Iterative methods

vs. direct methods

Matrix splitting

Want to solve $A\vec{x} = \vec{b}$

Split A:

$$A = S - T$$

$$Ax = b \Rightarrow (S-T)\vec{x} = \vec{b}$$

$$\Rightarrow S\vec{x} = T\vec{x} + \vec{b} \quad (*)$$

Start with x_0 and solve

$$S\vec{x}_1 = T\vec{x}_0 + b$$

repeat

$$S\vec{x}_{k+1} = T\vec{x}_k + b$$

S is chose so that each of these steps is fast. (diagonal, or triangular, e.g.) Does this converge?

Subtract off the exact solution in (*)

$$S(\vec{x}_{k+1} - \vec{x}) = T(\vec{x}_k - \vec{x})$$

$$S\,\vec{e}_{k+1} = T\,\vec{e}_k$$

So $\vec{e}_{k+1} = S^{-1} T \vec{e}_k$

this relates the error at iteration $k+1$ to the error at iteration k.

So $\| \vec{e}_{k+1} \| = \| S^{-1} T \vec{e}_k \| \leq \| S^{-1} T \| \, \| \vec{e}_k \|$

If $\| S^{-1} T \| \ll 1$, this can converge quickly, but often this is not the case.

We'll highlight two common splittings:

(1) Jacobi Method

Choose $S = \text{diag}(A) = D$

$$\boxed{D \vec{x}_{k+1} = -(L+U) \vec{x}_k + \vec{b}},$$

where $A = D + L + U$; L is the strictly lower triangular part of A, & U the strictly upper triangular part.

This is trivial to solve in each iteration.

It is also trivial to parallelize, but usually slow.

(2) Gauss-Seidel Method

$$\boxed{(D+L) \vec{x}_{k+1} = -U \vec{x}_k + \vec{b}}$$

Lower triangular system. Solve by forward subst. Sequential. Generally faster convergence than Jacobi.

§5.4 Conv. Rates & Stopping Criteria.

Iterative methods (vs. direct methods)

$$\text{Cost} = \frac{\text{cost}}{\text{iter}} \cdot \#\text{iter}$$

Conv. rate

$e_k = x_k - x^*$ error at iter k.

Conv w/ rate r if

$$\lim_{k\to\infty} \frac{\|e_{k+1}\|}{\|e_k\|^r} = C \quad , \quad C > 0.$$

- $r = 1$, $C < 1$ $\Rightarrow$ linear — gain constant # of correct digits each iter $-(\log_{10}(C))$
- $r > 1$ $\Rightarrow$ superlinear
- $r = 2$ $\Rightarrow$ quadratic
- $r = 3$ $\Rightarrow$ cubic

} r X as many correct digits as in prev iter.

Ex. 5.6. Conv rates.

plot $\log e_{k+1}$ vs $\log e_k$ slope = r

$10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, \ldots$ linear $C = 10^{-1}$

$10^{-2}, 10^{-4}, 10^{-6}, 10^{-8}$ linear $C = 10^{-2}$

$10^{-2}, 10^{-3}, 10^{-5}, 10^{-8}, 10^{-12}$ superlinear, not quad.

$10^{-2}, 10^{-4}, 10^{-8}, 10^{-16}$ quadratic.

Stopping criteria

don't know e_k — ant difficult to give general

$\|x_{k+1} - x_k\| / \|x_k\|$

Power Iteration

$$\boxed{\vec{x}_{k+1} = A\vec{x}_k}$$

$\vec{x}_{k+1}$ converges to <u>eigenvector</u> corresponding to largest eigenvalue.

$$x_0 = \alpha_1 v_1 + \alpha_2 v_2 + \dots + \alpha_n v_n$$

$$A^k x_0 = \alpha_1 \lambda_1^k v_1 + \alpha_2 \lambda_2^k v_2 + \dots + \alpha_n \lambda_n^k v_n$$

$$= \lambda_1^k \left(\alpha_1 v_1 + \alpha_2 \left(\frac{\lambda_2}{\lambda_1}\right)^k v_2 + \dots + \alpha_n \left(\frac{\lambda_n}{\lambda_1}\right)^k v_n \right)$$

$j > 1$, $\left(\frac{\lambda_j}{\lambda_1}\right)^k \to 0$ as $k \to \infty$

<u>Algorithm</u> : normalized power iteration

x_0
for $k = 1, 2, \dots$
$y_k = A x_{k-1}$
$x_k = y_k / \|y_k\|$
end

note: convergence depends on ratio $\left|\frac{\lambda_2}{\lambda_1}\right|$

- possible to choose a shift σ s.t.

$$\left| \frac{\lambda_2' - \sigma}{\lambda_1' - \sigma} \right| < \left| \frac{\lambda_2}{\lambda_1} \right|$$

speed up convergence

- but still we can only find one of two extreme eigenvalues

Inverse Iteration

$$A^{-1} v = \frac{1}{\lambda} v$$

- power iteration on A^{-1} would find largest eigenvalue of A^{-1} (smallest eigenvalue of A)

algorithm:

x_0

for $k = 1, 2, \ldots$

$$A y_k = x_{k-1}$$

$$x_k = y_k / \|y_k\|$$

end

- each iteration requires a solve, do by e.g., LU or Cholesky

- inverse iteration can find any λ of A with (appropriate choice of shift)

  $A - \sigma I$ smallest eigenvalue is one closest to σ.

- especially useful if estimate to λ is available

4.5.2 Inverse Iteration

$$A^{-1}v = \frac{1}{\lambda}v$$

- find smallest (in mag.) eigenvalue of A
  largest (") " A^{-1}

Algorithm: inverse iteration

x_0 = arbitrary nonzero vec.
for $k = 1, 2, \ldots$
  Solve $Ay_k = x_{k-1}$ [next k^{th} vector]
  $x_k = y_k / \|y_k\|_\infty$ [normalize]

- can solve by, e.g., LU or Cholesky
- with appropriate choice of shift σ, inverse iteration can compute any eigenvalue of $A - \sigma I$
  - smallest eigval. of $A - \sigma I$ is λ closest to σ
  - rapid convergence if shift is close to λ
  - particularly useful if estimate to λ is available

4.5.3 Rayleigh Quotient Iteration

- if x is an approx. eigenvec. of A, then

$$\lambda x \approx Ax$$

can be interpreted as L.S. problem, with normal equations

$$x^T x \lambda = x^T A x \Rightarrow \boxed{\lambda = \frac{x^T A x}{x^T x}}$$

- can accelerate power method — Rayleigh Quotient

Matlab 'eig'

$Av = \lambda Bv$

QZ or generalized Schur decomposition
- ignores symmetry of A or B

Cholesky - symm A + spd B

Algorithm: Rayleigh Quotient Iteration

x_0 = arbitrary non-zero vector

for $k = 1, 2, \ldots$

$\sigma_k = x_{k-1}^T A x_{k-1} / x_{k-1}^T x_{k-1}$ [compute shift]

Solve $(A - \sigma_k I) y_k = x_{k-1}$ [next vector]

$x_k = y_k / \|y_k\|_\infty$ [normalize]

end

- quadratic convergence rate (for non-defective eigenval).
  - cubic for normal matrices (including symm) $AA^T = A^TA$
- but must refactor the matrix each iteration - highest cost

4.5.4 Deflation

- Assume found $(\lambda_1, \vec{x}_1)$ s.t. $A\vec{x}_1 = \lambda_1 \vec{x}_1$
- Find H (e.g., Householder) s.t. $H\vec{x}_1 = \alpha \vec{e}_1$

Then $\frac{1}{\alpha}\vec{x}_1 = H^{-1}\vec{e}_1$

$HAH^{-1}\vec{e}_1 = HA(\frac{1}{\alpha}\vec{x}_1) = \frac{1}{\alpha} HA\vec{x}_1 = \frac{\lambda_1}{\alpha} H\vec{x}_1 = \lambda_1 \vec{e}_1$

i.e. first column of HAH^{-1} is $\lambda_1 \vec{e}_1$

4.5.5 Simultaneous Iteration

Algorithm: Simultaneous Iteration

X_0 = arbitrary $n \times p$ matrix (rank p) $\quad p < n$.

for $k = 1, 2, \ldots$

$$X_k = A X_{k-1} \qquad [\text{next matrix}]$$

end

Let $S_0 = \text{span}(X_0)$

$S = \text{span}(\{\vec{v}_1, \ldots, \vec{v}_p\})$ first p eigenvectors (largest $|\lambda|$)

- If no non-zero vec in S is $\perp S_0$, then

$$S_k = A^k S_0$$

has as basis

$$X_k = A^k X_0$$

- and if $|\lambda_p| > |\lambda_{p+1}|$

$$S_k \to S$$

→ "subspace iteration"

Problems:

- need to rescale cols. of X_k
- each column converging to mult. of v_1
  - X_k becoming increasingly ill-conditioned

Therefore ..., orthogonalize →

QR Iteration

$A_0 = A$

for $k = 0, 1, 2, \ldots$

$A_k = Q_k R_k$ QR decomposition of A_k

$A_{k+1} = R_k Q_k$

end

Note:

- $A_{k+1} = R_k Q_k = Q_k^T A_k Q_k$

  so A_{k+1} is similar to A_k (same λ's)

- stable algorithm, since it is based on orthogonal similarity transforms.

- under certain conditions, A_k converges to Schur form of A: $A = QTQ^*$

  T triangular