

SOLUTION of (13-1.o):

Since building an **If::fe::MeshFunctionFE** object requires a scalar-valued finite element space, we have to set up auxiliary objects of type **If::uscalfe::FeSpaceLagrangeO1** (for the pressure) and **If::uscalfe::FeSpaceLagrangeO1** (for the components of the velocity).

Of course, we cannot take for granted anything about the numbering of the global shape functions managed by the **If::assemble::DofHandlers** of those objects. This is why we have to remap the basis expansion coefficients.

C++ code 13.1.38: Sub-problem (13-1.o): Code for writing .vtk file. → [GitLab](#)

```
2  auto mesh_factory = std::make_unique<If::mesh::hybrid2d::MeshFactory>(2);
3  If::io::GmshReader reader(std::move(mesh_factory), meshfile_str);
4
5  const std::shared_ptr<const If::mesh::Mesh> mesh_ptr = reader.mesh();
6  const If::mesh::Mesh& mesh{*mesh_ptr};
7
8  // Initialize dof handler for Taylor-Hood FEM
9  If::assemble::UniformFEDofHandler dofh(mesh_ptr,
10                                         {{If::base::RefEl::kPoint(), 3},
11                                          {If::base::RefEl::kSegment(), 2},
12                                          {If::base::RefEl::kTria(), 0},
13                                          {If::base::RefEl::kQuad(), 0}});
14 // We define function providing the boundary values for the velocity
15 auto g = [](Eigen::Vector2d x) -> Eigen::Vector2d {
16     if ((x[0] < 1E-8) and (x[1] >= 0.5) and (x[1] <= 1.0)) {
17         // Left boundary: inlet, parabolic velocity profile
18         return {(1.0 - x[1]) * (x[1] - 0.5), 0.0};
19     }
20     if ((x[0] > 0.99999) and (x[1] >= 0.0) and (x[1] <= 0.5)) {
21         // Right boundary: outlet
22         return {(0.5 - x[1]) * x[1], 0.0};
23     }
24     return {0.0, 0.0};
25 };
26 // Solve the system
27 Eigen::VectorXd res = StokesPipeFlow::solvePipeFlow(dofh, g);
28
29 double p_diss = 0.0;
30 switch (powerflag) {
31     case VOLUME: {
32         p_diss = compDissPowVolume(dofh, res);
33         break;
34     }
35     case BOUNDARY: {
36         p_diss = compDissPowBd(dofh, res);
37         break;
38     }
39     default: {
40         std::cout << "Dissipated power not computed\n";
41     }
42 }
43
44 std::cout << "Dissipated power: " << p_diss << std::endl;
```

```

46 if (producevtk) {
47     LF_VERIFY_MSG(outfile != nullptr,
48         "Filename for .vtk files has to be provided");
49     // Define first- and second-order Lagrangian FE spaces for the
        piecewise
50     // linear Taylor-Hood pressure approximation and the piecewise
        quadratic
51     auto fes_o1_ptr =
52         std::make_shared<If::uscalfe::FeSpaceLagrangeO1<double>>(mesh_ptr);
53     auto fes_o2_ptr =
54         std::make_shared<If::uscalfe::FeSpaceLagrangeO2<double>>(mesh_ptr);
55     // Fetch dof handler for the components of the velocity
56     const If::assemble::DofHandler& dofh_u = fes_o2_ptr->LocGlobMap();
57     // Fetch dof handler for the pressure p
58     const If::assemble::DofHandler& dofh_p = fes_o1_ptr->LocGlobMap();
59
60     // Coefficient vectors for the first and second component of the
        velocity
61     Eigen::VectorXd coeff_vec_u1(dofh_u.NumDofs());
62     Eigen::VectorXd coeff_vec_u2(dofh_u.NumDofs());
63     // Coefficient vector for the pressure
64     Eigen::VectorXd coeff_vec_p(dofh_p.NumDofs());
65
66     // Loop over vertices and edges to remap the basis expansion
        coefficients
67     for (int codim = 2; codim >= 1; codim--) {
68         for (auto e : mesh.Entities(codim)) {
69             // Global indices for u1, u2 for the respective vertex or edge
70             auto glob_idx = dofh.GlobalDofIndices(*e);
71             auto glob_idx_o2 = dofh_u.GlobalDofIndices(*e)[0];
72
73             // Extract the correct elements for the coefficient matrix of the
        // components of u
74             coeff_vec_u1(glob_idx_o2) = res(glob_idx[0]);
75             coeff_vec_u2(glob_idx_o2) = res(glob_idx[1]);
76
77             // Global indices for p for the respective vertex
78             If::assemble::gdf_idx_t glob_idx_o1 = -1;
79             // The pressure is only defined on vertices
80             if (codim == 2) {
81                 glob_idx_o1 = dofh_p.GlobalDofIndices(*e)[0];
82                 coeff_vec_p(glob_idx_o1) = res(glob_idx[2]);
83             }
84         }
85     }
86 }
87
88 // Define finite-element mesh functions
89 If::fe::MeshFunctionFE<double, double> mf_o2_u1(fes_o2_ptr, coeff_vec_u1);
90 If::fe::MeshFunctionFE<double, double> mf_o2_u2(fes_o2_ptr, coeff_vec_u2);
91 If::fe::MeshFunctionFE<double, double> mf_o1_p(fes_o1_ptr, coeff_vec_p);
92
93 const std::string outfile_str = std::string(outfile);
94 std::cout << outfile_str << std::endl;
95 If::io::VtkWriter vtk_writer(mesh_ptr, outfile_str);
96 vtk_writer.WritePointData("u1", mf_o2_u1);
97 vtk_writer.WritePointData("u2", mf_o2_u2);
98 vtk_writer.WritePointData("p", mf_o1_p);
99 }

```

