MAI0119/Lecture 10.5 - Contents

Symmetric Matrices and Projection Methods

Lemma Let A be symmetric and positive definite. If $\mathcal{L}_m = \mathcal{K}_m$. Then the *projection method* is well defined.

Remark Recall that $(x,y)_A = x^T Ay$ is a *scalar product* and $\|x\|_A = (x,x)_A^{1/2}$ is a *norm*.

Question How to compute a basis for $\mathcal{K}_m(A, r^{(0)})$?

Take advantage of the fact that $(x,y)_A$ is a scalar product.

The Lanczos process

Algorithm Let A be positive definite and symmetric. Calculate a basis for the Krylov subspace $\mathcal{K}_n(A, q_1)$ by

```
$r_0 = q_1,\ \beta_0 = 1,\ q_0 = 0$
for $k = 0, 1, \ldots, n$ do
    $q_{k+1} = r_k/\beta_k$ and $\alpha_k = q_k^T A q_k$.
    $r_k = (A - \alpha_k I)q_k - \beta_{k-1}q_{k-1}$.
    $\beta_k = \|r_k\|_2$.
end
```

Remark Break-down occurs if $r_k = 0$. In that case we have an invariant subspace span$(q_1, \ldots, q_k)$.

Only need to make q_{k+1} orthogonal to q_k and q_{k-1}.

Proposition Let Q_k and T_k be the matrices obtained after k steps in the Lanczos process. Then

$$AQ_k = Q_k T_k + r_k e_k^T \quad \text{or} \quad A = QTQ^T.$$

Remark Since T is tridiagonal and obtained by a similarity transformation from A it is feasible to compute all eigenvalues of a sparse symmetric matrix.

Matlab `eigs` and `svds`.

Lemma The approximate solution $x^{(m)}$ a linear system $Ax = b$ is obtained by starting the Lanczos procedure with

$$q_0 = r^{(0)} = b - Ax^{(0)}, \quad \text{and,} \quad \beta_0 = \|r^{(0)}\|_2,$$

and then setting,

$$x^{(m)} = x^{(0)} + Q_m y_m, \qquad y_m = T_m^{-1}(\beta_0 e_1).$$

Remark Can compute solutions $x^{(k)}$ during the Lanczos steps.

Definition Vectors x and y are *conjugate* if $(x, y)_A = 0$.

Proposition Let A be symmetric and positive definite. The sequence $\{x^{(k)}\}$ calculated by the Lanczos process satisfies

$$x^{(k+1)} = x^{(k)} + \alpha_k p_k, \qquad r^{(k)} = \beta_k q_{k+1},$$

and the search directions $\{p_k\}$ form a conjugate set.

Corollary The residuals $\{r^{(k)}\}$ are orthogonal to each other.

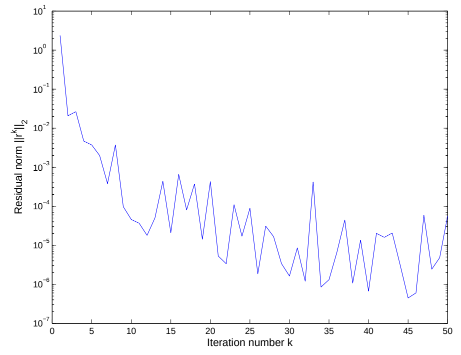
The Conjugate Gradient Method

Algorithm Compute an approximate solution $x^{(j)}$ by

```
$r^{(0)} = b - Ax^{(0)}$, $p_0 := r^{(0)}$.
for $j = 1, 2, \ldots$ do
    $\alpha_j := (r^{(j)}, r^{(j)})/(Ap_j, p_j)$.
    $x^{(j+1)} := x^{(j)} + \alpha_j p_j$.
    $r^{(j+1)} := r^{(j)} - \alpha_j A p_j$.
    $\beta_j := (r^{(j+1)}, r^{(j+1)})/(r^{(j)}, r^{(j)})$.
    $p_{j+1} := r_{j+1} + \beta_j p_j$.
end
```

Remark Need to store 4 vectors (x, p, Ap, and r). The Matlab function `pcg` implements this.

Example The *West0479* test problem



Residual norm $\|b - Ax^{(k)}\|_2$ for the first 50 CG iterations. Around 10 iterations is enough for a good solution.

The error $\|x^{(k)} - x^*\|_A$ is monotically decreasing. Not the residuals.

Convergence of the CG algorithm

Lemma Let $x^{(k)}$ be the approximate solution obtained from the kth step of CG. Then $x^{(k)}$ is of the form,

$$x^{(k)} = x^{(0)} + q_{k-1}(A)r^{(0)}$$

where q_{k-1} is a polynomial of degree $< k-1$ and

$$\|x^{(k)} - x^*\|_A = \min_{q \in \mathcal{P}_{k-1}} \|(I - Aq_{k-1}(A))(x^{(0)} - x^*)\|_A.$$

Remark Error estimate by picking the polynomial q in a clever way. The *residual polynomial* $r(x) = 1 - xq(x)$ satisfies $r(0) = 1$.

Chebyshev Polynomials

Definition The *Chebyshev* polynomioals are given by the relations, $c_0(x) = 1$, $c_1(x) = x$, and

$$c_{k+1}(x) = 2xc_k(x) - c_{k-1}(x), \qquad k \geq 1.$$

Theorem The polynomial $2^{-k+1}c_k(x)$ is the polynomial with the smallest maximum norm on the interval $[-1, 1]$ out of all polynomials with leading coefficient 1, and

$$|c_k(x)| \leq 2^{-k+1}, \qquad \text{for } -1 \leq x \leq 1.$$

Remark This follows from $c_k(x) = \cos(k \arccos(x))$, $-1 \leq x \leq 1$.

Theorem Let $[\alpha, \beta] \in \mathbb{R}$ be a non-empty interval not including 0. The minimum

$$\min_{p(0)=1} \max_{t \in [\alpha,\beta]} |p(t)|,$$

for polynomials of degree $< k$ is attained by the polynomial

$$p(t) = \frac{c_k(1 + 2\frac{t-\beta}{\beta-\alpha})}{c_k(1 + 2\frac{0-\beta}{\beta-\alpha})}.$$

Remark Since eigenvalues of symmetric A are real this version of the theorem suffice. To analyze GMRES compelx Chebyshev polynomials are needed! Otherwise very similar proofs.

Theorem Let $x^{(k)}$ be the approximate solution obtained after k steps of the CG algorithm and define

$$\eta = \frac{\lambda_{min}}{\lambda_{max} - \lambda_{min}}$$

Then

$$\|x^{(k)} - x^*\|_A \leq \frac{\|x^{(0)} - x^*\|_A}{c_k(1 + 2\eta)}$$

where $c_k(x)$ is the Chebyshev polynomial of degree k.

Corollary If $\kappa = \lambda_{max}/\lambda_{min}$ is the condition number of A then,

$$\|x^{(k)} - x^*\|_A \le 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k \|x^{(0)} - x^*\|_A.$$

Remark The number of iterations needed to reduce the error by a constant factor is proportional to $\sqrt{\kappa}$.

For general non-singular matrices we have the Arnoldi process.

Proposition Let H_m be the Hessenberg matrix and V_m be the orthogonal basis computed by the Arnoldi process. Then

$$AV_m = V_m H_m + w_m e_m^T, \quad \text{and,} \quad V_m^T A V_m = H_m.$$

After k steps we have $V_k^T A V_k \approx H_k$ and $\lambda_i(A) \approx \lambda_i(H_k)$, for $i < k$.

Remark Since k is small this is not very impressive.

Alternative Keep only the latest k vectors obtained during the Arnoldi process. Then

$$V_k^{(i)} = (v_1, \ldots, v_k), \quad \operatorname{span}(V_k^{(i)}) = \operatorname{span}(A^{i-k}v_0, \ldots, A^i v_0).$$

Compare with the power method. The subspace $V_k^{(i)}$ converges to the dominant subspace of A. We get the k largest eigenvalues of A.

Remark The k eigenvalues closest to a shift μ are obtained by running the algorithm with

$$B = (A - \mu I)^{-1}.$$

At each step the system $(A - \mu I)w_{i+1} = v_i$ is solved using GMRES. Accuracy of GMRES depends on $\kappa_2(A)$ and harder to know what errors you get. In Matlab `eigs` does this.

Preconditioning

If a system $Ax = b$ is difficult to solve using iterative methods then typically the issue is with the matrix, e.g. high condition number.

Idea Replace $Ax = b$ by an equivalent system $\hat{A}\hat{x} = \hat{b}$ that is “easier” to solve.

Questions

- How to construct the preconditioned system $\hat{A}\hat{x} = \hat{b}$.
- How to implement the iterative method efficiently.
- Preconditioners that preserve properties of the original system, e.g. symmetry.

Pick a non-singular matrix M and write

$$Ax = b \quad \Longleftrightarrow \quad M^{-1}Ax = M^{-1}b$$

The matrix M is selected so that

(1) $M \approx A$ or $M^{-1}A \approx I$.
(2) $\text{nnz}(A) \approx \text{nnz}(M)$.
(3) Easy to compute $u = M^{-1}v$.

Remark This is called *left-preconditioning*. The residuals are modified, $\hat{r} = \hat{b} - \hat{A}x = M^{-1}r$. May have to modify stopping criteria.

Right-preconditioning is $AM^{-1}y = b$, $y = Mx$. Variables change. Residuals stay the same.

Preserving Symmetry

Suppose A and M are symmetric and positive definite. We want to preserve the symmetry. Two options for preconditioning.

Split–preconditioning Let $M = LL^T$ and solve

$$\hat{A}u = (L^{-1}AL^{-T})u = L^{-1}b, \qquad x = L^{-T}u.$$

Left-predonditioning $M^{-1}A$ is *self-adjoint*, i.e. “symmetric”, with respect to the scalar product $(\cdot,\cdot)_M$. Rewrite *CG* to use this scalar product.

The Conjugate Gradient Method

The CG algorithm is

```
$r^{(0)} = b - Ax^{(0)}$, $p_0 := r^{(0)}$.
for $j = 1, 2, \ldots$ do
    $\alpha_j = (r^{(j)}, r^{(j)})/(Ap_j, p_j)$.
    $x^{(j+1)} := x^{(j)} + \alpha_j p_j$.
    $r^{(j+1)} := r^{(j)} - \alpha_j Ap_j$.
    $\beta_j := (r^{(j+1)}, r^{(j+1)})/(r^{(j)}, r^{(j)})$.
    $p_{j+1} := r_{j+1} + \beta_j p_j$.
end
```

Question How to modify to incorporate split– or left–preconditioning?

Preconditioning GMRES

- No need to worry about preserving symmetry.
- Left– and Right–preconditioning same as CG. Keep original variables and residuals.
- Split preconditioning can be implemented using a non-singular $M = LU$.

Finding a Preconditioner

There are two approaches to finding good preconditioners:

- *Problem specific:* Exploit knowledge about the specific problem that is solved, e.g. originates from a boudnary value problem for a *PDE*, and discretized using FEM or FDM.
- *General methods:* Adapt general solution methods to work better with sparse matrices, e.g. *incomplete LU/Cholesky* or *Sparse Approximate Inverse*.

Remark Most methods are of the first kind. Though not mentioned as much in books or courses.

Sparse Approximate Inverse

Approximate the inverse $A^{-1} \approx M$. Solve least squares problems,

- **Right-preconditioner:** $\|I - MA\|_F^2$.
- **Left-preconditioner:** $\|I - AM\|_F^2$.
- **Split-preconditioner:** $\|I - LAU\|_F^2$.

For the *right-preconditioner* we get an *objective function*

$$F(M) = \|I - AM\|_F^2 = \sum_{j=1}^{n} \|e_j - Am_j\|_2^2.$$

In Matlab: `pcg` and `gmres` both allow for preconditioning.