T U T O R I A L

"Numerical Methods for Solving Partial Differential Equations"

to the Lectures on NuPDE

T VII Monday, 10 December 2007 (Time: 08:30 – 10:00, Room: T 212)

1.9 Conjugate Gradient Method

Let K be a symmetric and positive definite matrix, and let $u^{(n)}$ be an approximation of the exact solution u of the linear system

$$K\,u = f\,, \tag{1.25}$$

obtained by the iterative method

$$u^{(n+1)} = u^{(n)} + \alpha^{(n)}\,p^{(n)} \tag{1.26}$$

where $p^{(n)}$ is a search direction and $\alpha^{(n)} \in \mathbb{R}$.

38 Determine the parameter $\alpha^{(n)}$ such that (1.26) satisfies the condition

$$\|f - K\,u^{(n+1)}\|_{\ell_2} = \min_{v \in n^{(n)} + \mathrm{span}(p^{(n)})} \|f - K\,v\|_{\ell_2}\,.$$

39 Given $p^{(n-1)}$ and the residual $r^{(n)} = f - K\,u^{(n)}$, determine the parameter $\beta^{(n-1)} \in \mathbb{R}$, such that the search direction $p^{(n)}$ given by

$$p^{(n)} = r^{(n)} + \beta^{(n-1)} p^{(n-1)}$$

satisfies the condition

$$(K\,p^{(n)},\, K\,p^{(n-1)})_{\ell_2} = 0\,.$$

40 We consider the so-called *conjugate residual (CR) method*. Given $u^{(n)}$, compute the residual $r^{(0)} = f - K\,u^{(0)}$. Then, for $n \geq 0$,

$$\begin{aligned} p^{(n)} &= \begin{cases} r^{(0)} & \text{for } n = 0\,, \\ r^{(n)} + \beta^{(n-1)} p^{(n-1)} & \text{for } n \geq 1\,, \end{cases} \\ u^{(n+1)} &= u^{(n)} + \alpha^{(n)} p^{(n)}\,, \\ r^{(n+1)} &= r^{(n)} - \alpha^{(n)} K\,p^{(n)}\,. \end{aligned}$$

The parameters $\beta^{(n-1)}$ and $\alpha^{(n)}$ are chosen such that

$$(K\,p^{(n)},\, K\,p^{(n-1)})_{\ell_2} = 0$$

and

$$\|f - K\,u^{(n+1)}\|_{\ell_2} = \min_{v \in n^{(n)} + \mathrm{span}(p^{(n)})} \|f - K\,v\|_{\ell_2}\,.$$

Prove that

$$(K\,r^{(n+1)},\, p^{(n)})_{\ell_2} = 0 \qquad \text{and} \qquad (K\,r^{(n+1)}\,, r^{(n)})_{\ell_2} = 0\,.$$

41 Write down the conjugate gradient (CG) method to solve equation (1.25) but use the scalar product

$$(w,\, v)_K := (K\, w,\, v)_{\ell_2}$$

instead of the Euclidean scalar product $(w,\, v)_{\ell_2}$. Show that this special CG method coincides with the CR method.

42 Write a function `CG(↓A, ↕x, ↓b, ↓M, ↕max_iter, ↕tol)` to solve the linear system

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

by the preconditioned CG method with a preconditioner M and 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, `M`=M, `x`=$\underline{x}^{(0)}$ in input and `x`=$\underline{x}^{(n)}$ in output, `b`=$\underline{b}$, `C`=C and `tol`=ε. 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 `cg.hpp` *and rewrite it for your own purposes.*

43 Test your CG with the Jacobi preconditioner for the problem in Exercise 29, i. e.,

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

with the data $f(x) = 8$, $\Omega = (0,\, 1)$, $\Gamma_D = \{0\}$, $g_D(x) = -1$, $\Gamma_N = \{1\}$, or a comparable problem of your choice.

cg.hpp

```
// Iterative template routine -- CG
//
// RICHARDSON solves the symmetric positive definite linear
// system Ax=b using the preconditioned conjugate gradient methd.
// 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
CG (const MATRIX & A, VECTOR & x, const VECTOR & b,
    const PRECONDITIONER & M, int & max_iter, REAL & tol)
{
  REAL resid;
  VECTOR p(b.size ());
  VECTOR z(b.size ());
  VECTOR q(b.size ());
  REAL alpha, beta, rho, rho_1;
```

```
  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);
      rho = dot (r, z);

      if (i==1)
        {
          p = z;
        }
      else
        {
          beta = rho / rho_1;
          p = z + beta * p;
        }

      q = A * p;
      alpha = rho / dot (p, q);

      x += alpha * p;
      r -= alpha * q;

      resid = norm(r) / normb;

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

      rho_1 = rho;
    }

  tol = resid;
  return 1;
}
```