0.0.1 General problem description

This example shows the use of the adaptive grid refinement and error estimation by the *DWR method* (For a description of the method, see [**?**].) applied to the laplace equation

$$-\Delta u = f \qquad \text{in } \Omega$$

with the analytical solution

$$u = \sin\left(\frac{\pi}{x^2+y^2}\right),$$

the corresponding right hand side $f = -\Delta u$ and appropriate Dirichlet Conditions on $\partial\Omega$, where the domain is given by

$$\Omega = [-2,2]^2 \setminus \overline{B}_{0.5}(0).$$

We want to estimate the error in the following functional of interest

$$\begin{array}{rccc} J & : & H^1(\Omega) & \longrightarrow & \mathbb{R} \\ & & u & \longmapsto & \frac{1}{|\Gamma|}\int_\Gamma u\,dx \end{array}$$

where $\Gamma = \left\{(x,y) \in \mathbb{R}^2 \,\middle|\, x = 0, -2 < y < 0.5\right\}$.

For this setting, we have the error representation

$$J(e) = \sum_{K\in\mathbb{T}_h} \left\{(R_h, z-\psi_h)_K + (r_h, z-\psi_h)_{\partial_K}\right\} \tag{1}$$

with the error $e = u - u_h$, the Triangulation $\mathbb{T}_h$, the dual solution z, arbitraty function $\phi_h \in V_h$ (the ansatz space) and the cell- and edge-residuals:

$$R_h\big|_K = f + \Delta u_h \tag{2}$$

resp.

$$r_h\big|_\Sigma = \begin{cases} \frac{1}{2}[\partial_n u_h], & \text{if } \Sigma \subset \partial_K \setminus \partial\Omega, \\ 0, & \text{if } \Sigma \subset \partial\Omega. \end{cases} \tag{3}$$

It holds $J(u) \approx 0.441956231972232$.

0.0.2 Program description

In this section we want to focus on what you have to do if you want to enhance your existing code to use the *DWR method*.

First, additionally to all the things one has to do when just solving the equation, we have to include the file

`higher_order_dwrc.h`

As we approximate the so called 'weights' $z - \phi_h$ in the error representation by a patchwise higher order interpolation of z_h (the computed dual solution), we have to enforce patch-wise refinement of the grid by giving the flag

```
Triangulation<2>::MeshSmoothing::patch_level_1
```

to the triangulation.

To be able to solve the adjoint equation for the error estimation one needs to implement some methods regarding the equation as well es the functional of interest:

- In `pdeinterface.h`
  - `CellEquation_U`: Weak form of the adjoint equation.
  - `CellMatrix_T`: The FE matrix for the adjoint problem.
  - `FaceEquation_U`: This one is needed in this case here because we have a functional of interest that lives on faces.
- `functionalinterface.h`
  - `FaceValue_U`: This is the right hand side of the adjoint equation.

During the evaluation of (1), the following methods are needed

- `StrongCellResidual`: The cell residual, see (2).
- `StrongFaceResidual`: The terms in (1) that lies in the interiour (i.e. the jumps).
- `StrongBoundaryResidual`:The terms in (1) that lies on the boundary (There are none in this case).

Note that in the above three functions we always apply the method `ResidualModifier` both to the residual as well as to the jumps on the faces. This is done to assert that we can apply both a DWR-error estimator where the residual should be multiplied with the computed weights (then this function does not do anything) as well as Residual Type error estimator for the L^2 or H^1 norm where we need to calculate element wise norms of the residual and the jumps. Then this function calculates the appropriate local terms, e.g., the square of the residual scaled with appropriate powers of the local mesh size.

After this, we tell the problem which functional we want to use for the error estimation, this is done via

```
P.SetFunctionalForErrorEstimation(LFF.GetName())
```

where `P` is of type `PDEProblemContainer` and `LFF` is the desired functional of interest.

The next thing we need is an object of the type

```
HigherOrderDWRContainer
```

This container takes care of the computation of the weights.

To build this, we need the following:

- `DOFH_higher_order`: With some higher order Finite Elemnts and the already defined triangulation, we build this `SpaceTimeHandler`. This is needed because we want to use the patch-wise higher order interpolation of the weights.
- `idc_high`: A `IntegratorDataContainer` in which we put some (face)quadrature formulas for the evaluation of the error Identity.
- A string which indicates how we want to store the weight-vectors (here: `"fullmem"`).
- `pr`: The `ParameterReader` which we have already defined.
- A enum of type `EETerms` that tells the container, which error terms we want to compute (primal error indicators vs. dual error indicators, see [**?**]).

The last preparation step is now to initialize the `DWRDataContainer` with the problem in use:

```
solver.InitializeHigherOrderDWRC(dwrc);
```

Succeding the solution of the state equation

```
solver.ComputeReducedFunctionals();
```

we compute the error indicators by calling

```
solver.ComputeRefinementIndicators(dwrc);
```

We can now get the error indicators (with signs!) out of `dwrc` by

```
dwrc.GetErrorIndicators();
```

With these indicators, we are now able to refine our grid adaptively (there are several mesh adaption strategies implemented, like 'optimized', 'fixednumber' or 'fixedfraction')

```
DOFH.RefineSpace("optimized", &error_ind);
```

Note, that one has to take the norm of each entry in the vector of the error indicators before feeding them into the `RefineSpace` method.