T U T O R I A L

"Numerical Methods for Solving Partial Differential Equations"

to the Lectures on NuPDE

T IV Monday, 13 November 2006 (Time: 13:45 - 15:15, Room: P 004)

1.4 FEM for BVPs of Second-order ODEs

We consider the same setting as in (1.18) of Tutorial III.

In the following, we denote the input parameters of the functions by ↓, the output parameters by ↑, and input/output parameters by ↕.

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

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

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` will update the input `matrix`=K_h and the input `vector`=$\underline{f}_h$, previously computed by `StiffnessMatrix` and `LoadVector`, respectively, in the case of homogeneous Neumann conditions.

20 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` will update the input `matrix`=K_h and the input `vector`=$\underline{f}_h$, previously computed by `StiffnessMatrix` and `LoadVector`, respectively, in the case of homogeneous Neumann conditions, and by `ImplementRobinBC`.

Hint: assume that the following linear system is obtained after applying `StiffnessMatrix`, `LoadVector` and `ImplementRobinBC`:

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

and that we want to impose $u_0 = u(x_0) = g_D(x_0) = g_0$. Then, we can replace the first equation by $K_{00}u_0 = K_{00}g_0$ and substitute u_0 by g_0 in the other equations. This would result in the modified linear system:

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

21 Write a function `Mult(↓matrix,↓vector,↑product)` which computes the matrix-vector product Ku (`product`) of a given tridiagonal matrix K (`matrix`), implemented by the data type `Matrix` (see Exercise 16 in Tutorial III), and of a given vector u (`vector`).

(In C++ this can be done by writing an appropriate member function for the class `Matrix`, or by overloading the operator `*`.)

22 Using `class` in C++, define a data type `Preconditioner` which implements the Jacobi preconditioner $C_h = D_h = \mathrm{diag}(K_h)$.

23 Write a function (or a member function of the class `Preconditioner`) to solve the linear system

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

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

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

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

by the preconditioned Richardson method:

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

with stopping criterion

$$\|\underline{r}^{(n)}\|_{\ell_2} \leq \epsilon \|\underline{b}\|_{\ell_2} \ ,$$

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

Hint: use the template `Richardson.h`.

Richardson.h

```
// Iterative template routine -- Preconditioned Richardson
//
// RICHARDSON solves the linear system Ax=b using
//  preconditioned Richardson iterations
// The returned value indicates convergence within max_iter iterations (0)
//  or non convergence within max_iter iterations (1)
// Upon successful return, the output arguments have the following values:
//        x: computed solution
// max_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;

  Real normb = norm(b);
  Vector r = b - A*x;

  if (normb == 0.0)
    normb = 1;

  if ((resid = norm(r) / normb) <= 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;

    if ((resid = norm(r) / normb) <= tol)
     {
      tol = resid;
      max_iter = i;
      return 0;
     }
   }

  tol = resid;
  return 1;
}
```