equation, called the mixed formulation:

$$
\begin{aligned}
K^{-1}\mathbf{u} - \quad p &= 0 && \text{in } \Omega,\\
-\mathrm{div}\ \mathbf{u} &= 0 && \text{in } \Omega,\\
p &= g && \text{on } \partial\Omega.
\end{aligned}
$$

The weak formulation of this problem is found by multiplying the two equations with test functions and integrating some terms by parts:

$$A(\{\mathbf{u}, p\}, \{\mathbf{v}, q\}) = F(\{\mathbf{v}, q\}),$$

where

$$
\begin{aligned}
A(\{\mathbf{u}, p\}, \{\mathbf{v}, q\}) &= (\mathbf{v}, K^{-1}\mathbf{u})_\Omega - (\mathrm{div}\ \mathbf{v}, p)_\Omega - (q, \mathrm{div}\ \mathbf{u})_\Omega\\
F(\{\mathbf{v}, q\}) &= -(g, \mathbf{v}\cdot\mathbf{n})_{\partial\Omega} - (f, q)_\Omega.
\end{aligned}
$$

Here, $\mathbf{n}$ is the outward normal vector at the boundary. Note how in this formulation, Dirichlet boundary values of the original problem are incorporated in the weak form.

To be well-posed, we have to look for solutions and test functions in the space $H(\mathrm{div}) = \{\mathbf{w} \quad L^2(\Omega)^d : \mathrm{div}\ \mathbf{w} \quad L^2\}$ for $\mathbf{u}, \mathbf{v}$, and L^2 for p, q. It is a well-known fact stated in almost every book on finite element theory that if one chooses discrete finite element spaces for the approximation of $\mathbf{u}, p$ inappropriately, then the resulting discrete saddle-point problem is instable and the discrete solution will not converge to the exact solution.

Vector-valued elements have already been discussed in previous tutorial programs, the first time and in detail in step-8. The main diﬀerence there was that the vector-valued space V_h is uniform in all its components: the *dim* components of the displacement vector are all equal and from the same function space. What we could therefore do was to build V_h as the outer product of the *dim* times the usual $Q(1)$ finite element space, and by this make sure that all our shape functions have only a single non-zero vector component. Instead of dealing with vector-valued shape functions, all we did in step-8 was therefore to look at the (scalar) only non-zero component and use the `fe.system_to_component_`

```
  return tmp;
}
```

What this function does is, given an `fe_values`

```
{
  const Tensor<1,dim>
    phi_i_u = extract_u (fe_face_values, i, q);
```

Here, the matrix $S = BM^{-1}B^T$ (called the *Schur complement* of A) is obviously symmetric and, owing to the positive definiteness of

```
}
```

defining P and U

A preconditioner for the Schur complement

One may ask whether it would help if we had a preconditioner for the Schur complement $S = BM^{-1}B^T$

```
    approximate_schur_complement (system_matrix);

  InverseMatrix<ApproximateSchurComplement>
    preconditioner (approximate_schur_complement)
```

That's all!

Taken together, the first block of our `solve()` function will then look like this:

```
  Vector<double> schur_rhs (solution.block(1).size());

  m_inverse.vmult (tmp, system_rhs.block(0));
  system_matrix.block(1,0).vmult (schur_rhs, tmp);
  schur_rhs -= system_rhs.block(1);

  SchurComplement
    schur_complement (system_matrix, m_inverse);

  ApproximateSchurComplement
    approximate_schur_complement (system_matrix);

  InverseMatrix<ApproximateSchurComplement>
    preconditioner (approximate_schur_complement);

  SolverControl solver_control (system_matrix.block(0,0).m(),
                                1e-6*schur_rhs.l2_norm());
  SolverCG<>    cg (solver_control);

  cg.solve (schur_complement, solution.block(1), schur_rhs,
            preconditioner);
```

Note how we pass the so-de ned preconditioner to the solver working on the Schureht8(ur)rtem_marki.all!