

18847 Final Project - Finite Element Method

Authors

Alan Abraham (amabraha@andrew.cmu.edu)

Myles Mwathe (mmwathe@andrew.cmu.edu)

Yao Xiao (yaox3@andrew.cmu.edu)

1. Integrate Triangle into FEGrid (Yao)

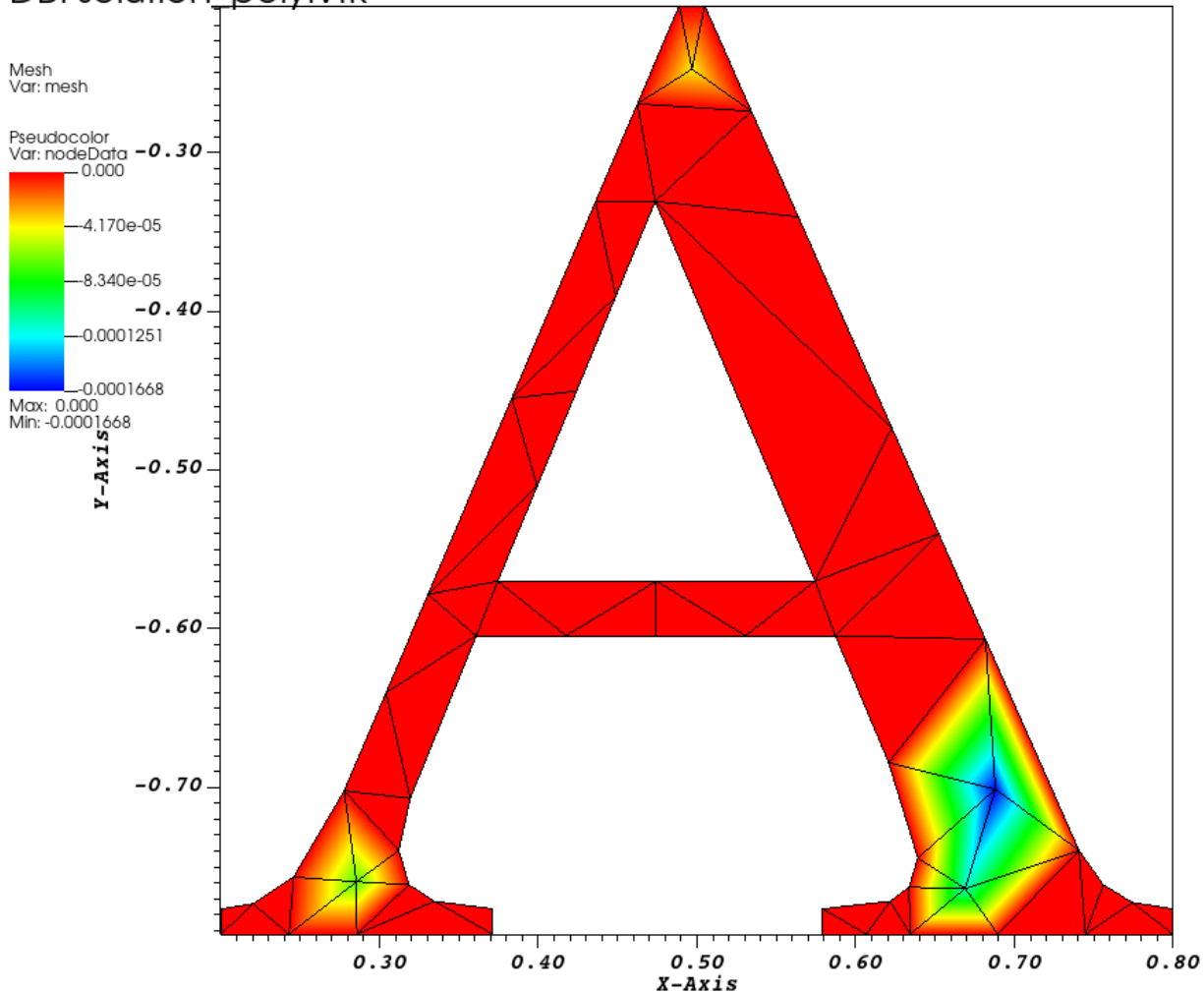
- Scan .poly file
- Call `triangulate()` from Triangle library (written in C) to refine triangulation with specified area constraint

```
extern "C"
```

- Load nodes and elements from output
 - same as original constructor

Sample VisIt output of .poly file

DB: solution_poly.vtk

user: yaoxiao
Mon Apr 7 14:35:14 2025

2. Template FEPoissonOperator by data type (Myles)

Key changes required

- Converted classes to template <typename T>
- Modified member variables and method signatures to use type T
- Added explicit template instantiations for each supported type
- Updated constructors and operators to handle templated types

Modified core classes

- `FEPoissonOperator`
- `SparseMatrix`
- `JacobiSolver`

Implementation approach

- Separated declarations (.H) from implementations (.hpp)
- Added necessary header includes (e.g., <complex>)

Testing

- Created test cases for each data type
- Verified solution correctness for annulus mesh
- Ensured consistent behavior across all supported types

3. Non-zero Dirichlet boundary conditions (Myles)

Goal

Set solution directly at boundaries to known values:

- **Before:** Boundary assumed to be zero.
- **Now:** Allows custom boundary values ϕ_Φ .

Implementation

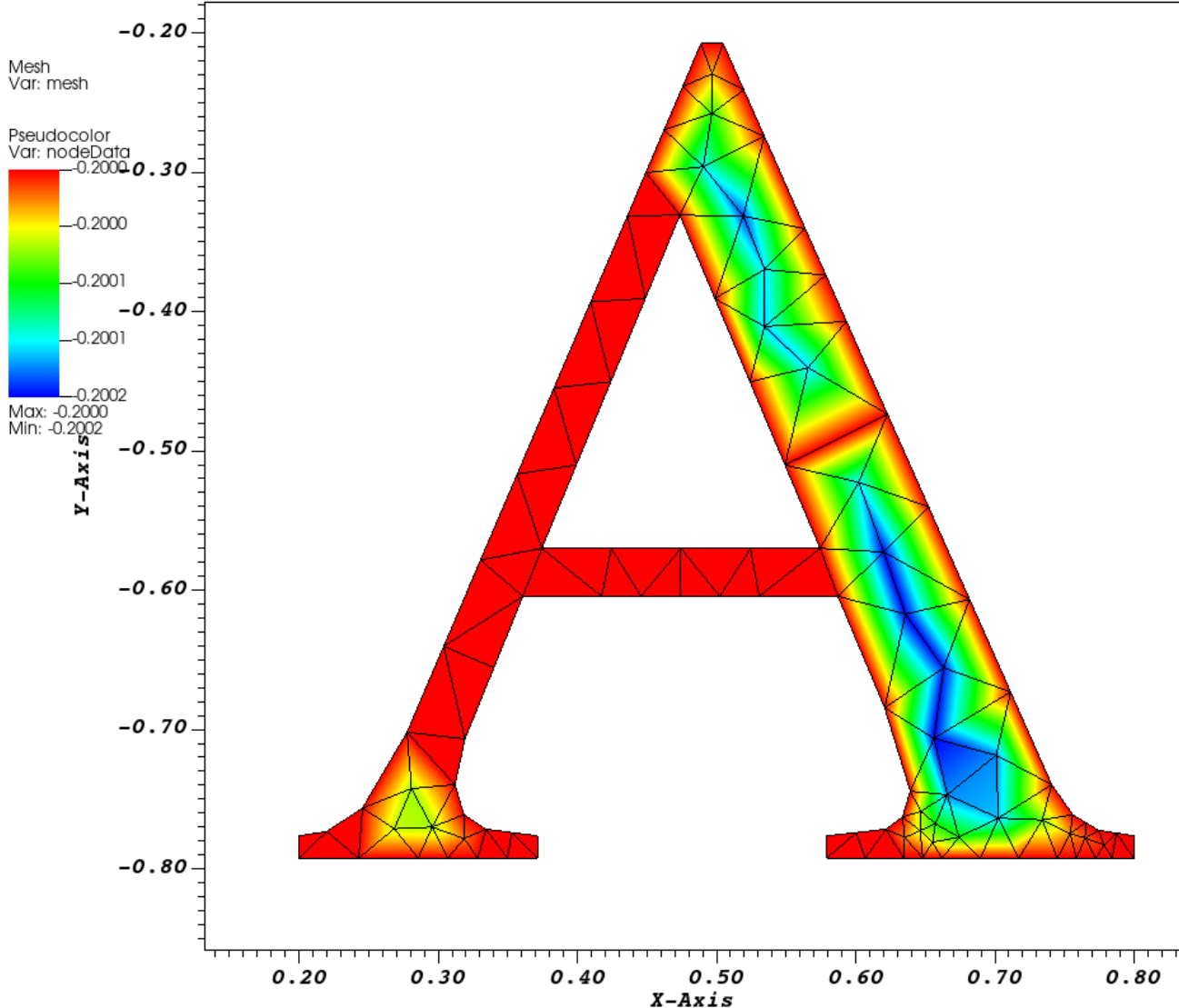
- Solve on *all* nodes (interior + boundary).
- Adjust matrix after assembly for boundary nodes:
 - Set diagonal = 1, other entries = 0.
 - Set RHS = known boundary values.

Effect

- System remains solvable (positive definite, diagonally dominant, not symmetric).

Sample .poly file with boundary conditions

DB: solution.vtk



user: mylesmwathe
Mon Apr 28 23:23:06 2025

4. Demonstrate Piecewise Linear elements converge at 2nd order accuracy

Setup

- Pick smooth Φ , build RHS from $-\Delta\Phi$.
- Enforce Φ on all boundary nodes.

Exact Solution

```
static auto Phi_exact = [](const array<double, DIM> &X) -> double
{
    // smooth function that vanishes on the unit square boundary
    return X[0] * X[0] + X[1] * X[1];
};

static auto source2D = [](const array<double, DIM> &X) -> double
{
    return -4.0;
};
```

Notes: Why still quadratic? 2D Linear function converges too fast to reveal the order.

Workflow

Refine mesh → smaller max element area.

1. Assemble stiffness matrix & load vector.
2. Stamp Dirichlet rows.
3. Solve for Φ_h .
4. Measure nodal max-error.

$$\Phi|_{\partial\Omega} = \Phi, \quad f = -\Delta\Phi, \quad h = 2\sqrt[3]{V}, \quad p \approx \frac{\ln(E_{L-1}/E_L)}{\ln(h_{L-1}/h_L)}.$$

Results

```
Estimated order p:
between lvl 0->1 : p ≈ 2
between lvl 1->2 : p ≈ 2
between lvl 2->3 : p ≈ 0.265492
between lvl 3->4 : p ≈ 2.68471
```

5. Implement a time-dependent FEM solver (Alan)

Solve the differential equation

$$\frac{\partial \phi_h}{\partial t} = f - L_h \phi_h$$

Via backwards euler, so we get the recurrence

$$\left(\frac{1}{\Delta t}I + L\right)(\phi_h(t + \Delta t)) = \frac{1}{\Delta t}\phi_h(t) + f_h(t)$$

L is the finite element approximation of the laplacian (with boundary conditions)

In the next sections, to run the time dependent FEM solver, run `make run_TD` in `exec/`. This will create a database of indexed visit files. The computed solution will be in `exec/vtk_output/solutionXXX.vtk` where `xxx` is the time index of the file. The reference Φ function that we use will be outputted in `exec/phi_output/solutionXXX.vtk`.

6. Verification of Time Dependent FEM Solver

General idea

- Pick a Φ
- Let

$$f = \frac{\partial \Phi}{\partial t} + \Delta \Phi$$

The theoretical solution to the differential equation should converge to the Φ we picked regardless of the initial conditions (if we ignore boundary conditions).

Notes: There is inherent error with a numerical time integrator chasing after the theoretical solution.

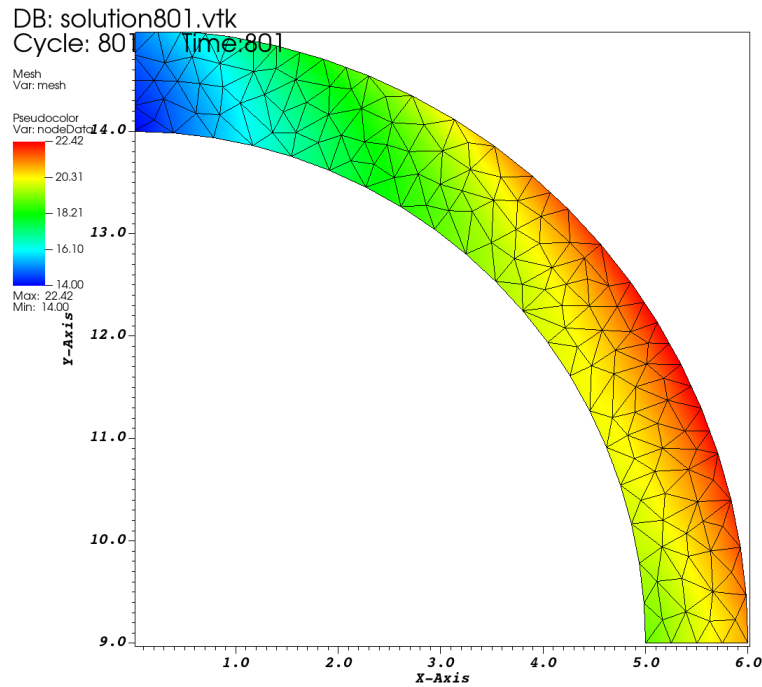
Example1: Time independent Φ

Final relative error **0.009**

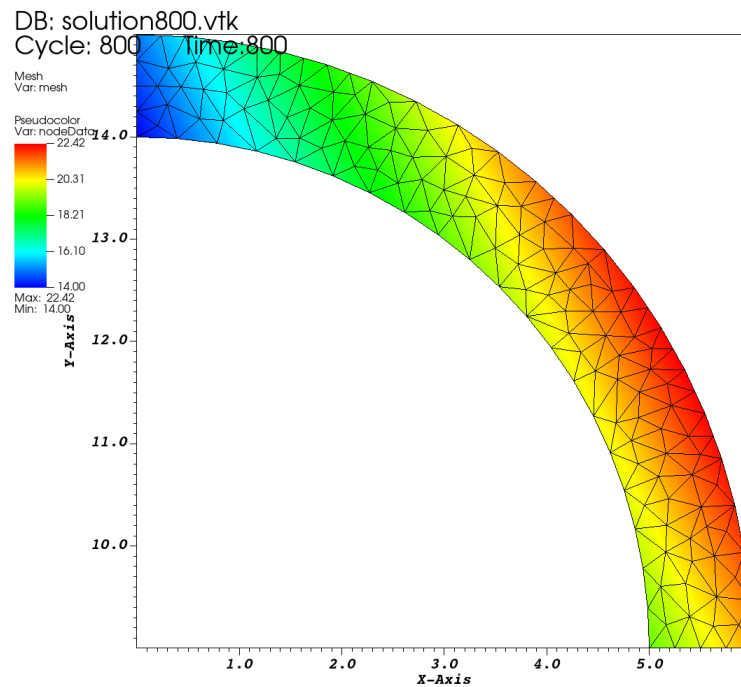
```

11 //our reference phi
12 double sourcePhi(double time, array<double, DIM> x)
13 {
14     return 2*x[0]+x[1];
15 }
16 }
17
18 double derivedf(double time, array<double, DIM> x)
19 {
20     //want to return d^2phi/dx^2 + d^2phi/dy^2 + dphi/dt
21     return 0;
22 }
```

Computed solution



Reference Φ



Example2: Simple time dependent Φ

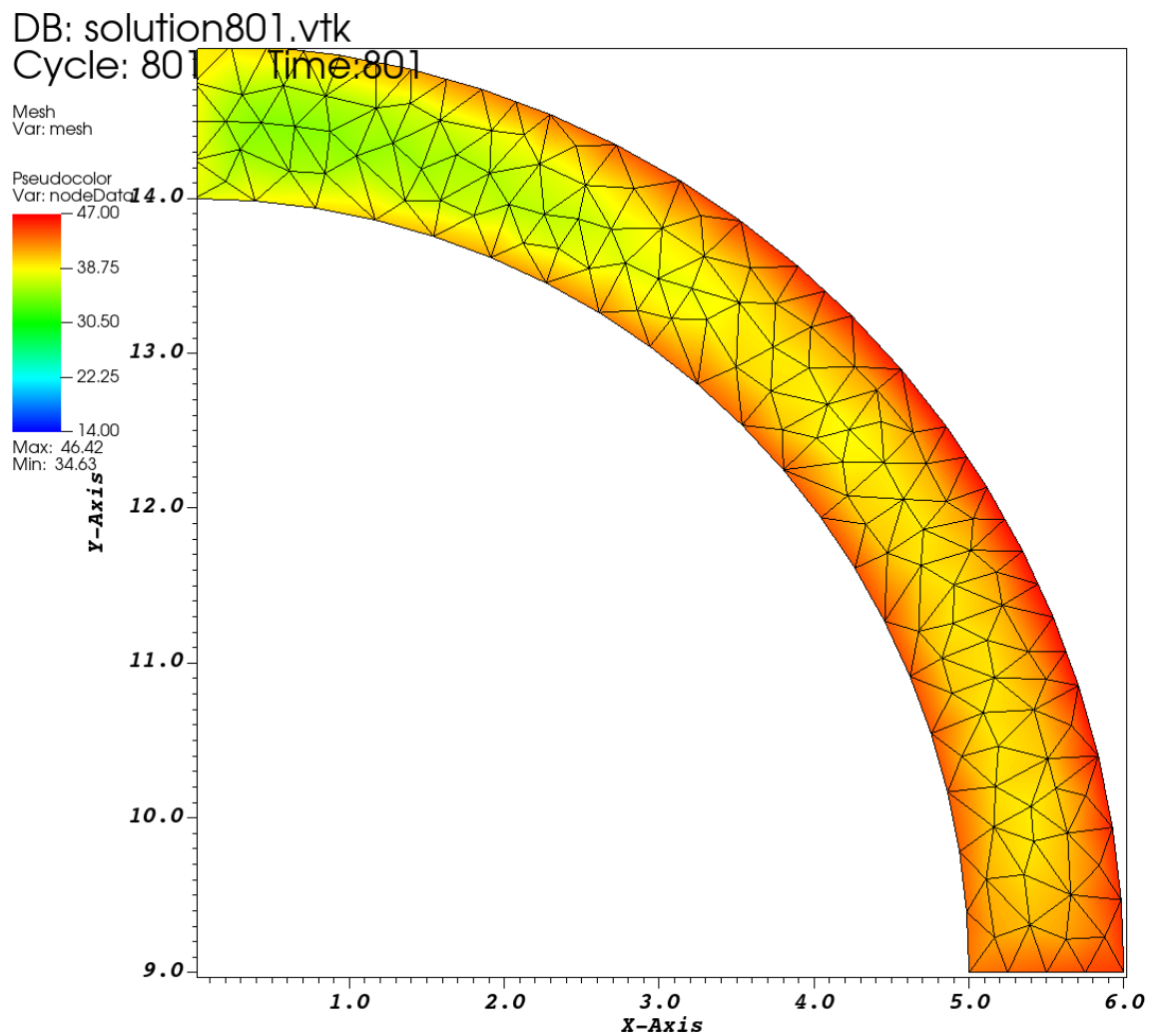
Final relative error **0.1385**

```

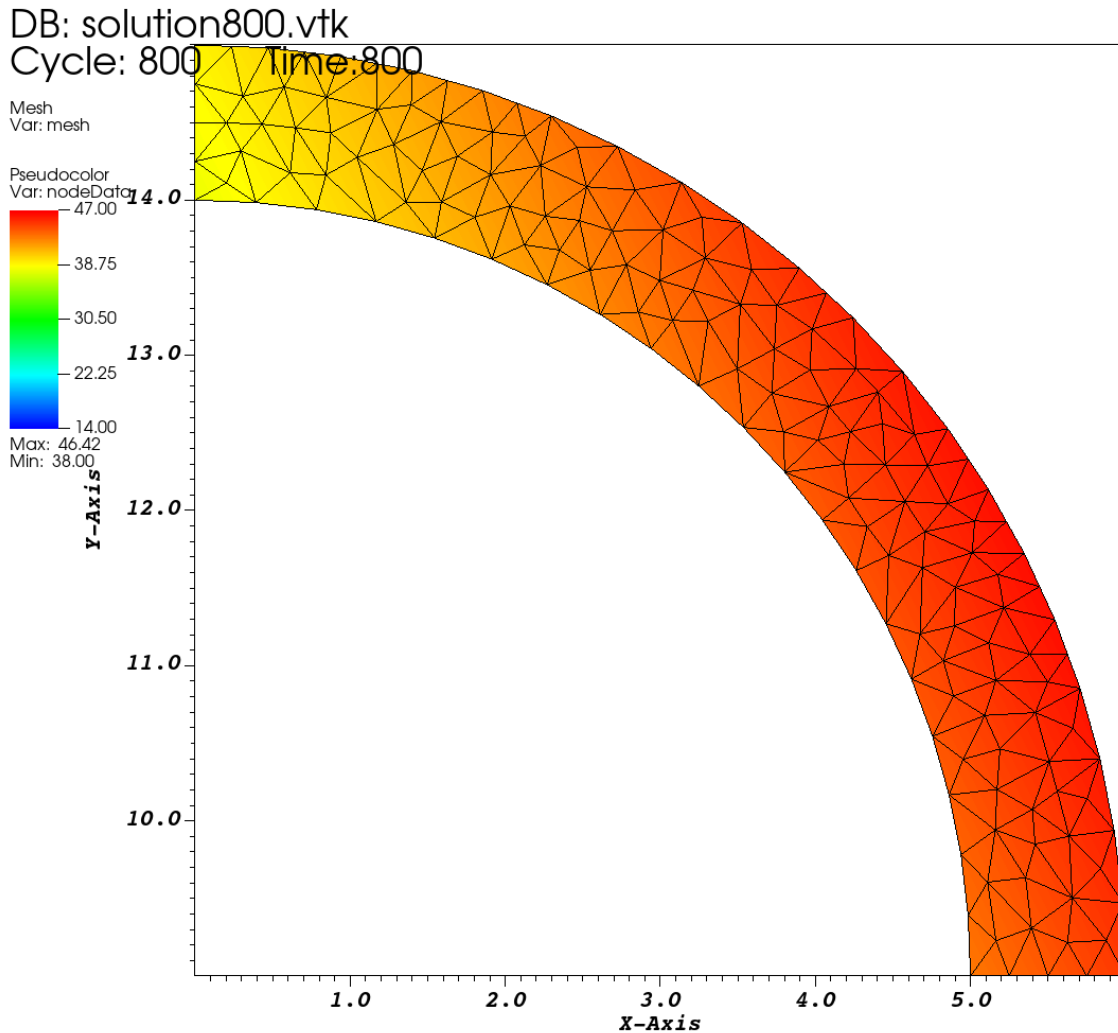
11 //our reference phi
12 double sourcePhi(double time, array<double, DIM> x)
13 {
14     return 2*x[0]+x[1]+3.0*time;
15 }
16
17
18 double derivedf(double time, array<double, DIM> x)
19 {
20     //want to return d^2phi/dx^2 + d^2phi/dy^2 + dphi/dt
21     return 3.0;
22 }

```

Computed solution



Reference Φ



Example3: More complex time dependent Φ

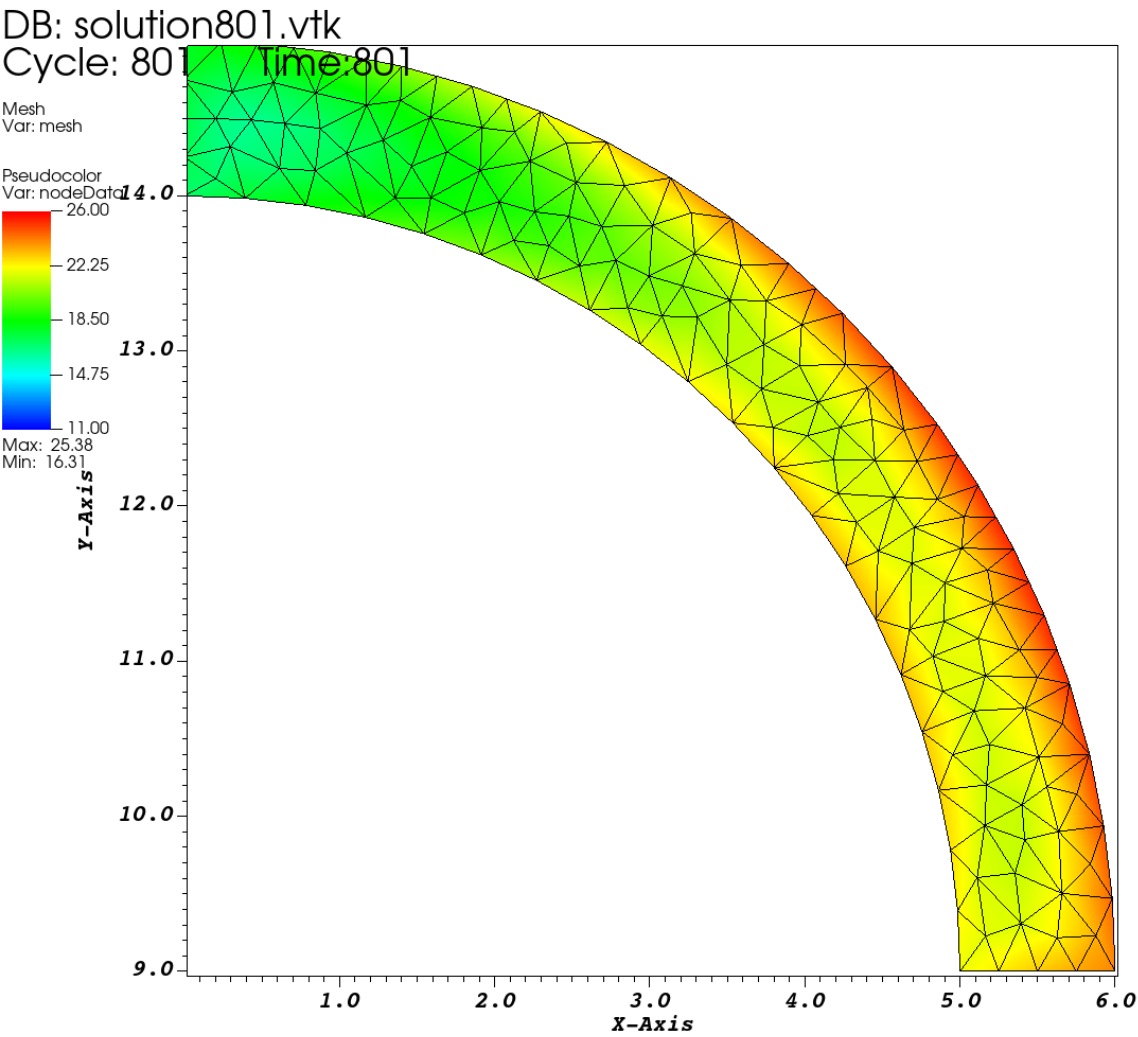
Final relative error **0.1057**

```

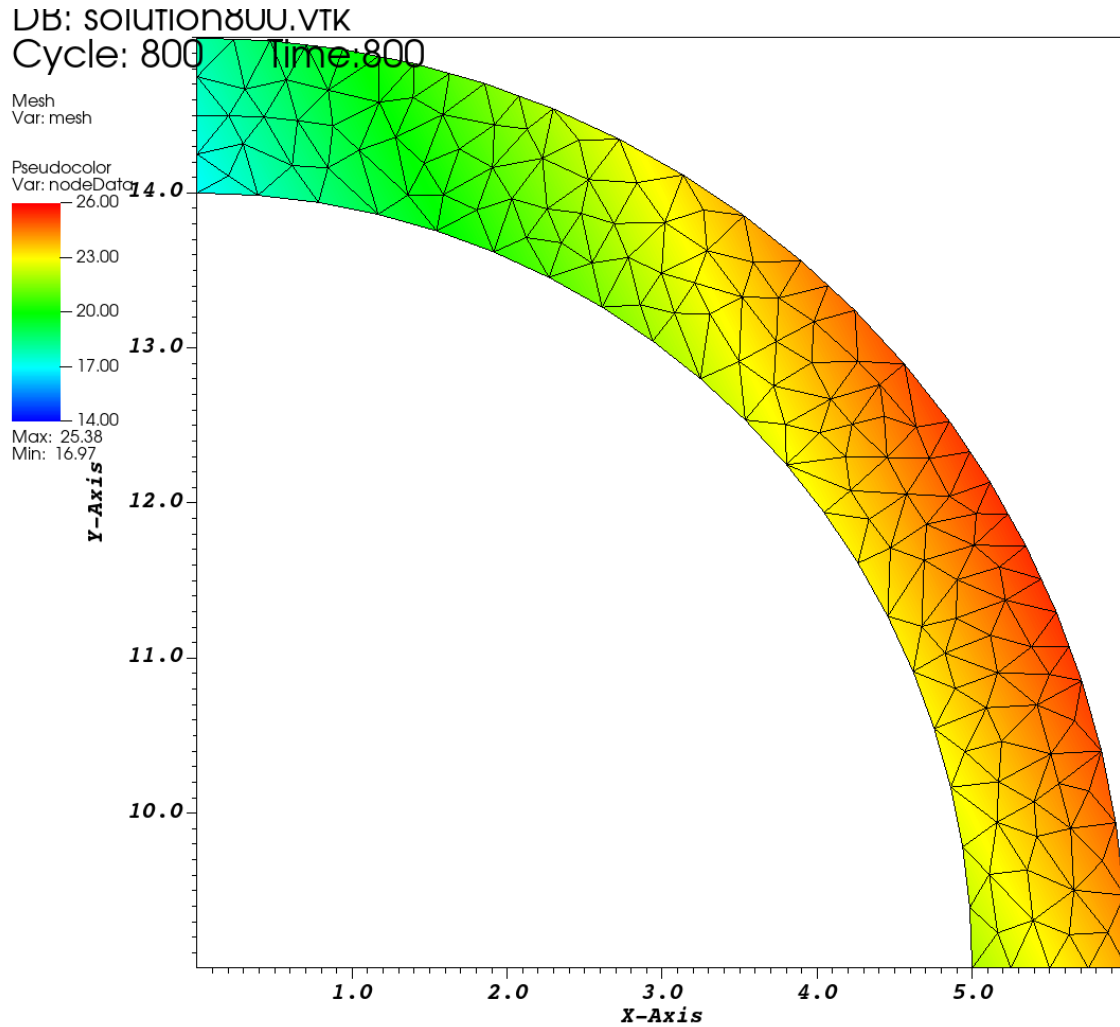
11 //our reference phi
12 double sourcePhi(double time, array<double, DIM> x)
13 {
14     return 2*x[0]+x[1]+3.0*sin(time);
15 }
16
17
18 double derivedf(double time, array<double, DIM> x)
19 {
20     //want to return d^2phi/dx^2 + d^2phi/dy^2 + dphi/dt
21     return 3.0*cos(time);
22 }

```

Computed solution



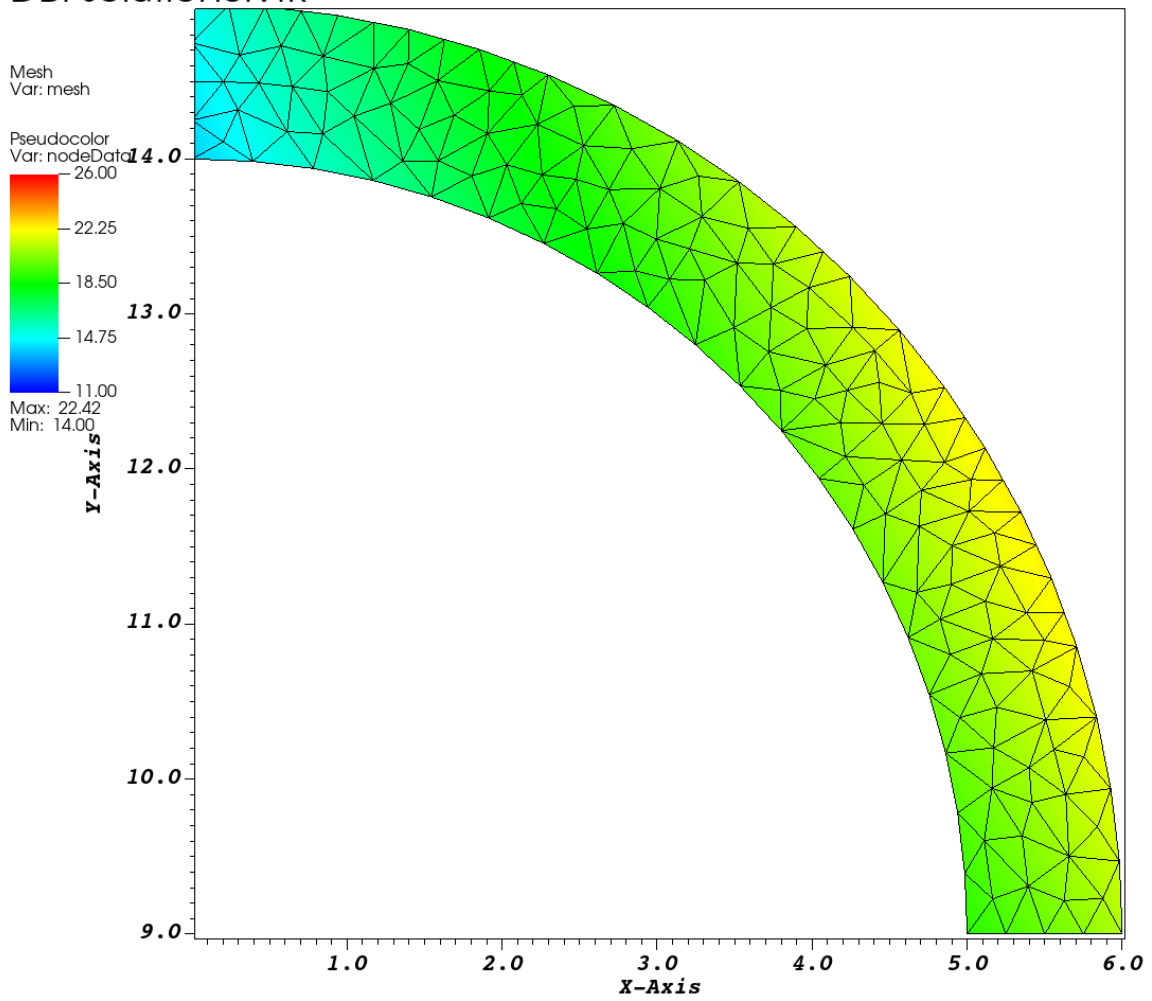
Reference Φ



7. Time-dependent animations

To create a time-dependent animation, run `make run_TD` in `exec/`. Then open visit and open the database in `exec/vtk_output/solutionXXX.vtk`. Then add the mesh and pseudocolor you want to see and use the next arrow to play the animation. For example, with the last example of $\Phi(x, y, t) = 2x + y + 3 \sin t$, some snapshots are shown below:

DB: solution0.vtk



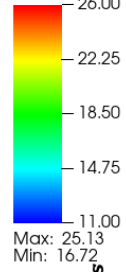
DB: solution200.vtk

DB: SOLUTION200.VTK

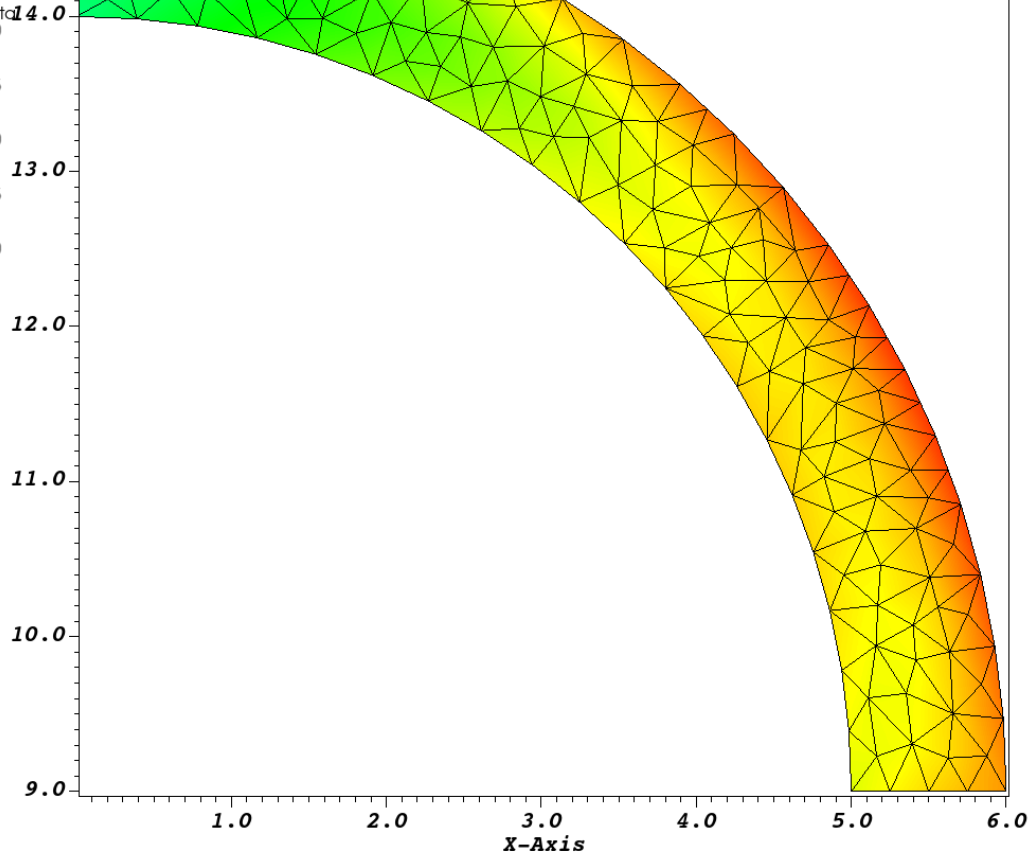
Cycle: 200 Time: 200

Mesh
Var: mesh

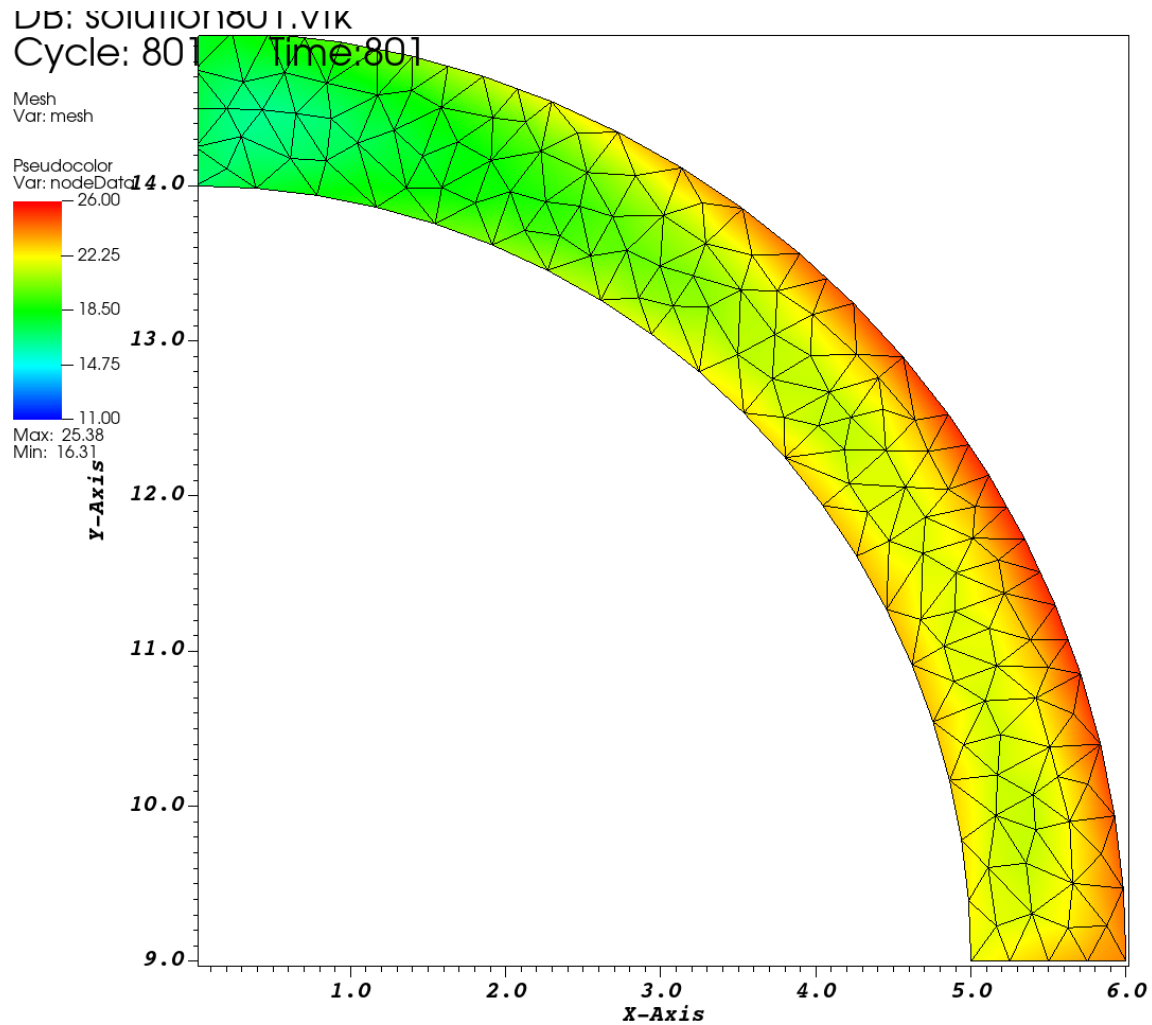
Pseudocolor
Var: nodeData



Y-Axis

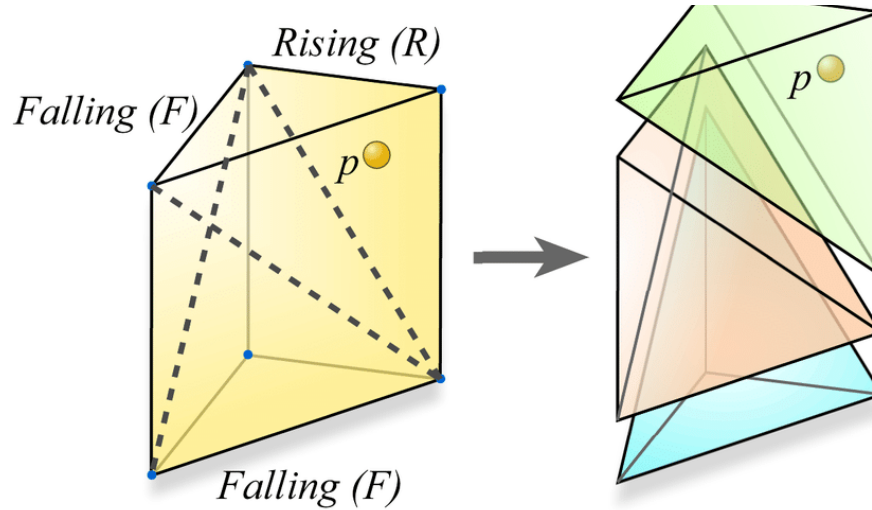


DB: solution201.vtk



8. Extrude the 2D elements into 3D (Yao)

- Extrude each triangle vertically in a prism
- Split prism into 3 tetrahedrons



9. Solve Poisson Equation in 3D (Yao)

2 key differences of another dimension

FEGrid::gradient()

Solve linear system in 3D (inverse matrix: compute determinant, cofactors)

We have

$$dx \times \Delta\phi = \begin{pmatrix} -1 \\ -1 \\ -1 \end{pmatrix} \quad (1)$$

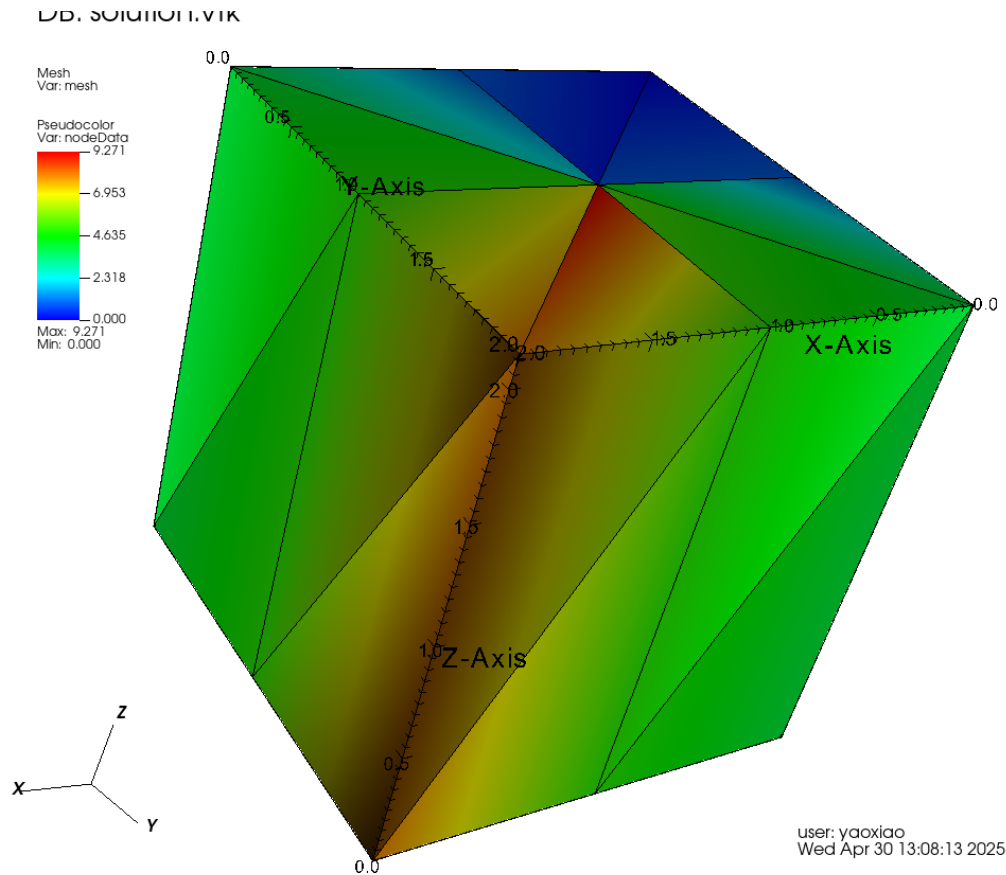
Then solve

$$\Delta\phi = (dx)^{-1} \times \begin{pmatrix} -1 \\ -1 \\ -1 \end{pmatrix} \quad (2)$$

FEGrid::elementArea()

Compute tetrahedron volume instead (value: determinant / 6).

Example extrusion plot of square mesh



10. Verify 3D steady solutions

The process closely resembles the 2D version.

Exact Solution

```
static auto Phi_exact = [](const array<double, DIM> &X) -> double
{
    // smooth function that vanishes on the unit square boundary
    return X[0] + X[1] + X[2];
};

static auto source3D = [](const array<double, DIM> &X) -> double
{
    return 0.0;
};
```

Results

Estimated order p:

between lvl 0→1 : $p \approx -0.0939246$

between lvl 1→2 : $p \approx 1.39756$

between lvl 2→3 : $p \approx 0.307041$

between lvl 3→4 : $p \approx 0.413396$

References

Dompierre, Julien & Labbé, Paul & Vallet, Marie-Gabrielle & Camarero, Ricardo. (1999). How to Subdivide Pyramids, Prisms, and Hexahedra into Tetrahedra.. 195-204.