C++ code 9.6.17: Implementation of `solveBoundaryWave()` ⇢ GitLab

```
template <typename FUNCTOR_U, typename FUNCTOR_V>
Eigen::VectorXd solveBoundaryWave(
    const std::shared_ptr<lf::uscalfe::FeSpaceLagrangeO1<double>> &fe_space_p,
    FUNCTOR_U &&u0, FUNCTOR_V &&v0, double T, unsigned int N) {
  Eigen::VectorXd bdyWaveSol;

  double step_size = T / N;
  // Obtain initial data
  std::pair<Eigen::VectorXd, Eigen::VectorXd> initialData =
      interpolateInitialData<std::function<double(Eigen::Vector2d)>,
                             std::function<double(Eigen::Vector2d)>>(
          fe_space_p, std::forward<FUNCTOR_U>(u0), std::forward<FUNCTOR_V>(v0));
  // Obtain Galerkin matrices
  lf::assemble::COOMatrix<double> M = buildM(fe_space_p);
  lf::assemble::COOMatrix<double> A = buildA(fe_space_p);
  // Convert COO matrix M and A into CRS format using Eigen's internal
  // conversion routines.
  Eigen::SparseMatrix<double> M_sparse = M.makeSparse();
  Eigen::SparseMatrix<double> A_sparse = A.makeSparse();
  // Compute LU decomposition of coefficient matrix of LSE to be solved in every
  // step of Crank-Nicolson timestepping
  Eigen::SparseLU<Eigen::SparseMatrix<double>> solver;
  solver.compute(M_sparse + 0.25 * (step_size * step_size) * A_sparse);
  LF_VERIFY_MSG(solver.info() == Eigen::Success,
                "LU decomposition of M failed");

  // Crank-Nicolson timestepping, first-order-system version
  Eigen::VectorXd u_cur = initialData.first;
  Eigen::VectorXd v_cur = initialData.second;
  Eigen::VectorXd u_next, v_next;
  for (int i = 1; i < N + 1; i++) {
    // step foward
    v_next = solver.solve(M_sparse * v_cur -
                          0.25 * (step_size * step_size) * A_sparse * v_cur -
                          step_size * A_sparse * u_cur);
    u_next = u_cur + 0.5 * step_size * (v_cur + v_next);
    // update
    v_cur = v_next;
    u_cur = u_next;
  }
  bdyWaveSol = u_cur;
  return bdyWaveSol;
};
```