C++11 code 9.5.20: Implementation of function `solvewave()` ⇢ GitLab

```
template <typename FUNCTION>
std::pair<Eigen::VectorXd, Eigen::VectorXd> solvewave(
    std::shared_ptr<lf::uscalfe::UniformScalarFESpace<double>> fes_p,
    FUNCTION c, const Eigen::VectorXd &u0_vec, const Eigen::VectorXd &v0_vec,
    double T, unsigned int m) {
  std::pair<Eigen::VectorXd, Eigen::VectorXd> solution_pair;
  double tau = T / m;  // time step
  std::cout << "Solving with uniform step size tau = " << tau << std::endl;
  progress_bar progress{std::clog, 70u, "Timestepping"};
  double progress_pourcentage;
  // Obtain local->global index mapping for current finite element space
  const lf::assemble::DofHandler &dofh{fes_p->LocGlobMap()};
  const lf::uscalfe::size_type N_dofs(dofh.NumDofs());  // dim.  of FE space size
  LF_VERIFY_MSG(u0_vec.size() == N_dofs, "Wrong size of initial conditions u0");
  LF_VERIFY_MSG(v0_vec.size() == N_dofs, "Wrong size of initial conditions");

  // Precomputing the required data for symplectic time stepping
  SympTimestepWaveEq<decltype(c)> timestepper(fes_p, c);

  /* Starting the evolution using the initial conditions */
  Eigen::VectorXd q = u0_vec;
  Eigen::VectorXd p = v0_vec;
  // Initialization to store energy at different times
  Eigen::VectorXd energies(m + 1);

  /* Iterating symplectic stepping */
  for (int i = 0; i < m; i++) {
    energies[i] = timestepper.computeEnergies(p, q);
    timestepper.compTimestep(tau, p, q);
    // Throw an exception in case of severe increase of the total
    // energy, which is a conserved quantity for the exact evolution.
    if (energies[i] > 10.0 * energies[0]) {
      throw "ENERGY BLOW UP";
    }
    progress_pourcentage = ((double)i) / m * 100.0;
    progress.write(progress_pourcentage / 100.0);
  }

  energies[m] = timestepper.computeEnergies(p, q);
  solution_pair = std::make_pair(q, energies);
  return solution_pair;
}
```