```
// manually enforce Dirichlet boundary conditions
for( unsigned int face = 0; face < GeometryInfo< dim >::faces_per_cell; ++face )
{

    if(cell->face(face)->at_boundary() && (cell->face(face)->boundary_id() == 0))
    {
          // initialization
          scratch.fe_face_values.reinit(cell, face);
          scratch.face_mass = 0.0;
          scratch.face_mass.diagadd(1.0);
          scratch.dirichlet_rhs      = 0.0;
          scratch.proj_dirichlet_rhs = 0.0;

          // set up face_mass
          for (unsigned int i = 0;
                i < scratch.fe_support_on_face[face].size();
                ++i)
          {
                for (unsigned int j = 0;
                      j < scratch.fe_support_on_face[face].size();
                      ++j)
                {
                      const unsigned int ii =
                            scratch.fe_support_on_face[face][i];
                      const unsigned int jj =
                            scratch.fe_support_on_face[face][j];
                      scratch.face_mass(ii, jj) = 0.0;
                }
          }

          // construct mass matrix on the boundary and rhs
          for( unsigned int q = 0; q < n_face_q_points; ++q )
          {
                for (unsigned int k = 0; k < scratch.fe_support_on_face[face].size(); ++k)
                      scratch.tr_phi[k] = scratch.fe_face_values.shape_value(
                                                scratch.fe_support_on_face[face][k], q);
                      const Point<dim> quadrature_point =
                                  scratch.fe_face_values.quadrature_point(q);

                // computing face mass
                for (unsigned int i = 0;
                     i < scratch.fe_support_on_face[face].size();
                     ++i)
                  for (unsigned int j = 0;
                       j < scratch.fe_support_on_face[face].size();
                       ++j)
                  {
                    const unsigned int ii =
                      scratch.fe_support_on_face[face][i];
                    const unsigned int jj =
                      scratch.fe_support_on_face[face][j];
```

```
                        scratch.face_mass(ii, jj) += scratch.tr_phi[i] * scratch.tr_phi[j] *
scratch.fe_face_values.JxW( q );
                      }

                  // computing rhs via projection
                  for (unsigned int i = 0;
                       i < scratch.fe_support_on_face[face].size();
                       ++i)
                  {
                    const unsigned int ii =
                       scratch.fe_support_on_face[face][i];
                    scratch.dirichlet_rhs(ii) += scratch.exact_solution.value(quadrature_point)
* scratch.tr_phi[i] * scratch.fe_face_values.JxW( q );
                  }

               }

                    // obtain projected boundary data
                    scratch.face_mass.gauss_jordan();
                    scratch.face_mass.vmult(scratch.proj_dirichlet_rhs, scratch.dirichlet_rhs);

                    // assign the projected boundary data to the cell_vector and make
cell_matrix diagonal
               for (unsigned int i = 0;
                    i < scratch.fe_support_on_face[face].size();
                    ++i)
                    for (unsigned int j = 0;
                          j < scratch.fe_support_on_face[face].size();
                          ++j)
                    {
                          const unsigned int ii =
                                scratch.fe_support_on_face[face][i];
                          if (i == j)
                          {
                                task_data.cell_vector(ii) = scratch.proj_dirichlet_rhs(ii) *
task_data.cell_matrix(ii,ii);
                          }else{
                                const unsigned int jj =
                                      scratch.fe_support_on_face[face][j];
                                task_data.cell_matrix(ii,jj) = 0.0;
                          }
                    }
          }
      } // end of enforcement of Dirichlet BC
```