C++ code 2.4.15: Implementation of `solveA()` → GitLab

```cpp
template <typename FUNCTOR1, typename FUNCTOR2>
Eigen::VectorXd solveA(const Eigen::VectorXd &mesh, FUNCTOR1 &&gamma,
                       FUNCTOR2 &&f) {
  // Nodes are indexed as 0=x_0 < x_1 < ... < x_N = 1
  // Note: What is called N here, is M in the project description!
  unsigned N = mesh.size() - 1;
  // Initializations (notice initialization with zeros here)
  Eigen::VectorXd u = Eigen::VectorXd::Zero(N + 1);  // solution vec
  Eigen::SparseMatrix<double> A(N + 1, N + 1);       // laplacian galerkin mat
  Eigen::SparseMatrix<double> M(N + 1, N + 1);       // mass galerkin mat
  Eigen::SparseMatrix<double> L(N + 1, N + 1);       // full galerkin mat

  // I. Build the (full) Galerkin matrix L for the lin. sys.
  // I.i Compute the entries of the Laplace Galerkin matrix A
  // (const. ceoff. func. alpha = 1.0)
  std::vector<Eigen::Triplet<double>> triplets_A =
      computeA(mesh, [](double) -> double { return 1.0; });
  // I.ii Compute the entries of the mass matrix M
  std::vector<Eigen::Triplet<double>> triplets_M = computeM(mesh, gamma);
  // I.iii Assemble the sparse matrices
  A.setFromTriplets(triplets_A.begin(), triplets_A.end());
  M.setFromTriplets(triplets_M.begin(), triplets_M.end());
  L = A + M;  // Full Galerkin matrix of the LSE \Label[line]{SAApM}

  // II. Build the right hand side source vector
  Eigen::VectorXd rhs_vec = computeRHS(mesh, f);

  // III. Enforce (zero) dirichlet boundary conditions
  // The B.C. are u(0) = u(N) = 0. The boundary nodes are indexed by 0
  // and N. We can therefore opt for the option of droping first and last rows
  // and columns of L, along with the first and last entries of the rhs_vec.
  Eigen::SparseMatrix<double> L_reduced = L.block(1, 1, N - 1, N - 1);
  Eigen::VectorXd rhs_vec_reduced = rhs_vec.segment(1, N - 1);

  // IV. Solve the LSE L*u = rhs_vec using an Eigen solver
  Eigen::SimplicialLDLT<Eigen::SparseMatrix<double>, Eigen::Lower> solver;
  solver.compute(L_reduced);
  if (solver.info() != Eigen::Success) {
    throw std::runtime_error("Could not decompose the matrix");
  }
  u.segment(1, N - 1) = solver.solve(rhs_vec_reduced);

  // The solution vector u was initialized with zeros, and therefore already
  // contains the zero Dirichlet boundary data in the first and last entry
  return u;
}  // solveA
```