Numerical Methods for Partial Differential Equations WS 2008 / 2009
Tutorial 6 Monday, December 1, 10.15–11.45, T 212

$\boxed{31}$ Let $A \in \mathbb{R}^{n\times n}$ be a symmetric matrix. Show that for $M_\tau = I - \tau A$, $\tau \in \mathbb{R}$:

$$\|M_\tau\|_{\ell_2} \;=\; \max_{\lambda\in\sigma(A)} |1-\tau\lambda| \;=\; q(\tau)\,,$$

with $q(\tau) = \max\big(|1-\tau\lambda_{\max}(A)|,\, |1-\tau\lambda_{\min}(A)|\big)$.

$\boxed{32}$ Let $A \in \mathbb{R}^{n\times n}$ be a symmetric matrix with at least one negative and one positive eigenvalue (A is indefinite). Show that

$$\max_{\lambda\in\sigma(A)} |1-\tau\,\lambda| > 1 \qquad \forall \tau \neq 0\,.$$

$\boxed{33}$ Let $A \in \mathbb{R}^{n\times n}$ be a symmetric matrix and let $\lambda_- < 0$ and $\lambda_+ > 0$ be two eigenvalues with the corresponding eigenvectors e_- and e_+. Show that there is *no* parameter $\tau \in \mathbb{R}$, such that the Richardson method

$$x_{k+1} \;=\; x_k + \tau(b - \tau A\, x_k)$$

converges for the initial value $x_0 = x + e_- + e_+$, where $x = A^{-1}b$.

$\boxed{34}$ For the CG method it was shown (in the lecture) that

$$x_k \in x_0 + \mathcal{K}_k(A,\, r_0) \qquad \text{and} \qquad r_k \perp \mathcal{K}_k(A,\, r_0)\,,$$

where $\perp$ means orthogonal in the corresponding scalar product.

We consider now the GMRES (generalized minimal residual) method, which can be applied also to indefinite matrices. There, the iterates are constructed such that

$$x_k \in x_0 + \mathcal{K}_k(A,\, r_0) \qquad \text{and} \qquad \|b - A\,x_k\|_{\ell_2} \;=\; \min_{y\in x_0+\mathcal{K}_k(A,\,r_0)} \|b - A\,y\|_{\ell_2}\,.$$

Show that in this case,

$$x_k \in x_0 + \mathcal{K}_k(A,\, r_0) \qquad \text{and} \qquad r_k \perp A(\mathcal{K}_k(A,\, r_0)\,,$$

where $\perp$ means orthogonal in the Euclidean scalar product.
Hint: Rewrite the above minimization problem as an equivalent variational problem.

$\boxed{35}$ Write a function `CG(`$\downarrow$`A, `$\updownarrow$`x, `$\downarrow$`b, `$\downarrow$`M, `$\updownarrow$`max_iter, `$\updownarrow$`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.*

$\boxed{43}$ Test your CG with the Jacobi preconditioner for the problem 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;
}
```