25 Write a function `ImplementRobinBC(↓i, ↓g, ↓alpha, ↕matrix, ↕vector)` to implement the Robin boundary condition

$$u'(x_i) = \alpha(x_i)\left(g_R(x_i) - u(x_i)\right)$$

for given values `g`$=g_R(x_i)$, `alpha`$=\alpha(x_i)$ at the boundary node x_i identified by the index `i`$=i$. The function `ImplementRobinBC` must update the stiffness matrix `matrix` and the load vector `vector`, previously computed by `AssembleStiffnessMatrix` and `AssembleLoadVector`, respectively, in the case of homogeneous Neumann conditions.

26 Write a function `ImplementDirichletBC(↓i, ↓g, ↕matrix, ↕vector)` to implement the Dirichlet boundary condition

$$u(x_i) = g_D(x_i)$$

for a given value `g`$=g_D(x_i)$ at the boundary node x_i identified by the index `i`$=i$. The function `ImplementDirichletBC` must update the stiffness matrix `matrix` and the load vector `vector`, previously computed by `AssembleStiffnessMatrix` and `AssembleLoadVector`, respectively, in the case of homogeneous Neumann conditions, and by `ImplementRobinBC`.

Hint: Assume that applying `AssembleStiffnessMatrix`, `AssembleLoadVector` and `ImplementRobinBC` yields the following linear system

$$\begin{pmatrix} K_{00} & K_{01} & K_{02} \\ K_{10} & K_{11} & K_{12} \\ K_{20} & K_{21} & K_{22} \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} f_0 \\ f_1 \\ f_2 \end{pmatrix}$$

and that we want to impose the Dirichlet boundary condition $u_0 = u(x_0) = g_D(x_0) = g_0$. In this case, we can replace the first equation by $K_{00}u_0 = K_{00}g_0$ and substitute u_0 by g_0 in the remaining equations. The modified system reads

$$\begin{pmatrix} K_{00} & 0 & 0 \\ 0 & K_{11} & K_{12} \\ 0 & K_{21} & K_{22} \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} K_{00}g_0 \\ f_1 - K_{10}g_0 \\ f_2 - K_{20}g_0 \end{pmatrix}.$$

27 Write a function `Mult(↓matrix, ↓vector, ↑result)` which computes the matrix-vector product `result`$=K\,u$ of a given tridiagonal matrix K (`matrix`), implemented by the data type `Matrix` (see Exercise 16 in Tutorial 3), and of a given vector `vector`$=u$.

If you like, you can then use C++'s operator overloading to allow `x = A * y;`

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

28 Define a C^{++} class `Preconditioner` which implements the Jacobi Preconditioner $C_h = D_h = \operatorname{diag}(K_h)$. Write a function (or a member function of the class `Preconditioner` which solves the linear system

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

for $C_h = D_h$ (diagonal) and for a given vector $\underline{r}_h$.

29 Write a function `Richardson(↓A, ↕x, ↓b, ↓C, ↕max_iter, ↕tol)` to solve the linear system

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

by the preconditioned Richardson method:

$$\underline{x}^{(n+1)} = \underline{x}^{(n)} + C^{-1}(\underline{b} - A \, \underline{x}^{(n)})$$

with the stopping criterion

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

where `A`$=A$, `x`$=\underline{x}^{(0)}$ in input and `x`$=\underline{x}^{(n)}$ in output, `b`$=\underline{b}$, `C`$=C$ and `tol`$=\varepsilon$. In input, `maxiter` is the maximal number of iterations. In output, `maxiter`$=n$ returns the number of iterations needed to satisfy the stopping criterion.

Hint: use the template `Richardson.hpp` *and rewrite it for your own purposes, or use, e. g.,* `std::valarray<double>` *(*`#include <valarray>`*) as a vector class.*

30 Use your program to discretize the following boundary value problem:

Find a function $u(x)$ satisfying

$$\begin{aligned} -u''(x) &= f(x) \qquad & x \in \Omega \\ u(x) &= g_D(x) \qquad & x \in \Gamma_D \\ \frac{\partial u}{\partial n}(x) &= 0 \qquad & x \in \Gamma_N \end{aligned}$$

with the data $f(x) = 8$, $\Omega = (0, 1)$, $\Gamma_D = \{0\}$, $g_D(x) = -1$, $\Gamma_N = \{1\}$. Then solve the discretized problem

$$K_h \underline{u}_h = \underline{f}_h$$

by the preconditioned Richardson method with the Jacobi preconditioner $C_h = D_h = \operatorname{diag}(K_h)$.

```
// file richardson.hpp

#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
```