---

29 Let Ω be a bounded domain in $\mathbb{R}^2$ with sufficiently smooth boundary Γ and the outward unit normal vector n. Let Γ_D, Γ_N, $\Gamma_R \subset \Gamma$ be disjoint such that $\overline{\Gamma}_D \cup \overline{\Gamma}_N \cup \overline{\Gamma}_R = \Gamma$. Derive the variational formulation for the following boundary value problem: Find $u : \Omega \to \mathbb{R}$ such that

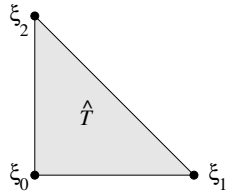
$$\begin{aligned} -\Delta u &= f && \text{in } \Omega, \\ u &= g_D && \text{on } \Gamma_D\,, \\ \frac{\partial u}{\partial n} &= g_N && \text{on } \Gamma_N\,, \\ \frac{\partial u}{\partial n} &= \alpha\left(g_R - u\right) && \text{on } \Gamma_R\,, \end{aligned}$$

for given f, g_D, g_N, g_R, and α. In particular specify V, V_g and V_0.

30 Let $\widehat{T} := \{(x, y) \in \mathbb{R}^2 : x \geq 0,\ y \geq 0,\ x + y \leq 1\}$ denote the two-dimensional reference element with the corner points $\xi_0 = (0, 0)$, $\xi_1 = (1, 0)$, and $\xi_2 = (0, 1)$. Let $\widehat{\varphi}_0$, $\widehat{\varphi}_1$, and $\widehat{\varphi}_2$ denote affine linear functions on $\widehat{T}$ that fulfill

$$\widehat{\varphi}_i(\xi_j) = \delta_{ij} \qquad \forall i, j \in \{0, 1, 2\}.$$

Derive an explicit formula for $\widehat{\varphi}_0$, $\widehat{\varphi}_1$, and $\widehat{\varphi}_2$ in terms of $\xi = (\xi^{(1)}, \xi^{(2)})$.



Programming

Even though we have an optimal solver (Gauss/Thomas) for our one-dimensional model problem, we will now turn to iterative solvers for the same problem. Keep in mind that all the concepts we use in the following can be generalized to higher dimensions; we just stay in 1D to make the programming simpler. With a working Gauss/Thomas solver, you can check if your iterative solver has converged to the correct solution.

31 Write a function `Mult(↓matrix, ↓vector, ↑result)` that computes the matrix-vector product: For a given tridiagonal matrix `matrix`$=K_h$ and a vector `vector`$=\underline{v}_h$, the function should produce `result`$=K_h\,\underline{v}_h$.

For advanced programmers: If you use your own vector-class, you may try out C^{++}'s operator overloading to allow statements like `x = A * y;`

```
inline Vector operator* (const Matrix& mat, const Vector& vec)
{
  Vector res(vec.size());
  Mult (mat, vec, res);
  return res;
}
```

32 Define a C^{++} class `Preconditioner` that implements the Jacobi preconditioner $C_h = D_h = \mathrm{diag}\,(K_h)$. Write a function (or a member function of the class `Preconditioner` that solves the linear system

$$C_h\,\underline{w}_h \;=\; \underline{r}_h\,,$$

for a given vector $\underline{r}_h$.

33 Write a function `Richardson(`$\downarrow$`A, `$\updownarrow$`x, `$\downarrow$`b, `$\downarrow$`C, `$\updownarrow$`max_iter, `$\updownarrow$`tol)` to solve the linear system

$$A\,\underline{x} \;=\; \underline{b}$$

by the preconditioned Richardson method

$$\begin{aligned} &\underline{x}^{(0)} \text{ given} \\ &\underline{x}^{(k+1)} \;=\; \underline{x}^{(k)} + C^{-1}(\underline{b} - A\,\underline{x}^{(k)}) \end{aligned}$$

with the stopping criterion

$$\| \underbrace{\underline{b} - A\,\underline{x}^{(k)}}_{=\underline{r}^{(k)}} \|_{\ell_2} \;\le\; \varepsilon\, \|\underline{b}\|_{\ell_2}\,.$$

Input:
`A`=A
`x`=$\underline{x}^{(0)}$
`b`=$\underline{b}$
`C`=C
`max_iter` ... given maximal number of iterations
`tol`=ε ... given relative accuracy

Output:
`x`=$\underline{x}^{(n)}$ ... approximate solution
`max_iter`=n ... number of iterations that were performed
`tol` ... relative accuracy that was reached

Hint: use `Richardson.hh` (download from website or see next page) and rewrite it for your own purposes.

34 Use your program to solve the problem given in Exercise 28* (see Tutorial 5) with the preconditioned Richardson method and the Jacobi preconditioner. Try different equidistant meshes ($h = 1/10$, $h = 1/20$, $h = 1/100$, etc.) and report the number of iterations to reach the relative accuracy $\varepsilon = 10^{-6}$.

File richardson.hh

```
#ifndef __RICHARDSON_H
#define __RICHARDSON_H

// Iterative template routine -- preconditioned Richardson
//
// RICHARDSON solves the linear system Ax=b using
// the preconditioned richardson iteration.
// The returned value indicates convergence within
// max_iter iterations (return value 0)
// or no convergence within max_iter iterations (return value 1)
// Upon successful return (0), the output arguments have the
// following values:
//        x: computed solution
// mat_iter: number of iterations to satisfy the stopping criterion
//      tol: residual after the final iteration

template <class MATRIX, class VECTOR, class PRECONDITIONER, class REAL>
int
RICHARDSON (const MATRIX & A, VECTOR & x, const VECTOR & b,
    const PRECONDITIONER & M, int & max_iter, REAL & tol)
{
  REAL resid;
  VECTOR z(b.size ());
  REAL normb = norm (b);
  VECTOR r = b - A * x;

  if (normb == 0.0) normb = 1;
  resid = norm (r) / normb;

  if (resid <= tol)
    {
      tol = resid;
      max_iter = 0;
      return 0;
    }

  for (int i=1; i<max_iter; i++)
    {
      z = M.solve (r);
      x += z;
      r = b - A * x;
      resid = norm(r) / normb;

      if (resid <= tol)
        {
          tol = resid;
          max_iter = i;
          return 0;
        }
    }

  tol = resid;
  return 1;
}

#endif // __RICHARDSON_H
```