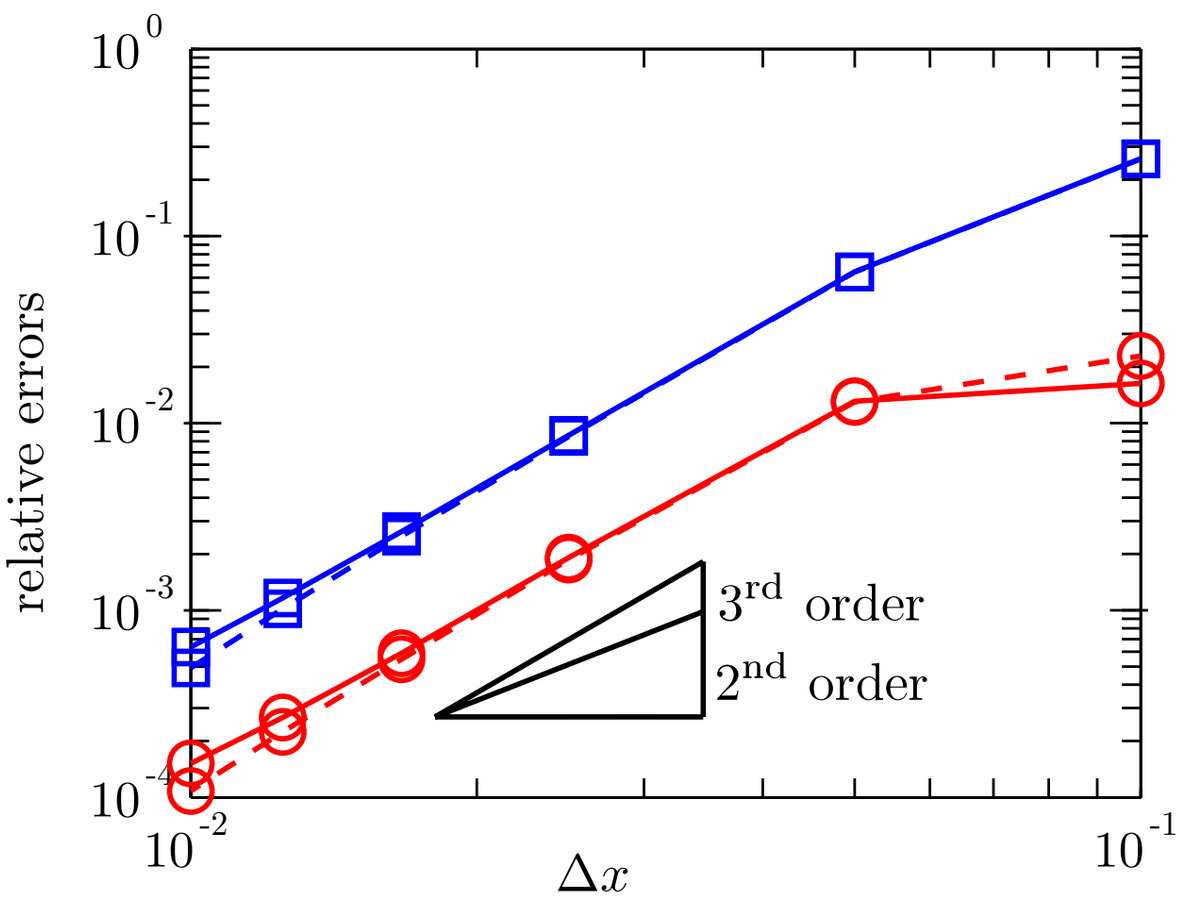


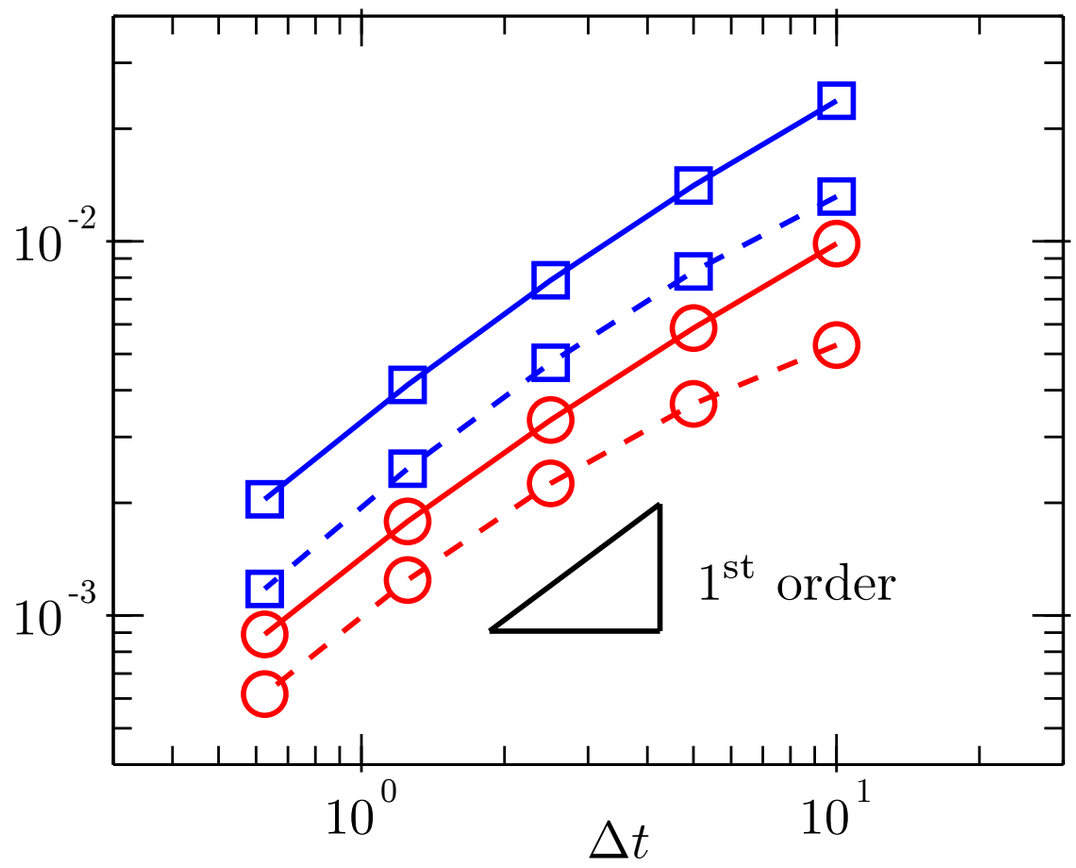
2 viscosity ratios
2 aspect ratios

Empirical recommendations

- Discretization, spatial point density:

- Time marching, time step:

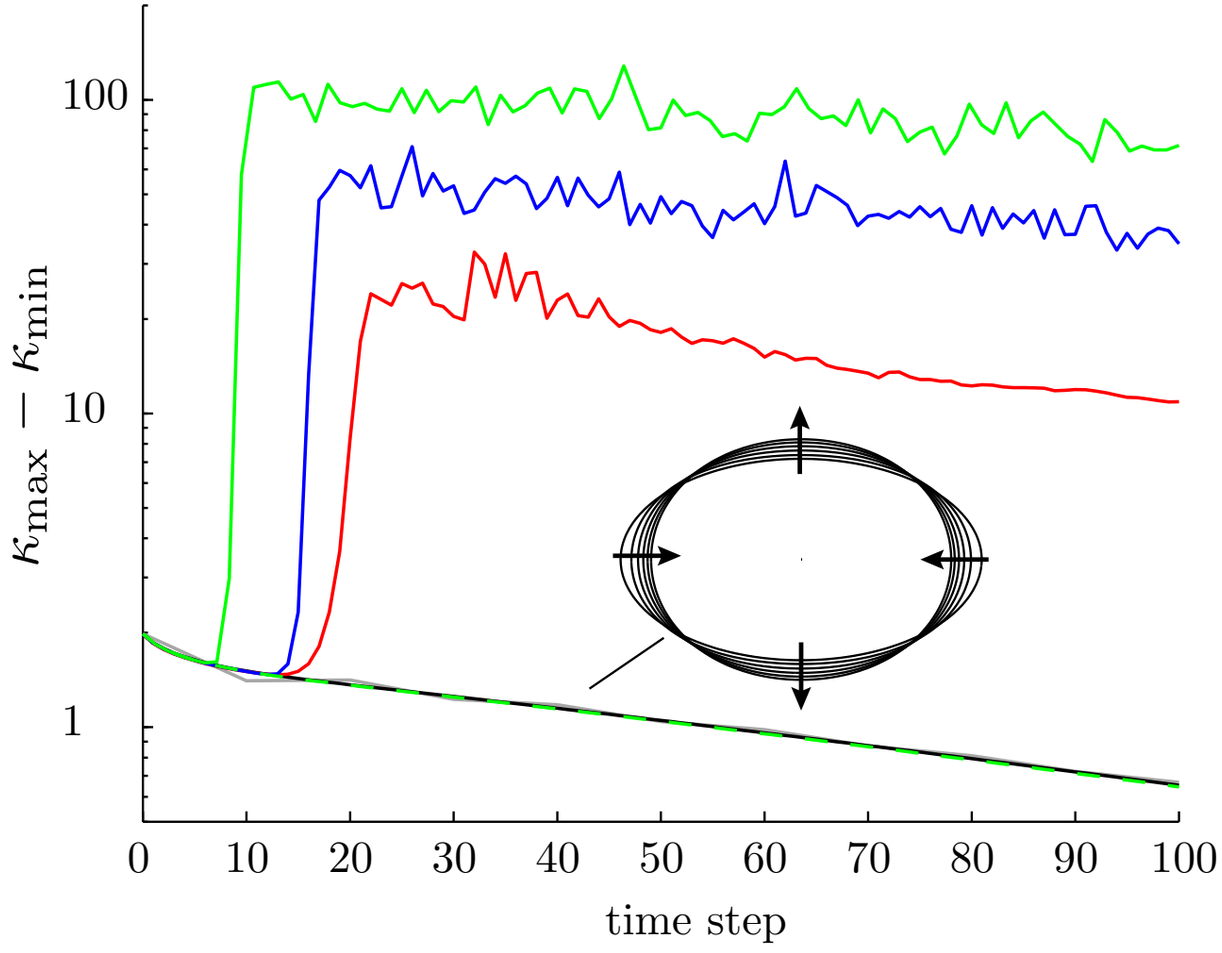




Empirical recommendations

- Discretization, spatial point density:

- Time marching, time step:

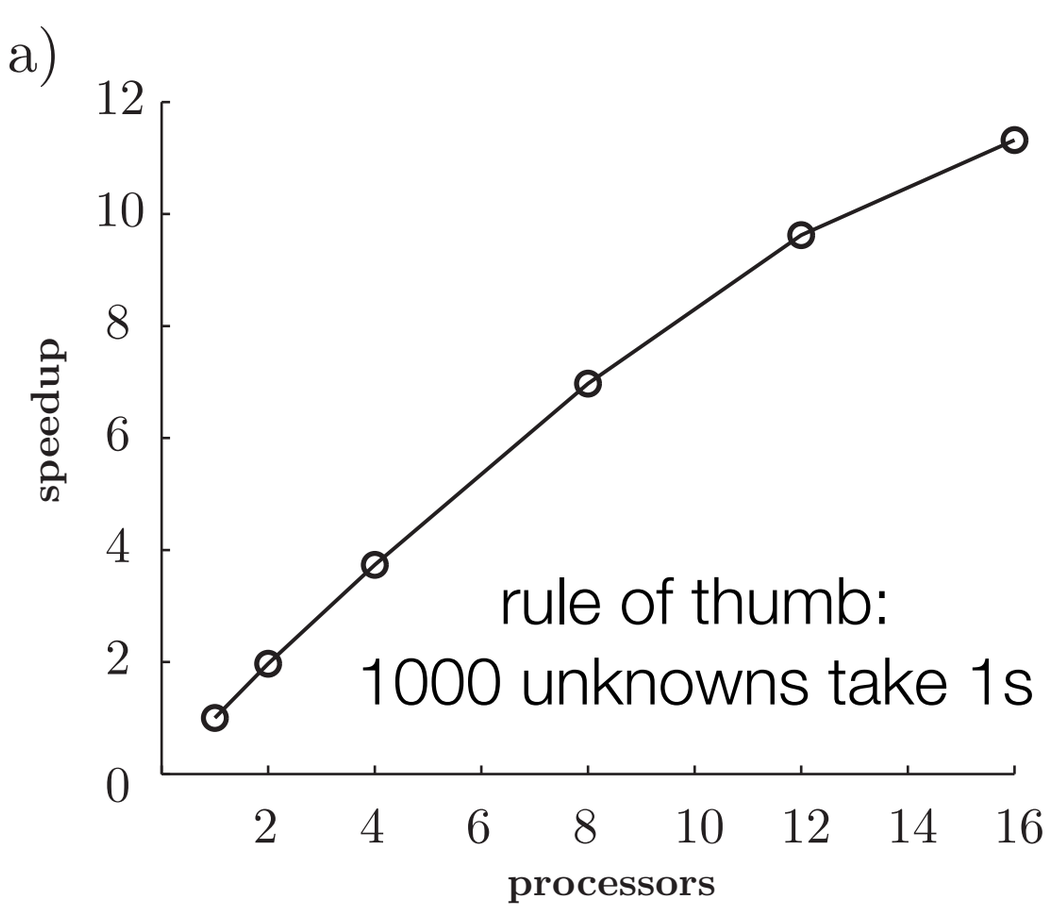


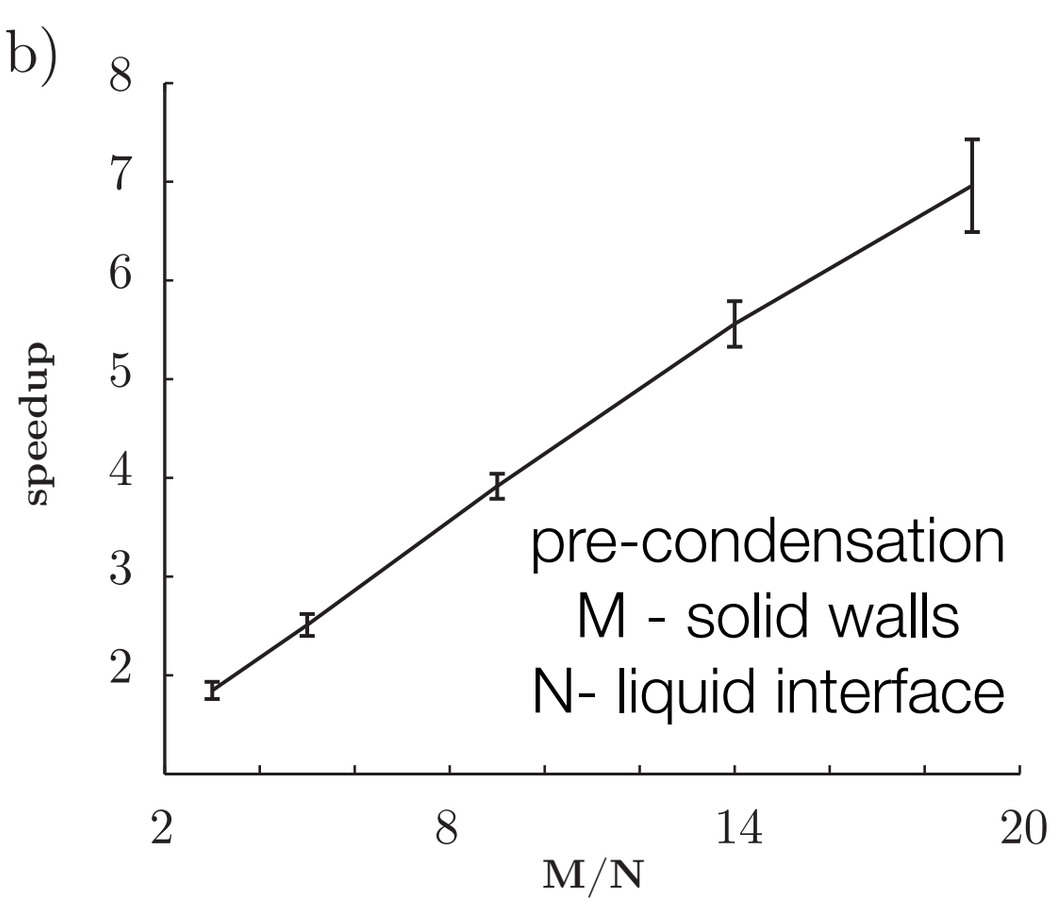
$$\frac{\Delta T}{\Delta S} \leq C$$

different density,
above and below threshold

Empirical recommendations

- Discretization, spatial point density:
- Time marching, time step:
- Number of unknowns and solution time:

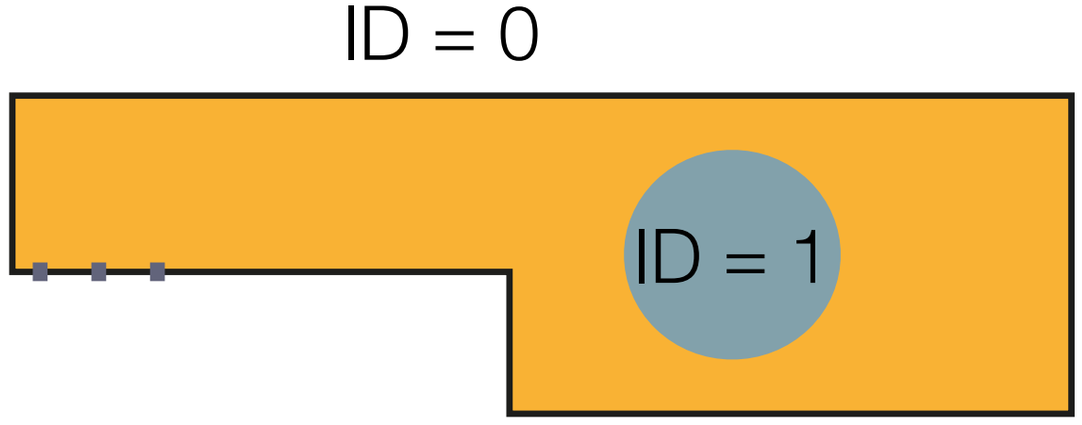




Data structures in `Bndelement.cpp`

---

my_bem->**Elements**[ID].*Variable*

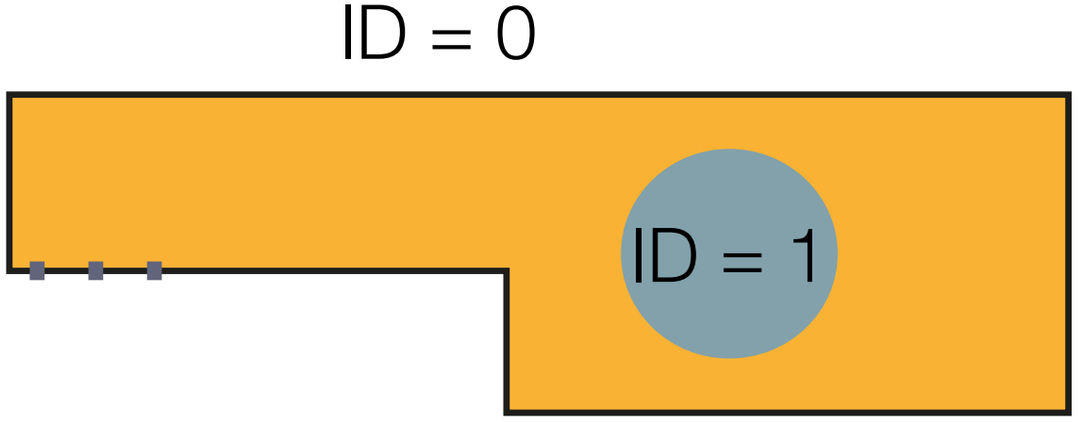


Data structures in `Bndelement.cpp`

*my_bem->***Elements**[ID].*Variable*

Panels

PanelSize[*Panel*]



Data structures in `Bndelement.cpp`

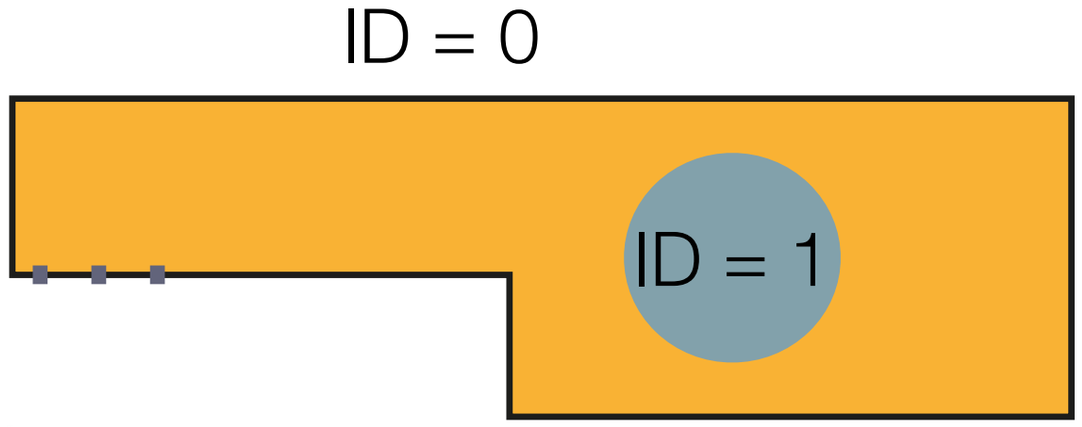
*my_bem->***Elements**[ID].*Variable*

Panels

PanelSize[*Panel*]

Xld[*Panel*][*Point*]

Yld[*Panel*][*Point*]



continuous array
indexing to -5 and end+5
is possible

Data structures in `Bndelement.cpp`

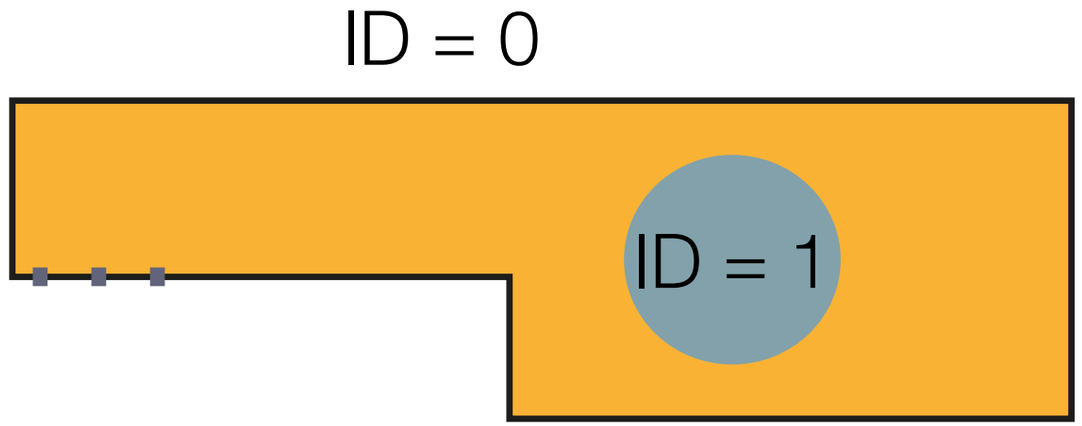
my_bem->**Elements**[ID].*Variable*

Panels

PanelSize[*Panel*]

XId[*Panel*][*Point*]

YId[*Panel*][*Point*]

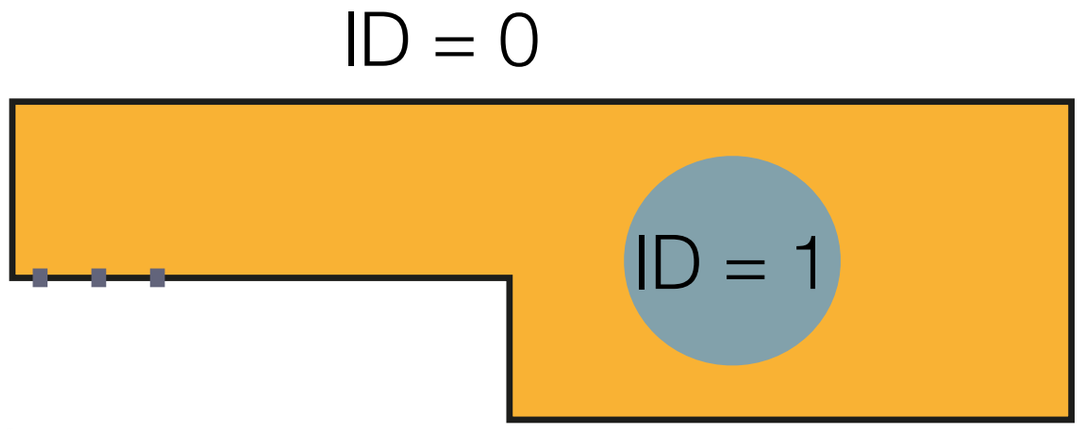


continuous array
indexing to -5 and end+5
is possible

KC[*Point*] Curvature of a drop (*my_bem*.**UpdateCourbure**(ID))

Data structures in `Bndelement.cpp`

my_bem->**Elements**[ID].*Variable*



Panels

PanelSize[*Panel*]

XId[*Panel*][*Point*]
YId[*Panel*][*Point*]

continuous array
indexing to -5 and end+5
is possible

KC[*Point*] Curvature of a drop (*my_bem*.**UpdateCourbure**(ID))

BCtype[*Panel*]
BCvalueX[*Panel*]
BCvalueY[*Panel*]

Boundary condition
type and value

Data structures in `MatrixBlock.cpp`

---

my_mat->Variable

$$\begin{pmatrix} T_{11} & T_{12} & T_{13} & T_{14} \\ T_{21} & T_{22} & T_{23} & T_{24} \\ T_{31} & T_{32} & T_{33} & T_{34} \\ T_{41} & T_{42} & T_{43} & T_{44} \end{pmatrix} \begin{pmatrix} \vec{u}_1 \\ \vec{u}_2 \\ \vec{u}_3 \\ \vec{u}_4 \end{pmatrix} = \begin{pmatrix} \vec{G}_1 \cdot \vec{f} \\ \vec{G}_2 \cdot \vec{f} \\ \vec{G}_3 \cdot \vec{f} \\ \vec{G}_4 \cdot \vec{f} \end{pmatrix}$$

Data structures in `MatrixBlock.cpp`

my_mat->Variable

$$\begin{pmatrix} T_{11} & T_{12} & T_{13} & T_{14} \\ T_{21} & T_{22} & T_{23} & T_{24} \\ T_{31} & T_{32} & T_{33} & T_{34} \\ T_{41} & T_{42} & T_{43} & T_{44} \end{pmatrix} \begin{pmatrix} \vec{u}_1 \\ \vec{u}_2 \\ \vec{u}_3 \\ \vec{u}_4 \end{pmatrix} = \begin{pmatrix} \vec{G}_1 \cdot \vec{f} \\ \vec{G}_2 \cdot \vec{f} \\ \vec{G}_3 \cdot \vec{f} \\ \vec{G}_4 \cdot \vec{f} \end{pmatrix}$$

Lambda (viscosity ratio)

kF (confinement) $k_F = \sqrt{12}\frac{L}{H}$

Data structures in `MatrixBlock.cpp`

my_mat->Variable

$$\begin{pmatrix} T_{11} & T_{12} & T_{13} & T_{14} \\ T_{21} & T_{22} & T_{23} & T_{24} \\ T_{31} & T_{32} & T_{33} & T_{34} \\ T_{41} & T_{42} & T_{43} & T_{44} \end{pmatrix} \begin{pmatrix} \vec{u}_1 \\ \vec{u}_2 \\ \vec{u}_3 \\ \vec{u}_4 \end{pmatrix} = \begin{pmatrix} \vec{G}_1 \cdot \vec{f} \\ \vec{G}_2 \cdot \vec{f} \\ \vec{G}_3 \cdot \vec{f} \\ \vec{G}_4 \cdot \vec{f} \end{pmatrix}$$

Lambda (viscosity ratio)

kF (confinement) $k_F = \sqrt{12}\frac{L}{H}$

nOFm (number of unknowns)

Data structures in `MatrixBlock.cpp`

my_mat->Variable

$$\begin{pmatrix} T_{11} & T_{12} & T_{13} & T_{14} \\ T_{21} & T_{22} & T_{23} & T_{24} \\ T_{31} & T_{32} & T_{33} & T_{34} \\ T_{41} & T_{42} & T_{43} & T_{44} \end{pmatrix} \begin{pmatrix} \vec{u}_1 \\ \vec{u}_2 \\ \vec{u}_3 \\ \vec{u}_4 \end{pmatrix} = \begin{pmatrix} \vec{G}_1 \cdot \vec{f} \\ \vec{G}_2 \cdot \vec{f} \\ \vec{G}_3 \cdot \vec{f} \\ \vec{G}_4 \cdot \vec{f} \end{pmatrix}$$

Lambda (viscosity ratio)

kF (confinement) $k_F = \sqrt{12}\frac{L}{H}$

nOFm (number of unknowns)

A[*line*•*nOFm*+*row*] (Matrix in line format)

rhs[2•point (+1)] (right hand side or solution

Customized Output routines

my_mat->**EnableWrite**(Directory) At first and always!

my_mat->**LogPrepare**(LogID,description)

Creates or overwrites a file,
without it data could be appended

my_bem->**LogCustom**(LogID,mode)

my_mat->**LogCustom**(LogID,mode)

Extract data you are interested,
in a format that suits your application

Customized Output routines

```
case 2:
{
    // FOR CORDERO STRETCH
    perimeter = 0;
    double xmax = -10.0;
    double xmin = 10.0;
    double ymax = -10.0;
    double ymin = 10.0;

    for(int i=0; i<Elements[1].PanelSize[0];i++){
        perimeter += sqrt( pow(Elements[1].XId[0][i+1]-Elements[1].XId[0][i] ,2)+pow(Elements[1].YId[0][i+1]-Elements[1].YId
            [0][i],2));
        xmax = max(xmax, Elements[1].XId[0][i]);
        xmin = min(xmin, Elements[1].XId[0][i]);
        ymax = max(ymax, Elements[1].YId[0][i]);
        ymin = min(ymin, Elements[1].YId[0][i]);

    }

    char logurl[120];
    FILE* logfile;
    sprintf(logurl,"%s/log%i.txt", workdir.c_str(),logid);
    logfile = fopen(logurl, "a");
    getCoM(1, xcm, ycm);
    someWay += xcm;
    fprintf(logfile, "%f %f %f %f %f %f\n", xcm, ycm, xmax-xmin, ymax-ymin, perimeter, runtime);
    fclose(logfile);
    return xcm;
    break;
}
```

Extract data you are interested,
in a format that suits your application

Time Marching

One step Euler explicit/Runge Kutta scheme.

```
int MatrixBlock::SolveRK1(){
```

Initialize variables for time adaptation.

```
    int status = 0;
    double stretch =1.0;
    double deltaT = deltaT0;
    cout<<"RK1";
```

Perform `nSubSteps` iterations. At first `SeedDrop` remeshes the droplet if necessary.

```
    for (int n=0; n<nSubSteps; ++n) {
        bem->SeedDrop();
        dominantForce = 0;
```

Create and solve the problem.

```
        Build(deltaT);
        Solve();
```

Determine if the time step should be reduced with `DetTime`, then evolve the interface.

```
        stretch = DetTime();
        deltaT = deltaT0*stretch;
        AdvanceTimestep(deltaT, 1);

        cout<<"."<<flush;
    }
    bem->runstep++;
    return status;
}
```

Time Marching

AdvanceTimeStep

```
if (modus==1){
    // One step advance
    bem->allEvolve(1.0, 0.0, deltaT, &rhs[2*fixsize]);
} else if (modus==2){
    // Two step 2/2 advance
    bem->allEvolve(0.5, 0.5, deltaT*0.5, &rhs[2*fixsize]);
}

bem->runtime += deltaT;
bem->allStock();

bem->correctArea();
bem->allStock();
```

Time Marching

AdvanceTimeStep

```
if (modus==1){
    // One step advance
    bem->allEvolve(1.0, 0.0, deltaT, &rhs[2*fixsize]);
} else if (modus==2){
    // Two step 2/2 advance
    bem->allEvolve(0.5, 0.5, deltaT*0.5, &rhs[2*fixsize]);
}
```

```
void BndBlock::Evolve(int blockId, double Cold, double Cnew, double deltaT, double* xvec){
    // Evolves droplets from buffer IntfXY to XYId
    int dynNodes;
    double *Xn, *Yn, *Xo, *Yo;
    dynNodes = 0;

    dynNodes = Elements[blockId].getFullSize();
    Xn = Elements[blockId].XId[0];
    Yn = Elements[blockId].YId[0];
    Xo = &Elements[blockId].IntfX[5];
    Yo = &Elements[blockId].IntfY[5];

    for (int n=0; n<dynNodes; n++) {
        Xn[n] = Cold*Xo[n] + Cnew*Xn[n] + xvec[2*n]*deltaT;//*fabs(lnx);
        Yn[n] = Cold*Yo[n] + Cnew*Yn[n] + xvec[2*n+1]*deltaT;//*fabs(lny);
    }

    WrapInterface(blockId);
}
```

Time for your own cases

- What do you want to find out?
- From what side to approach your problem first?
- Make a sketch and non-dimensionalize it.
- Translate it into Ulambator
- Solve and Inspect
- Where do you go from here?