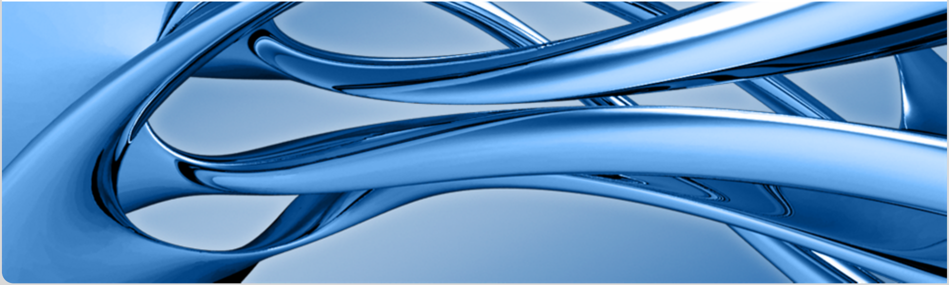
Karlsruher Institut für Technologie

Implementierung der Heterogenen Multiskalenmethode in Deal II

Karlsruhe Institute of Technology (KIT)

KARLSRUHER INSTITUT FÜR TECHNOLOGIE (KIT)



Methode

Code

Fehler-Plot

Vergleich zu Matlab

Ausblick

Heterogene Multiskalenmethode

Karlsruher Institut für Technologie

Modellproblem: (F in $L^2(\Omega)$, $0 < \eta \ll 1$)

$$\begin{cases} \text{Finde } u^\eta \in H_0^1(\Omega) \text{, sodass} \\ B^\eta(u^\eta, w) = (F, w), \forall w \in H_0^1(\Omega), \end{cases}$$

$$\text{mit } B^\eta(v, w) = \int_\Omega a^\eta(x) \nabla v(x) \cdot \nabla w(x) dx.$$

Heterogene Multiskalenmethode

Homogenisiertes Problem:

$$\begin{cases} \text{Finde } u^{\text{eff}} \in H_0^1(\Omega), \text{ sodass} \\ B^{\text{eff}}(u^{\text{eff}}, w) = (F, w), \forall w \in H_0^1(\Omega), \end{cases}$$

$$\text{mit } B^{\text{eff}}(v, w) = \int_\Omega a^{\text{eff}}(x) \nabla v(x) \cdot \nabla w(x) dx,$$

wobei

$$a^{\text{eff}}(\bar{x}) = \frac{1}{|Y_\eta(\bar{x})|} \int_{Y_\eta(\bar{x})} \left(I + D_x^T \chi\left(\bar{x}, \frac{x}{\eta}\right)\right)^T a^\eta\left(\bar{x}, \frac{x}{\eta}\right) \left(I + D_x^T \chi\left(\bar{x}, \frac{x}{\eta}\right)\right) dx.$$

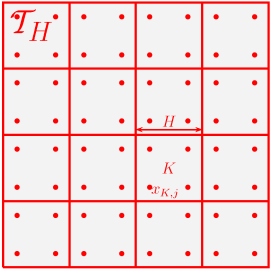
Zellprobleme:

$$\begin{cases} -\nabla_x \cdot \left(a\left(\bar{x}, \frac{x}{\eta}\right) \left(e_i + \nabla_x \chi_i\left(\bar{x}, \frac{x}{\eta}\right)\right)\right) = 0, \text{ in } Y_\eta = \left(-\frac{\eta}{2}, \frac{\eta}{2}\right)^d, \\ \chi_i \text{ Y-periodisch in der zweiten Variablen.} \end{cases}$$

Heterogene Multiskalenmethode

Diskretisierung:

$$B_H^{\text{eff}}\left(v_H, w_H\right) = \sum_{K,j} \omega_{K,j} a^{\text{eff}}\left(x_{K,j}\right) \nabla v_H \left(x_{K,j}\right) \cdot \nabla w_H \left(x_{K,j}\right) dx$$



$$B^{\text{eff}}(v, w) = \int_\Omega a^{\text{eff}}(x) \nabla v(x) \cdot \nabla w(x) dx$$

Heterogene Multiskalenmethode

Diskretisierung:

$$B_H^{\text{eff}}(v_H, w_H) = \sum_{K,j} \omega_{K,j} a^{\text{eff}}(x_{K,j}) \nabla v_H(x_{K,j}) \cdot \nabla w_H(x_{K,j}) \, dx$$

$$= \sum_{K,j} \frac{\omega_{K,j}}{|Y_\eta|} \int_{Y_\eta(x_{K,j})} a\left(x_{K,j}, \frac{x}{\eta}\right) \left[\left(I + D_x^T \chi\left(x_{K,j}, \frac{x}{\eta}\right)\right) \nabla v_H(x_{K,j})\right] \cdot \left[\left(I + D_x^T \chi\left(x_{K,j}, \frac{x}{\eta}\right)\right) \nabla w_H(x_{K,j})\right] dx$$

$$a^{\text{eff}}(x) = \frac{1}{|Y_\eta(x)|} \int_{Y_\eta(x)} \left(I + D_x^T \chi\left(x, \frac{x}{\eta}\right)\right)^T a^\eta(x) \left(I + D_x^T \chi\left(x, \frac{x}{\eta}\right)\right) dx$$

Heterogene Multiskalenmethode

Diskretisierung:

$$B_H^{\text{eff}}(v_H, w_H) = \sum_{K,j} \omega_{K,j} a^{\text{eff}}(x_{K,j}) \nabla v_H(x_{K,j}) \cdot \nabla w_H(x_{K,j})\, dx$$

$$= \sum_{K,j} \frac{\omega_{K,j}}{|Y_\eta|} \int_{Y_\eta(x_{K,j})} a\left(x_{K,j}, \frac{x}{\eta}\right) \left[\left(I + D_x^T \chi\left(x_{K,j}, \frac{x}{\eta}\right)\right) \nabla v_H(x_{K,j})\right] \cdot \left[\left(I + D_x^T \chi\left(x_{K,j}, \frac{x}{\eta}\right)\right) \nabla w_H(x_{K,j})\right] dx$$

$$\approx \sum_{K,j} \frac{\omega_{K,j}}{|Y_\eta|} \int_{Y_\eta(x_{K,j})} a\left(x_{K,j}, \frac{x}{\eta}\right) \nabla_x \tilde{v}(x_{K,j}, x) \cdot \nabla_x \tilde{w}(x_{K,j}, x)\, dx$$

Was wissen wir über $\tilde{v}$ und $\tilde{w}$?

Heterogene Multiskalenmethode

Für $\tilde{v}$ (bzw. $\tilde{w}$) gilt:

$$\nabla_x \tilde{v}\left(x_{K,j}, x\right) = \left(I + D_x^T \chi\left(x, \frac{x}{\eta}\right)\right) \nabla v_H\left(x_{K,j}\right)$$

$$\tilde{v}\left(x_{K,j}, x\right) = v_{H,\text{lin}\left(x_{K,j},x\right)}(x) + \eta \chi\left(x_{K,j}, \frac{x}{\eta}\right) \cdot \nabla v_H\left(x_{K,j}\right) + const.$$

Heterogene Multiskalenmethode

Für $\tilde{v}$ (bzw. $\tilde{w}$) gilt:

$$\nabla_x \tilde{v}\left(x_{K,j}, x\right) = \left(I + D_x^T \chi\left(x, \frac{x}{\eta}\right)\right) \nabla v_H\left(x_{K,j}\right)$$

$$\tilde{v}\left(x_{K,j}, x\right) = v_{H,\mathrm{lin}\left(x_{K,j},x\right)}(x) + \eta\chi\left(x_{K,j}, \frac{x}{\eta}\right) \cdot \nabla v_H\left(x_{K,j}\right) + const.$$

Es folgt:

$$-\nabla_x \cdot \left(a\left(x_{K,j}, \frac{x}{\eta}\right) \nabla_x \tilde{v}\left(x_{K,j}, x\right)\right) = 0$$

$$\begin{cases} -\nabla_x \cdot \left(a\left(\bar{x}, \frac{x}{\eta}\right)\left(e_i + \nabla_x \chi_i\left(\bar{x}, \frac{x}{\eta}\right)\right)\right) = 0, \text{ in } Y_\eta = \left(-\frac{\eta}{2}, \frac{\eta}{2}\right)^d, \\ \chi_i \text{ Y-periodisch in der zweiten Variablen.} \end{cases}$$

Heterogene Multiskalenmethode

Für $\tilde{v}$ (bzw. $\tilde{w}$) gilt:

$$\nabla_x \tilde{v}\left(x_{K,j}, x\right) = \left(I + D_x^T \chi\left(x, \frac{x}{\eta}\right)\right) \nabla v_H\left(x_{K,j}\right)$$

$$\tilde{v}\left(x_{K,j}, x\right) = v_{H,\mathrm{lin}\left(x_{K,j},x\right)}(x) + \eta\chi\left(x_{K,j}, \frac{x}{\eta}\right) \cdot \nabla v_H\left(x_{K,j}\right) + const.$$

Es folgt:

$$-\nabla_x \cdot \left(a\left(x_{K,j}, \frac{x}{\eta}\right) \nabla_x \tilde{v}\left(x_{K,j}, x\right)\right) = 0$$

$$\int_{Y_\eta\left(x_{K,j}\right)} a\left(x_{K,j}, \frac{x}{\eta}\right) \nabla \tilde{v} \cdot \nabla z_h dx = 0, \forall z_h \in S^q_{\mathrm{per}}\left(Y_\eta\left(x_{K,j}\right), \mathcal{T}_h\right).$$

Heterogene Multiskalenmethode

Karlsruher Institut für Technologie

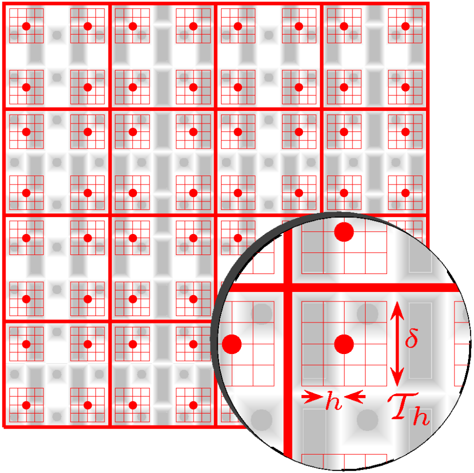
Makroproblem:

$$\begin{cases} \text{Finde } u_H \in S_0^\ell(\Omega), \text{ sodass} \\ B_H^{\text{HMM}}(u_H, w_H) = (F, w_H), \forall w_H \in S_0^\ell(\Omega), \end{cases}$$

$$\text{mit } B_H^{\text{HMM}}(v_H, w_H) = \sum_{K,j} \frac{\omega_{K,j}}{|Y_\eta|} \int_{Y_\eta(x_{K,j})} a\left(x_{K,j}, \frac{x}{\eta}\right) \nabla \tilde{v}_h(x) \cdot \nabla \tilde{w}_h(x)\, dx.$$

Mikroproblem:

$$\begin{cases} \text{Finde } \tilde{v}_h \in v_{H,\text{lin}(x_{K,j})}(x_{K,j}) + S_{\text{per}}^q(Y_\eta(x_{K,j}), \mathcal{T}_h)), \text{ sodass} \\ \int_{Y_\eta(x_{K,j})} a\left(x_{K,j}, \frac{x}{\eta}\right) \nabla \tilde{v}_h \cdot \nabla z_h dx = 0, \forall z_h \in S_{\text{per}}^q(Y_\eta(x_{K,j}), \mathcal{T}_h). \end{cases}$$


δ
h
$\mathcal{T}_h$

Mikroproblem (1)

```
template <int dim>
FullMatrix<double> Micro<dim>::solve_micro (FEValues<dim>* fe_values,
                                            unsigned int n_q_points, unsigned int dofs_per_cell) {
  for (unsigned int m=0; m<n_q_points; m++) {
    for (unsigned int n=0; n<dofs_per_cell; n++) {
      const MultiscaleTensor<dim> multiscale_tensor_mic
                 (fe_values->quadrature_point (m));
      for (; cell_mic!=endc_mic; ++cell_mic) {
        for (unsigned int q_index=0; q_index<n_q_points_mic; ++q_index)
          for (unsigned int i=0; i<dofs_per_cell_mic; ++i) {
            for (unsigned int j=0; j<dofs_per_cell_mic; ++j)
              cell_matrix_mic(i,j) += fe_values_mic.shape_grad (i, q_index) *
                                  multiscale_tensor_mic.value(
                                           fe_values_mic.quadrature_point (q_index)) *
                                  fe_values_mic.shape_grad (j, q_index) *
                                  fe_values_mic.JxW (q_index);
            cell_rhs_mic(i) -= fe_values_mic.shape_grad (i, q_index) *
                                  multiscale_tensor_mic.value(
                                            fe_values_mic.quadrature_point (q_index)) *
                                  fe_values->shape_grad(n, m) *
                                  fe_values_mic.JxW (q_index);
          }
        cell_mic->get_dof_indices (local_dof_indices_mic);
      }
```

Mikroproblem (1)

Karlsruher Institut für Technologie

```
template <int dim>
FullMatrix<double> Micro<dim>::solve_micro (FEValues<dim>* fe_values,
                    unsigned int n_q_points, unsigned int dofs_per_cell) {
  for (unsigned int m=0; m<n_q_points; m++) {
    for (unsigned int n=0; n<dofs_per_cell; n++) {
      const MultiscaleTensor<dim> multiscale_tensor_mic
                (fe_values->quadrature_point (m));
      for (; cell_mic!=endc_mic; ++cell_mic) {
        for (unsigned int q_index=0; q_index<n_q_points_mic; ++q_index)
          for (unsigned int i=0; i<dofs_per_cell_mic; ++i) {
            for (unsigned int j=0; j<dofs_per_cell_mic; ++j)
              cell_matrix_mic(i,j) += fe_values_mic.shape_grad (i, q_index) *
                          multiscale_tensor_mic.value(
                                   fe_values_mic.quadrature_point (q_index)) *
                          fe_values_mic.shape_grad (j, q_index) *
                          fe_values_mic.JxW (q_index);
            cell_rhs_mic(i) -= fe_values_mic.shape_grad (i, q_index) *
                          multiscale_tensor_mic.value(
                                    fe_values_mic.quadrature_point (q_index)) *
                          fe_values->shape_grad(n, m) *
                          fe_values_mic.JxW (q_index);
          }
        cell_mic->get_dof_indices (local_dof_indices_mic);
      }
```

Mikroproblem (2)

```
      if (Parameters::micro_periodic_bc) {
      } else {
        VectorTools::interpolate_boundary_values (dof_handler_mic, 0,
                    BoundaryValuesMic<dim>(), boundary_values_mic); }

      SolverControl           solver_control (1000, 1e-12);
      SolverCG<>              solver (solver_control);
      solver.solve (system_matrix_mic, solution_mic, system_rhs_mic,
                    PreconditionIdentity());

      if(Parameters::micro_periodic_bc)
        constraints.distribute(solution_mic);

      LinShapeFunc<dim> lin_shape_func(fe_values->quadrature_point (m),
                   fe_values->shape_value (n, m), fe_values->shape_grad (n, m));
      solution_mic += lin_shape_func_fin;

      for (unsigned int k=0; k<dof_handler_mic.n_dofs(); ++k)
        solutions_mic[k][n] =  solution_mic[k];
    }
    FullMatrix<double> tmp (dof_handler_mic.n_dofs(),dofs_per_cell);
    system_matrix_mic_wo_bc_full.mmult(tmp, solutions_mic);
    tmp *= fe_values->JxW(m)/pow(Parameters::delta,dim);
    solutions_mic.Tmmult(return_value, tmp, true);
  }
```

Mikroproblem (2)


```
      if (Parameters::micro_periodic_bc) {
      } else {
        VectorTools::interpolate_boundary_values (dof_handler_mic, 0,
                    BoundaryValuesMic<dim>(), boundary_values_mic); }

      SolverControl           solver_control (1000, 1e-12);
      SolverCG<>              solver (solver_control);
      solver.solve (system_matrix_mic, solution_mic, system_rhs_mic,
                    PreconditionIdentity());

      if(Parameters::micro_periodic_bc)
        constraints.distribute(solution_mic);

      LinShapeFunc<dim> lin_shape_func(fe_values->quadrature_point (m),
                   fe_values->shape_value (n, m), fe_values->shape_grad (n, m));
      solution_mic += lin_shape_func_fin;

      for (unsigned int k=0; k<dof_handler_mic.n_dofs(); ++k)
        solutions_mic[k][n] =  solution_mic[k];
    }
    FullMatrix<double> tmp (dof_handler_mic.n_dofs(),dofs_per_cell);
    system_matrix_mic_wo_bc_full.mmult(tmp, solutions_mic);
    tmp *= fe_values->JxW(m)/pow(Parameters::delta,dim);
    solutions_mic.Tmmult(return_value, tmp, true);
  }
```

Mikroproblem (2)

```
      if (Parameters::micro_periodic_bc) {
      } else {
        VectorTools::interpolate_boundary_values (dof_handler_mic, 0,
                    BoundaryValuesMic<dim>(), boundary_values_mic); }

      SolverControl            solver_control (1000, 1e-12);
      SolverCG<>               solver (solver_control);
      solver.solve (system_matrix_mic, solution_mic, system_rhs_mic,
                    PreconditionIdentity());

      if(Parameters::micro_periodic_bc)
        constraints.distribute(solution_mic);

      LinShapeFunc<dim> lin_shape_func(fe_values->quadrature_point (m),
                   fe_values->shape_value (n, m), fe_values->shape_grad (n, m));
      solution_mic += lin_shape_func_fin;

      for (unsigned int k=0; k<dof_handler_mic.n_dofs(); ++k)
        solutions_mic[k][n] =  solution_mic[k];
    }
    FullMatrix<double> tmp (dof_handler_mic.n_dofs(),dofs_per_cell);
    system_matrix_mic_wo_bc_full.mmult(tmp, solutions_mic);
    tmp *= fe_values->JxW(m)/pow(Parameters::delta,dim);
    solutions_mic.Tmmult(return_value, tmp, true);
  }
```

Mikroproblem (2)

```
      if (Parameters::micro_periodic_bc) {
      } else {
        VectorTools::interpolate_boundary_values (dof_handler_mic, 0,
                    BoundaryValuesMic<dim>(), boundary_values_mic); }

      SolverControl           solver_control (1000, 1e-12);
      SolverCG<>              solver (solver_control);
      solver.solve (system_matrix_mic, solution_mic, system_rhs_mic,
                    PreconditionIdentity());

      if(Parameters::micro_periodic_bc)
        constraints.distribute(solution_mic);

      LinShapeFunc<dim> lin_shape_func(fe_values->quadrature_point (m),
                   fe_values->shape_value (n, m), fe_values->shape_grad (n, m));
      solution_mic += lin_shape_func_fin;

      for (unsigned int k=0; k<dof_handler_mic.n_dofs(); ++k)
        solutions_mic[k][n] =  solution_mic[k];
    }
    FullMatrix<double> tmp (dof_handler_mic.n_dofs(),dofs_per_cell);
    system_matrix_mic_wo_bc_full.mmult(tmp, solutions_mic);
    tmp *= fe_values->JxW(m)/pow(Parameters::delta,dim);
    solutions_mic.Tmmult(return_value, tmp, true);
  }
```

Heterogene Multiskalenmethode

Makroproblem:

$$\begin{cases} \text{Finde } u_H \in S_0^\ell(\Omega), \text{ sodass} \\ B_H^{\text{HMM}}(u_H, w_H) = (F, w_H), \forall w_H \in S_0^\ell(\Omega), \end{cases}$$

$$\text{mit } B_H^{\text{HMM}}(v_H, w_H) = \sum_{K,j} \frac{\omega_{K,j}}{|Y_\eta|} \int_{Y_\eta(x_{K,j})} a\left(x_{K,j}, \frac{x}{\eta}\right) \nabla \tilde{v}_h(x) \cdot \nabla \tilde{w}_h(x)\, dx.$$

Mikroproblem:

$$\begin{cases} \text{Finde } \tilde{v}_h \in v_{H,\text{lin}(x_{K,j})}(x_{K,j}) + S_{\text{per}}^q(Y_\eta(x_{K,j}), \mathcal{T}_h)), \text{ sodass} \\ \int_{Y_\eta(x_{K,j})} a\left(x_{K,j}, \frac{x}{\eta}\right) \nabla \tilde{v}_h \cdot \nabla z_h dx = 0, \forall z_h \in S_{\text{per}}^q(Y_\eta(x_{K,j}), \mathcal{T}_h). \end{cases}$$

Makroproblem


```
template <int dim>
void Macro<dim>::assemble_system () {

  Micro<dim> microproblem;

    for (unsigned int q_index=0; q_index<n_q_points; ++q_index) {
      for (unsigned int i=0; i<dofs_per_cell; ++i) {
        cell_rhs(i) += (fe_values.shape_value (i, q_index) *
                right_hand_side.value (fe_values.quadrature_point (q_index)) *
                fe_values.JxW (q_index));
      }
    }
    cell_matrix = microproblem.solve_micro (&fe_values, n_q_points,
                                dofs_per_cell);

    cell->get_dof_indices (local_dof_indices);
    for (unsigned int i=0; i<dofs_per_cell; ++i) {
      for (unsigned int j=0; j<dofs_per_cell; ++j) {
        system_matrix.add (local_dof_indices[i], local_dof_indices[j],
                           cell_matrix(i,j));
      }
      system_rhs(local_dof_indices[i]) += cell_rhs(i);
    }
  }
```

Setup

$$\Omega = [-1, 1] \times [-1, 1]$$

$$a^\eta(x) = \begin{pmatrix} \sqrt{2} + \sin(2\pi \frac{x_1}{\eta}) & 0 \\ 0 & \sqrt{2} \end{pmatrix}$$

$$f(x) = \pi^2(1 + \sqrt{2}) \sin(\pi x_1) \sin(\pi x_2)$$

$$g_D \equiv 0 \text{ auf } \partial\Omega$$

$$\begin{cases} \text{Finde } u^\eta \in H_0^1(\Omega), \text{ sodass} \\ B^\eta(u^\eta, w) = (F, w), \forall w \in H_0^1(\Omega), \end{cases}$$

$$B^\eta(v, w) = \int_\Omega a^\eta(x) \nabla v(x) \cdot \nabla w(x) dx.$$

Setup

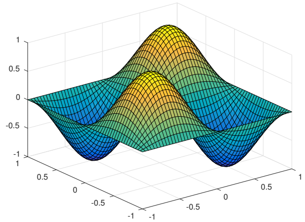
Karlsruher Institut für Technologie

$$\Omega = [-1,1] \times [-1,1]$$

$$a^\eta(x) = \begin{pmatrix} \sqrt{2} + \sin(2\pi \frac{x_1}{\eta}) & 0 \\ 0 & \sqrt{2} \end{pmatrix}$$

$$f(x) = \pi^2 (1+\sqrt{2}) \sin(\pi x_1) \sin(\pi x_2)$$

$$g_D \equiv 0 \text{ auf } \partial\Omega$$



homogenisierte Lösung

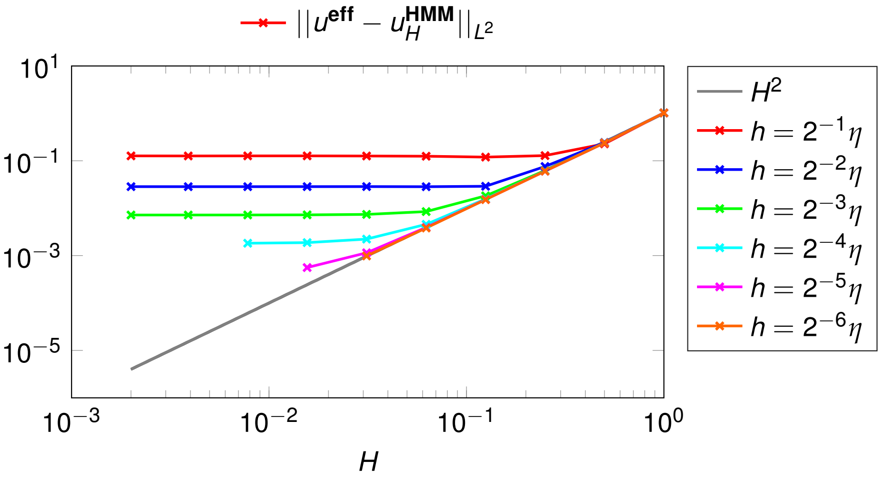
$$\begin{cases} \text{Finde } u^\eta \in H_0^1(\Omega), \text{ sodass} \\ B^\eta(u^\eta, w) = (F, w), \forall w \in H_0^1(\Omega), \end{cases}$$

$$B^\eta(v,w) = \int_\Omega a^\eta(x) \nabla v(x) \cdot \nabla w(x) dx.$$

$$\Rightarrow a^{\text{eff}}(x) = \begin{pmatrix} 1 & 0 \\ 0 & \sqrt{2} \end{pmatrix}$$

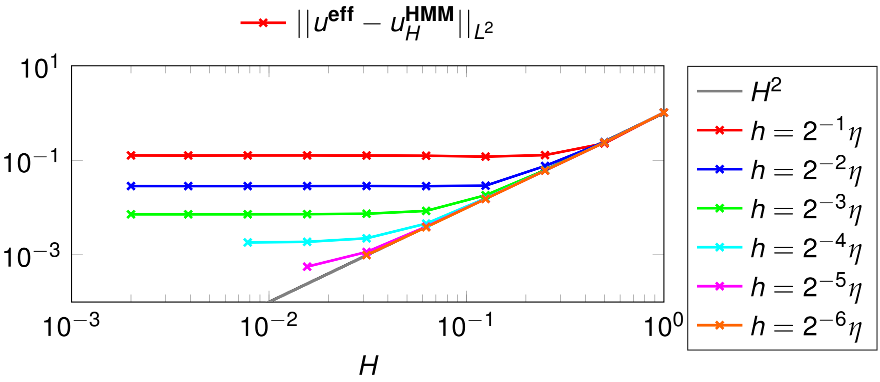
$$\Rightarrow u^{\text{eff}}(x) = \sin(\pi x_1) \sin(\pi x_2)$$

L^2-Fehler



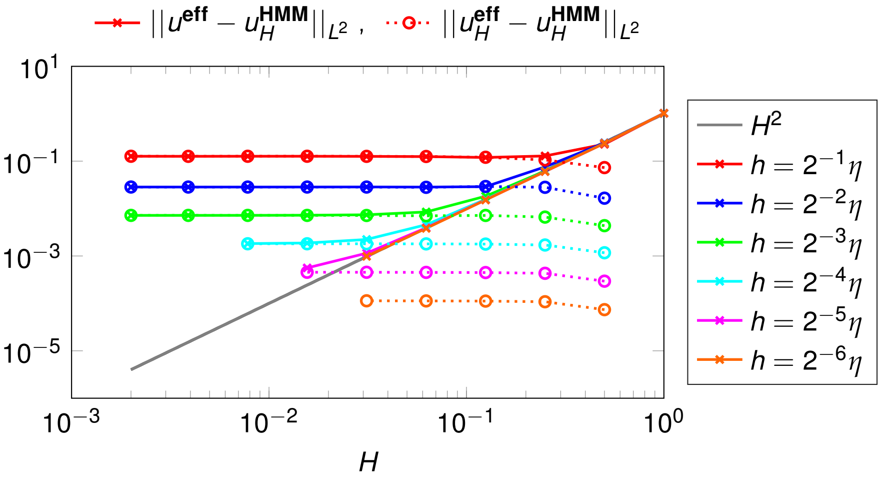
$||u^{\text{eff}} - u_H^{\text{HMM}}||_{L^2}$
10^1
10^{-1}
10^{-3}
10^{-5}
10^{-3}
10^{-2}
10^{-1}
10^0
H
H^2
$h = 2^{-1}\eta$
$h = 2^{-2}\eta$
$h = 2^{-3}\eta$
$h = 2^{-4}\eta$
$h = 2^{-5}\eta$
$h = 2^{-6}\eta$

L^2-Fehler



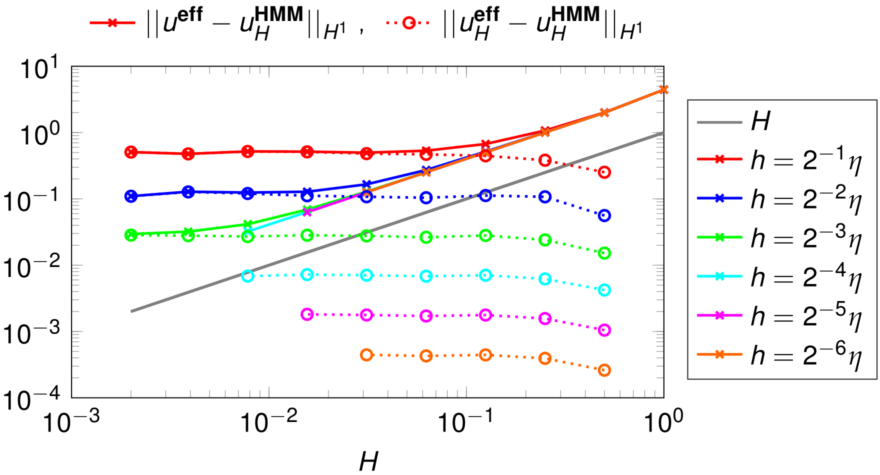
$$||u^{\text{eff}} - u_H^{\text{HMM}}|| \leq \underbrace{||u^{\text{eff}} - u_H^{\text{eff}}||}_{e_{\text{makro}}} + \underbrace{||u_H^{\text{eff}} - u_H^{\text{HMM}}||}_{e_{\text{mikro}}}$$

L^2-Fehler



$$||u^{\mathrm{eff}} - u_H^{\mathrm{HMM}}||_{L^2} \leq C\left(H^2 + \left(\frac{h}{\eta}\right)^2\right)$$

H^1-Fehler



$$||u^{\text{eff}} - u_H^{\text{HMM}}||_{H^1} \leq C \left(H + \left(\frac{h}{\eta} \right)^2 \right)$$

Übersicht

Matlab-Code von A. Abdulle et al. ermöglicht:

- elliptische und parabolische Probleme, jeweils linear und nichtlinear (2009 bzw. 2011)
- Stokes-Gleichungen mit DG-FEM (2015)
- Dreiecks-, Vierecks- und gemischte Gitter

Aber dieser Code ist langsamer!

Rechenzeit

CPU: AMD FX-8320 (8 Kerne, 3.5 GHz)
RAM: DDR3 8 GB

Auflösung		Rechenzeit	
Makro	Mikro	Matlab (2009)	Deal II
4	4	ca. 58*s*	ca. 16*s*
4	5	ca. 228*s*	ca. 64*s*
5	4	ca. 225*s*	ca. 65*s*
5	5	ca. 930*s*	ca. 270*s*

Ausblick

- Anwendung auf Wellengleichungen und Maxwellgleichungen
- paralleles Lösen der Mikroprobleme