The Following is bilinear form of a large strain elasticity problem based on step-44:

The linearised approximation to nonlinear governing equations to be solved using the Newton-Raphson method is:

Find

In which:

The following is the assembly part of the code

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

template <int dim>

void Solid<dim>::assemble\_system ()

{

tangent\_matrix = 0;

system\_rhs = 0;

FEValues<dim> fe\_values (fe, qf\_cell,

update\_values | update\_gradients |

update\_quadrature\_points | update\_JxW\_values);

FEFaceValues<dim> fe\_face\_values (fe, qf\_face,

update\_values | update\_quadrature\_points |

update\_normal\_vectors | update\_JxW\_values);

FullMatrix<double> cell\_matrix (dofs\_per\_cell, dofs\_per\_cell);

Vector<double> cell\_rhs (dofs\_per\_cell);

std::vector<types::global\_dof\_index> local\_dof\_indices (dofs\_per\_cell);

std::vector<double> Nx(dofs\_per\_cell);

std::vector<Tensor<2, dim> > grad\_Nx(dofs\_per\_cell);

std::vector<SymmetricTensor<2, dim> > symm\_grad\_Nx(dofs\_per\_cell);

typename DoFHandler<dim>::active\_cell\_iterator

cell = dof\_handler.begin\_active(),

endc = dof\_handler.end();

for (; cell!=endc; ++cell)

if (cell->is\_locally\_owned())

{

fe\_values.reinit (cell);

cell\_matrix = 0;

cell\_rhs = 0;

PointHistory<dim> \*lqph =

reinterpret\_cast<PointHistory<dim>\*>(cell->user\_pointer());

for (unsigned int q\_point=0; q\_point<n\_q\_points; ++q\_point)

{

const Tensor<2, dim> F\_inv = lqph[q\_point].get\_F\_inv();

const Tensor<2, dim> tau = lqph[q\_point].get\_tau();

const SymmetricTensor<2, dim> symm\_tau = lqph[q\_point].get\_tau();

const SymmetricTensor<4, dim> Jc = lqph[q\_point].get\_Jc();

const double JxW = fe\_values.JxW(q\_point);

for (unsigned int k=0; k<dofs\_per\_cell; ++k)

{

grad\_Nx[k] = fe\_values[u\_fe].gradient(k, q\_point) \* F\_inv;

symm\_grad\_Nx[k] = symmetrize(grad\_Nx[k]);

}

for (unsigned int i=0; i<dofs\_per\_cell; ++i)

{

const unsigned int component\_i = fe.system\_to\_component\_index(i).first;

for (unsigned int j=0; j<dofs\_per\_cell; ++j)

{

const unsigned int component\_j = fe.system\_to\_component\_index(j).first;

cell\_matrix(i, j) += symm\_grad\_Nx[i] \* Jc // The material contribution:

\* symm\_grad\_Nx[j] \* JxW;

if (component\_i == component\_j) // geometrical stress contribution

cell\_matrix(i, j) += grad\_Nx[i][component\_i] \* tau

\* grad\_Nx[j][component\_j] \* JxW;

}

cell\_rhs(i) -= symm\_grad\_Nx[i] \* symm\_tau \* JxW;

}

}

cell->get\_dof\_indices (local\_dof\_indices);

constraints.distribute\_local\_to\_global (cell\_matrix,

cell\_rhs,

local\_dof\_indices,

tangent\_matrix,

system\_rhs);

}

tangent\_matrix.compress (VectorOperation::add);

system\_rhs.compress (VectorOperation::add);

}