C++ code 9.13.30: Implementation of `solve1DWavePML()`

```
std::vector<double> trackEnergy(const Eigen::VectorXd &zeta_0,
                                const Eigen::VectorXd &gamma,
                                const Eigen::VectorXd &sigma, unsigned int M,
                                double T) {
  // Grid resolution parameter N = number of grid nodes - 1
  const unsigned int N = gamma.size() - 1;
  const double L = L_default;
  const double h = (2.0 + 2 * L) / N;
  const double tau = T / M;
  // Storage for energies
  std::vector<double> en{};
  // Temporary storage for solution
  Eigen::VectorXd zeta_old(2 * N + 1);
  // Flag telling recorder lambda function that we are at the first
     timestep
  bool first_step = true;
  // Recorder lambda function
  auto rec = [&](const Eigen::VectorXd &zeta) -> void {
    if (first_step) {
      first_step = false;
    } else {
      auto delta = zeta - zeta_old;
      // Norm of discrete temporal derivative of u-component
      double mu_d_norm = 0.5 * delta[0] * delta[0];
      for (unsigned int i = 1; i < N; ++i) {
        mu_d_norm += delta[i] * delta[i];
      }
      mu_d_norm += 0.5 * delta[N] * delta[N];
      mu_d_norm *= 0.5 * h / (tau * tau);
      // Norm of discrete temporal derivative of v-component
      double nu_d_norm = 0.0;
      for (unsigned int i = 0; i < N; ++i) {
        nu_d_norm += (delta[i + N + 1] * delta[i + N + 1]) / gamma[i];
      }
      nu_d_norm *= 0.5 * h / (tau * tau);
      en.push_back(nu_d_norm + mu_d_norm);
    }
    zeta_old = zeta;
  };
  (void)solve1DWavePML(zeta_0, gamma, sigma, M, T, rec);
  return en;
}
```