

SOLUTION of (12-1.p):

The volume-based formula for the dissipated power in the setting of this project is

$$P_{\text{diss}} = \int_{\Omega} \left| \frac{\partial v_1}{\partial x_2} - \frac{\partial v_2}{\partial x_1} \right|^2 dx, \quad (12.1.39)$$

into which we plug the finite element solution $\mathbf{v}_h \in \mathcal{S}_2^0(\mathcal{M})$.

The policy of the implementation is the same as for the member function `Eval()` in Sub-problem (12-1.h). We encapsulate the gradient of the local shape functions of $\mathcal{S}_2^0(\mathcal{M})$ in lambda functions and use the edge midpoint quadrature rule for the exact evaluation of the local integrals.

C++ code 12.1.40: Sub-problem (12-1.p): Volume formula for dissipated power → [GitLab](#)

```
2 double compDissPowVolume(const If::assemble::DofHandler& dofh,
3                           const Eigen::VectorXd& mu_vec) {
4     // First fetch underlying mesh
5     const If::mesh::Mesh& mesh{ *dofh.Mesh() };
6     // Summation variable
7     double p_diss{0.0};
8     // Loop over all cells and add up contributions of local integrals to
9     // the
10    // dissipated power.
11    for (const If::mesh::Entity* cell : mesh.Entities(0)) {
12        LF_ASSERT_MSG(cell->RefEl() == If::base::RefEl::kTria(),
13                      "Only implemented for triangles");
14        // Obtain coefficients for local shape functions
15        // Local indices for LSFs fpr velocity components
16        std::array<Eigen::Index, 6> vx_idx{0, 3, 6, 9, 11, 13};
17        std::array<Eigen::Index, 6> vy_idx{1, 4, 7, 10, 12, 14};
18        // !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
19        // Workaround for a side-effect in LehrFEM++
20        // Local shape functions associated with edges have to be swapped,
21        // if the
22        // edge has "negative" orientation with respect to the triangle.
23        // This is
24        // meant to offset such a swapping done by LehrFEM++'s
25        // UniformFEDofHandler.
26        auto edge_orientations = cell->RelativeOrientations();
27        LF_ASSERT_MSG(edge_orientations.size() == 3,
28                      "Triangle should have 3 edges!?");
29        for (int k = 0; k < 3; ++k) {
30            if (edge_orientations[k] == If::mesh::Orientation::negative) {
31                // The rows and columns of the element matrix with numbers 9+2*k
32                // and
33                // 9+2*k+1 have to be swapped.
34                std::swap(vx_idx[3 + k], vy_idx[3 + k]);
35            }
36        }
37        // !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
38        std::span<const If::assemble::gdof_idx_t> dof_idx{
39            dofh.GlobalDofIndices(*cell)};
40        LF_ASSERT_MSG(dof_idx.size() == 15, "Taylor-Hood FEM involves 15 LSFs!");
41
42        // Copied from Eval(); Found no good way of moving this to a
43        // function
44        // Area of the triangle
45        double area = If::geometry::Volume(*cell->Geometry());
```

```

40 // Compute gradients of barycentric coordinate functions, see
41 // [Lecture → Code 2.4.5.11] Get vertices of the triangle
42 auto endpoints = lf::geometry::Corners(*(cell->Geometry()));
43 Eigen::Matrix<double, 3, 3> X; // temporary matrix
44 X.block<3, 1>(0, 0) = Eigen::Vector3d::Ones();
45 X.block<3, 2>(0, 1) = endpoints.transpose();
46 // This matrix contains  $\text{grad } \lambda_i$  in its columns
47 const auto G{X.inverse().block<2, 3>(1, 0)};
48 // Gradients of local shape functions of quadratic Lagrangian finite
49 // element
50 // space as lambda functions, see (??)
51 std::array<std::function<Eigen::Vector2d(Eigen::Vector3d)>, 6> gradq{
52     [&G](Eigen::Vector3d c) -> Eigen::Vector2d {
53         return (4 * c[0] - 1) * G.col(0);
54     },
55     [&G](Eigen::Vector3d c) -> Eigen::Vector2d {
56         return (4 * c[1] - 1) * G.col(1);
57     },
58     [&G](Eigen::Vector3d c) -> Eigen::Vector2d {
59         return (4 * c[2] - 1) * G.col(2);
60     },
61     [&G](Eigen::Vector3d c) -> Eigen::Vector2d {
62         return 4 * (c[0] * G.col(1) + c[1] * G.col(0));
63     },
64     [&G](Eigen::Vector3d c) -> Eigen::Vector2d {
65         return 4 * (c[1] * G.col(2) + c[2] * G.col(1));
66     },
67     [&G](Eigen::Vector3d c) -> Eigen::Vector2d {
68         return 4 * (c[2] * G.col(0) + c[0] * G.col(2));
69     }
70 };
71 // Barycentric coordinates of the midpoints of the edges for
72 // use with the 3-point edge midpoint quadrature rule (??)
73 // Note that we only integrate a quadratic polynomial so that the
74 // edge
75 // midpoint quadrature rule evaluates the integral exactly.
76 const std::array<Eigen::Vector3d, 3> mp = { Eigen::Vector3d({0.5, 0.5, 0}),
77                                              Eigen::Vector3d({0, 0.5, 0.5}),
78                                              Eigen::Vector3d({0.5, 0, 0.5}) };
79
80 // We have to integrate  $|\frac{\partial v_2}{\partial x_1} - \frac{\partial v_1}{\partial x_2}|^2$ .
81 // Loop over quadrature points
82 double s = 0.0;
83 for (int qp_idx = 0; qp_idx < 3; ++qp_idx) {
84     // Compute  $\text{grad } v_1$  and  $\text{grad } v_2$  quadrature point
85     Eigen::Vector2d grad_vx{0.0, 0.0};
86     Eigen::Vector2d grad_vy{0.0, 0.0};
87     for (int i = 0; i < 6; ++i) {
88         grad_vx += mu_vec[dof_idx[vx_idx[i]]] * gradq[i](mp[qp_idx]);
89         grad_vy += mu_vec[dof_idx[vy_idx[i]]] * gradq[i](mp[qp_idx]);
90     }
91     double curl_qp = grad_vy[0] - grad_vx[1];
92     s += curl_qp * curl_qp;
93 }
94 p_diss += (area / 3.0 * s);
95
96 }
97
98 return p_diss;
99
100 }

```

