Computational Methods
CMSC/AMSC/MAPL 460

EigenValue decomposition
Singular Value Decomposition

Ramani Duraiswami,
Dept. of Computer Science

Hermitian Matrices

- A square matrix for which $A = A^H$ is said to be an *Hermitian* matrix.
- If A is real and Hermitian it is said to be *symmetric*, and $A = A^T$.
- Every Hermitian matrix is positive definite.
- Every eigenvalue of an Hermitian matrix is real.
- Different eigenvectors of an Hermitian matrix are orthogonal to each other, i.e., their scalar product is zero.

The Power Method

- Label the eigenvalues in order of decreasing absolute value so $|\lambda_1| > |\lambda_2| > \ldots |\lambda_n|$.
- Consider the iteration formula:

$$\mathbf{y}_{k+1} = A\mathbf{y}_k$$

where we start with some initial $\mathbf{y}_0$, so that:

$$\mathbf{y}_k = A^k\mathbf{y}_0$$

- Then $\mathbf{y}_k$ converges to the eigenvector $\mathbf{x}_1$ corresponding the eigenvalue λ_1.

Proof

- We know that $A^k = X \Lambda^k X^{-1}$, so:

$$\mathbf{y}_k = A^k \mathbf{y}_0 = X \Lambda^k X^{-1} \mathbf{y}_0$$

- Now we have:

$$\Lambda^k = \begin{pmatrix} \lambda_1^k & & & \\ & \lambda_2^k & & \\ & & \ddots & \\ & & & \lambda_n^k \end{pmatrix} = \lambda_1^k \begin{pmatrix} 1 & & & \\ & \lambda_2^k / \lambda_1^k & & \\ & & \ddots & \\ & & & \lambda_n^k / \lambda_1^k \end{pmatrix}$$

- The terms on the diagonal get smaller in absolute value as k increases, since λ_1 is the dominant eigenvalue.

Proof (continued)

- So we have

$$y_k = \lambda_1^k \begin{pmatrix} \vdots & & \vdots \\ x_1 & \cdots & x_n \\ \vdots & & \vdots \end{pmatrix} \begin{pmatrix} 1 & & \\ & 0 & \\ & & \ddots \\ & & & 0 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{pmatrix} = \lambda_1^k c_1 x_1$$

  - Since $\lambda_1^k c_1 \mathbf{x}_1$ is just a constant times $\mathbf{x}_1$ then we have the required result.

Rayleigh Quotient

- Note that once we have the eigenvector, the corresponding eigenvalue can be obtained from the *Rayleigh quotient*:

  $$\text{dot}(A\mathbf{x},\mathbf{x})/\text{dot}(\mathbf{x},\mathbf{x})$$

  where dot(**a**,**b**) is the scalar product of vectors **a** and **b** defined by:

  $$\text{dot}(\mathbf{a},\mathbf{b}) = a_1b_1+a_2b_2+\ldots+a_nb_n$$

- So for our example, $\lambda_1 = -2$.

Scaling

- The λ_1^k can cause problems as it may become very large as the iteration progresses.
- To avoid this problem we scale the iteration formula:

$$\mathbf{y}_{k+1} = A(\mathbf{y}_k / r_{k+1})$$

where r_{k+1} is the component of $A\mathbf{y}_k$ with largest absolute value.

Example with Scaling

- Let A = [2 -12; 1 -5] and $\mathbf{y}_0$=[1 1]'
- $A\mathbf{y}_0$ = [-10 -4]' so r_1=-10 and $\mathbf{y}_1$=[1.00 0.40]'.
- $A\mathbf{y}_1$ = [-2.8 -1.0]' so r_2=-2.8 and $\mathbf{y}_2$=[1.0 0.3571]'.
- $A\mathbf{y}_2$ = [-2.2857 -0.7857]' so r_3=-2.2857 and $\mathbf{y}_3$=[1.0 0.3437]'.
- $A\mathbf{y}_3$ = [-2.1250 -0.7187]' so r_4=-2.1250 and $\mathbf{y}_4$=[1.0 0.3382]'.
- $A\mathbf{y}_4$ = [-2.0588 -0.6912]' so r_5=-2.0588 and $\mathbf{y}_5$=[1.0 0.3357]'.
- $A\mathbf{y}_5$ = [-2.0286 -0.6786]' so r_6=-2.0286 and $\mathbf{y}_6$=[1.0 0.3345]'.
- r is converging to the correct eigenvector -2.

Scaling Factor

- At step k+1, the scaling factor r_{k+1} is the component with largest absolute value is $A\mathbf{y}_k$.
- When k is sufficiently large $A\mathbf{y}_k \simeq \lambda_1 \mathbf{y}_k$.
- The component with largest absolute value in $\lambda_1 \mathbf{y}_k$ is λ_1 (since $\mathbf{y}_k$ was scaled in the previous step to have largest component 1).
- Hence, $r_{k+1} \rightarrow \lambda_1$ as $k \rightarrow \infty$.

MATLAB Code

```
function [lambda,y]=powerMethod(A,y,n)
for (i=1:n)
    y = A*y;
    [c j] = max(abs(y));
    lambda = y(j);
    y = y/lambda;
end
```

Convergence

- The Power Method relies on us being able to ignore terms of the form $(\lambda_j/\lambda_1)^k$ when k is large enough.
- Thus, the convergence of the Power Method depends on $|\lambda_2|/|\lambda_1|$.
- If $|\lambda_2|/|\lambda_1|=1$ the method will not converge.
- If $|\lambda_2|/|\lambda_1|$ is close to 1 the method will converge slowly.

The QR Algorithm

- The QR algorithm for finding eigenvalues is based on the QR factorisation that represents a matrix A as:

  $$A = QR$$

  where Q is a matrix whose columns are orthonormal, and R is an upper triangular matrix.
- Note that $Q^H Q = I$ and $Q^{-1} = Q^H$.
- Q is termed a *unitary* matrix.

QR Algorithm without Shifts

$A_0 = A$

for k=1,2,…

$Q_k R_k = A_k$

$A_{k+1} = R_k Q_k$

end

Since:

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

then A_k and A_{k+1} are similar and so have the same eigenvalues.

A_{k+1} tends to an upper triangular matrix with the same eigenvalues as A. These eigenvalues lie along the main diagonal of A_{k+1}.

QR Algorithm with Shift

$A_0 = A$
for k=1,2,…
    $s = A_k(n,n)$
    $Q_k R_k = A_k - sI$
    $A_{k+1} = R_k Q_k + sI$
end

Since:

$$A_{k+1} = R_k Q_k + sI$$
$$= Q_k^{-1}(A_k - sI)Q_k + sI$$
$$= Q_k^{-1} A_k Q_k$$

so once again A_k and A_{k+1} are similar and so have the same eigenvalues.

The shift operation subtracts s from each eigenvalue of A, and speeds up convergence.

MATLAB Code for QR Algorithm

- Let A be an n× n matrix

```
n = size(A,1);
I = eye(n,n);
s = A(n,n); [Q,R] = qr(A-s*I); A = R*Q+s*I
```

- Use the up arrow key in MATLAB to iterate or put a loop round the last line.

Deflation

- The eigenvalue at A(n,n) will converge first.
- Then we set s=A(n-1,n-1) and continue the iteration until the eigenvalue at A(n-1,n-1) converges.
- Then set s=A(n-2,n-2) and continue the iteration until the eigenvalue at A(n-2,n-2) converges, and so on.
- This process is called *deflation*.

The SVD

- **Definition:** Every matrix A of dimensions $m \times n$ ($m \geq n$) can be decomposed as

$$A = U \Sigma V^*$$

- where
  - U has dimension $m \times m$ and $U^*U = I$,
  - Σ has dimension $m \times n$,
    the only nonzeros are on the main diagonal, and they are nonnegative real numbers $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n$,
- V has dimension $n \times n$ and $V^* V = I$.

Relation with the Eigenvalue Decomposition

- Let $A = U \Sigma V^*$. Then

$$A^* A = (U\Sigma V^*)^* \, U\Sigma V^*$$

$$= V\Sigma^* U^* \, U\Sigma V^* = V\Sigma^2 V^*$$

- This tells us that the singular value decomposition of A is related to the Eigenvalue decomposition of $A^* A$
- Recall eigen value decomposition $A = (X \Lambda X^*)$
  - So V which contains the right singular vectors of A has the right eigenvectors of $A^* A$
  - Σ^2 are the eigenvalues of $A^* A$
  - The **singular values** σ_i of A are the square roots of the **eigenvalues** of $A^* A$.

Relation with the Eigenvalue Decomposition (2)

- Let $A = U \Sigma V$. Then

$$A A^* = (U\Sigma V^*)\,(U\Sigma V^*)^*$$
$$= U\Sigma V^* V \Sigma^* U^* = U\Sigma^2 U^*$$

- This tells us that the singular value decomposition of A is related to the Eigenvalue decomposition of AA^*
- Recall eigen value decomposition $A = (X \Lambda X^*)$
  - So U contains the the **left singular vectors** of A, which are also the left eigenvectors of $A^* A$
  - Σ^2 are the eigenvalues of AA^* *and* the **singular values** σ_i of A are the square roots of the **eigenvalues** of AA^*

Computing the SVD

- The algorithm is a variant on algorithms for computing eigendecompositions.
  - rather complicated, so better to use a high-quality existing code rather than writing your own.
- In Matlab: [U,S,V] = svd(A)
- The cost is $O(mn^2)$ when $m \geq n$. The constant is of order 10.

Uses of the SVD

- Recall to solve least squares problems we could look at the normal equations ($A^*Ax=A^*b$)
  - So, SVD is closely related to solution of least-squares
  - Used for solving ill conditioned least-squares
- Used for creating low-rank approximations
- Both applications are related

SVD and reduced rank approximation

- $\mathbf{Ax}=\mathbf{b}$ $\mathbf{A}$ is $m\times n$, $\mathbf{x}$ is $n\times 1$ and $\mathbf{b}$ is $m\times 1$.
- $\mathbf{A}=\mathbf{USV}^t$ where $\mathbf{U}$ is m×m, $\mathbf{S}$ is m×n and $\mathbf{V}$ is $n\times n$
- $\mathbf{USV}^t\,\mathbf{x}=\mathbf{b}$. So $\mathbf{SV}^t\,\mathbf{x}=\mathbf{U}^t\mathbf{b}$
- If $\mathbf{A}$ has rank r, then r singular values are significant

$$\mathbf{V}^t\mathbf{x}= \mathrm{diag}(\sigma_1^{-1},\ldots,\sigma_r^{-1},0,\ldots,0)\mathbf{U}^t\mathbf{b}$$

$$\mathbf{x}= \mathbf{V}\,\mathrm{diag}(\sigma_1^{-1},\ldots,\sigma_r^{-1},0,\ldots,0)\mathbf{U}^t\mathbf{b}$$

$$\mathbf{x}_r = \sum_{i=1}^{r} \frac{\mathbf{u}_i^t\mathbf{b}}{\sigma_i}\mathbf{v}_i \qquad \sigma_r > \varepsilon,\quad \sigma_{r+1} \le \varepsilon$$

- We can truncate r at any value and achieve "reduced-rank" approximation to the matrix
- For ordered signular values, this gives the "best reduced rank approximation"

SVD and pseudo inverse

- Pseudoinverse $\mathbf{A}^{+}=\mathbf{V}\ \mathrm{diag}(\sigma_1^{-1},\ldots,\sigma_r^{-1},0,\ldots,0)\ \mathbf{U}^t$
  - $\mathbf{A}^{+}$ is a $n\times m$ matrix.
  - If rank $(\mathbf{A}) = n$ then $\mathbf{A}^{+}=(\mathbf{A}^t\mathbf{A})^{-1}\mathbf{A}$
  - If $\mathbf{A}$ is square $\mathbf{A}^{+}=\mathbf{A}^{-1}$

Well posed problems

- Hadamard postulated that for a problem to be "well posed"

1. Solution must exist
2. It must be unique
3. Small changes to input data should cause small changes to solution

- Many problems in science and computer vision result in "ill-posed" problems.
  - Numerically it is common to have condition 3 violated.
  - Recall from the SVD $\mathbf{x} = \sum_{i=1}^{n} \frac{\mathbf{u}_i^t \mathbf{b}}{\sigma_i} \mathbf{v}_i \qquad \sigma_r > \varepsilon, \quad \sigma_{r+1} \leq \varepsilon$

    If the σ are close to zero small changes in the "data" vector **b** cause big changes in **x**.
- Converting ill-posed problem to well-posed one is called *regularization*.

SVD and Regularization

- Pseudoinverse provides one means of regularization
- Another is to solve $(\mathbf{A}+\varepsilon\mathbf{I})\mathbf{x}=\mathbf{b}$ $\mathbf{x} = \sum_{i=1}^{n} \frac{\sigma_i}{\varepsilon + \sigma_i^2}(\mathbf{u}_i^t \mathbf{b})\mathbf{v}_i$
- Solution of the regular problem requires minimizing of $\|\mathbf{Ax-b}\|^2$
- Solving this modified problemcorresponds to minimizing

$$\|\mathbf{Ax-b}\|^2 + \varepsilon\|\mathbf{x}\|^2$$

- Philosophy – pay a "penalty" of $O(\varepsilon)$ to ensure solution does not blow up.
- In practice we may know that the data has an uncertainty of a certain magnitude … so it makes sense to optimize with this constraint.
- Ill-posed problems are also called "ill-conditioned"