C++ code 2.4.13: Implementation of `computeM()` ↪ GitLab

```
template <typename FUNCTOR1>
std::vector<Eigen::Triplet<double>> computeM(const Eigen::VectorXd &mesh,
                                             FUNCTOR1 &&gamma) {
  // Nodes are indexed as 0=x_0 < x_1 < ... < x_N = 1
  unsigned N = mesh.size() - 1;

  // The basis tent functions spanning the space of continuous piecewise
  // linear polynomial satisfy the cardinal basis property with respect
  // to the nodes of the mesh. Using  the trapezoidal rule to approximate
  // the integrals therefore lead to a diagonal (N+1) x (N+1) matrix
  std::vector<Eigen::Triplet<double>> triplets;
  triplets.reserve(N + 1);

  // Some tool variables
  double diag, off_diag;
  double dx;  // cell widths

  /* Computing diagonal entries */
  // First diagonal entry (left boundary node)
  dx = mesh(1) - mesh(0);
  diag = gamma(mesh(0)) * 0.5 * dx;
  triplets.push_back(Eigen::Triplet<double>(0, 0, diag));
  // Last diagonal entry (right boundary node)
  dx = mesh(N) - mesh(N - 1);
  diag = gamma(mesh(N)) * 0.5 * dx;
  triplets.push_back(Eigen::Triplet<double>(N, N, diag));
  // All diagonal entries associated to interior nodes
  for (unsigned i = 1; i < N; ++i) {
    dx = mesh(i + 1) - mesh(i - 1);
    diag = gamma(mesh(i)) * 0.5 * dx;
    triplets.push_back(Eigen::Triplet<double>(i, i, diag));
  }  // computeM

  return triplets;
}  // computeM
```