C++ code 3.17.8: Implementation of `edgeResiduals()` ⇝ GitLab

```
lf::mesh::utils::CodimMeshDataSet<double> edgeResiduals(
    const dataDiscreteBVP &disc_bvp, const Eigen::VectorXd &u_vec) {
  // Get pointer to underlying mesh
  std::shared_ptr<const lf::mesh::Mesh> mesh_p =
      disc_bvp.pwlinfespace_p_->Mesh();
  // Mesh data set for returning the volume residuals
  lf::mesh::utils::CodimMeshDataSet<double> edge_res(mesh_p, 1, 0.0);
  // Step I: Create an auxiliary MeshDataSet storing non-normalized (!) edge
  // normals = edge direction vectors turned by 90 degrees
  lf::mesh::utils::CodimMeshDataSet<Eigen::Vector2d> edge_normals(mesh_p, 1);
  lf::mesh::utils::CodimMeshDataSet<Eigen::Vector2d> edge_startpt(mesh_p, 1);
  for (const lf::mesh::Entity *edge : mesh_p->Entities(1)) {
    // Obtain information about the shape of the edge
    const lf::geometry::Geometry &geo{*(edge->Geometry())};
    const Eigen::MatrixXd corners{lf::geometry::Corners(geo)};
    // Starting point of the edge
    edge_startpt(*edge) = corners.col(0);
    // Direction vector of the edge
    const Eigen::Vector2d dir = corners.col(1) - corners.col(0);
    // Rotate counterclockwise by 90 degrees an d store
    edge_normals(*edge) = Eigen::Vector2d(dir(1), -dir(0));
  }
  // Step II: Compute the gradient of the finite element solution, which
  // LehrFEM++ can do for us.
  const lf::fe::MeshFunctionGradFE grad_uh(disc_bvp.pwlinfespace_p_, u_vec);

  // Step III: Visit the triangles of the mesh, fetch the constant(!) gradient
  // of the finite-element solution, multiply it with the edge normals and the
  // coefficient and sum the results for every edge not loacted on the
  // bounmdary.  This summation is done in an auxiliary MeshDataSet.
  lf::mesh::utils::CodimMeshDataSet<double> alpha_max(mesh_p, 1, 0.0);
  lf::mesh::utils::CodimMeshDataSet<double> edge_flux_jump(mesh_p, 1, 0.0);
  lf::mesh::utils::CodimMeshDataSet<bool> bd_ed_flags{
      lf::mesh::utils::flagEntitiesOnBoundary(mesh_p, 1)};
  for (const lf::mesh::Entity *cell : mesh_p->Entities(0)) {
    LF_ASSERT_MSG(cell->RefEl() == lf::base::RefEl::kTria(),
                  "Implemented for triangles only");
    // Retrieve constant diffusion coefficient
    const Eigen::MatrixXd dummy = Eigen::Vector2d(1.0 / 3.0, 1.0 / 3.0);
    const double alpha_K = disc_bvp.mf_alpha_(*cell, dummy)[0];
    LF_ASSERT_MSG(alpha_K > 0,
                  "Diffusion coefficients must be strictly positive!");
    // Fetch scaled gradient of the finite element solution
    const Eigen::Vector2d nablau_K = (grad_uh(*cell, dummy)[0]) * alpha_K;
    /* Debugging output
    std::cout << "cell " << mesh_p->Index(*cell)
              << ": alpha*grad u_h = " << nablau_K.transpose() << std::endl;
    */
    // Obtain shape of cell
    const lf::geometry::Geometry &cell_geo{*(cell->Geometry())};
    const Eigen::MatrixXd cell_vert{lf::geometry::Corners(cell_geo)};
    // Visit all three edges of the triangle
    std::span<const lf::mesh::Entity *const> edges{cell->SubEntities(1)};
    LF_ASSERT_MSG(edges.size() == 3, "Triangle must have three edges!");
```