Numerical Methods for Partial Differential Equations WS 2012 / 13
Bonus: Multilevel diagonal scaling

The following instructions can serve as a guideline in case you would like to implement a multilevel diagonal scaling preconditioner yourself. The examples are not a regular part of the tutorial, but if you get results you are welcome to present them in the lesson (and gain some bonus credits for them)

A Write a function

```
void RefineUniform (const Mesh& coarseMesh, Mesh& fineMesh);
```

that computes the refined mesh $\mathcal{T}_{\ell+1}$=`fineMesh` from a given mesh $\mathcal{T}_\ell$=`coarseMesh` as shown above.

B (a) Write a function

```
void Restrict (const Vector& fineRes, Vector& coarseRes);
```

that computes the coarse residual `coarseRes`=$\underline{r}_\ell = I^{\ell}_{\ell+1}\, \underline{r}_{\ell+1}$ from the fine residual `fineRes`=$\underline{r}_{\ell+1}$.
Hint: Use the entries of $I^{\ell}_{\ell+1}$ from above, but set $r_{\ell,0} = 0$ (due to the incorporated Dirichlet condition).

(b) Write a function

```
void Prolongate (const Vector& coarseVec, Vector& fineVec);
```

that computes `fineVec`=$\underline{v}_{\ell+1} = I^{\ell+1}_{\ell} \underline{v}_\ell$ from `coarseVec`=$\underline{v}_\ell$.
Hint: Use the entries of $I^{\ell+1}_{\ell}$ from above, but set $v_{\ell,0} = 0$ (due to the incorporated Dirichlet condition).

Hint: Don't build/store the matrices $I^{\ell+1}_{\ell}$, $I^{\ell}_{\ell+1}$ but implement their multiplication to a vector.

C Consider `mds.hh` from the website and implement the class routines of MDSPreconditioner. Some comments/hints:

The field `jacobi_` stores the diagonals of the stiffness matrix at different levels. The routine InitDiagonal fills an element of `jacobi_` with diagonal entries of the given matrix.

The recursive routine `ApplyCL` should do the following:

```
apply the Jacobi preconditioner at the current level to get a correction w from r
   (solve the diagonal equation system)
if level > 0
   restrict r to a coarse residual rc
   call ApplyCL(level-1, rc, wc) (recursively) to get a coarse correction wc
   prolongate wc to a fine correction wf
   add wf to w
```

If you want, you can use a `vector<JacobiPreconditioner>` for `jacobi_` and reuse your Jacobi class from the Tutorial 9. However you might have to adapt it such that it has a default constructor (with no arguments) and an Initialize function which can be called in InitDiagonal.

D Solve a boundary value problem of your choice with the MDS-preconditioned PCG method, reusing your PCG code from Tutorial 9. Start with a simple mesh of e.g. two elements and perform uniform refinement. The core part of your main program could be as follows:

```
create mesh with two elements
create K and f from mesh (with BC!)
call mds.InitDiagonal (0, K)
for m=$1, \ldots, L-1$
    call mesh.RefineUniform()
    create K and f from mesh (with BC!)
    call mds.InitDiagonal (m, K)
end for
call PCG
```

Report the number of PCG iterations for L levels, where $L = 0, 1, \ldots, 10$, and compare with results of other methods.