July 20, 2013

# Chapter 8: Eigenvalues and Singular Values

Uri M. Ascher and Chen Greif
Department of Computer Science
The University of British Columbia
{ascher,greif}@cs.ubc.ca

Slides for the book
**A First Course in Numerical Methods** (published by SIAM, 2011)
`http://www.ec-securehost.com/SIAM/CS07.html`

# Goals of this chapter

- To find out how eigenvalues and singular values of a given matrix are computed;
- to find out how the largest (and smallest) few eigenvalues and singular values are computed;
- to see some interesting applications of eigenvalues and singular values.

# Outline

- Algorithms for a few eigenvalues
- Uses of eigenvalues and eigenvectors
- Uses of SVD
- (A taste of) algorithms for all eigenvalues and SVD

# Recall eigenvalues and singular values (Ch. 4)

- For a real, square $n \times n$ matrix $A$, an eigenvalue $\lambda$ and corresponding eigenvector $\mathbf{x} \neq \mathbf{0}$ satisfy $A\mathbf{x} = \lambda\mathbf{x}$.
- There are $n$ (possibly complex) eigenpairs $\lambda_1, \lambda_2, \ldots, \lambda_n$ and eigenvectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ s.t. $A\mathbf{x}_i = \lambda_i \mathbf{x}_i$. For $\Lambda = \mathrm{diag}(\lambda_i)$

$$AX = X\Lambda.$$

- If $A$ is non-defective then the eigenvector matrix $X$ is nonsingular:

$$X^{-1}AX = \Lambda.$$

- For a real, $m \times n$ matrix $A$, the singular value decomposition (SVD) is

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

where $U$ $m \times m$ and $V$ $n \times n$ are orthogonal matrices and $\Sigma$ is "diagonal" consisting of zeros and singular values $\sigma_1 \geq \sigma_2, \cdots \geq \sigma_r > 0$, $r \leq \min(m, n)$.
- Note: singular values are square roots of eigenvalues of $A^TA$.

# Classes of methods for eigenvalue problem

- In general, must iterate to find eigenvalues. Nonetheless, methods for finding all eigenvalues for non-large matrices resemble properties of direct solvers; in particular, they are based on matrix decompositions.
- To find a few eigenvalues there are the basic power and inverse power methods, and their generalization to orthogonal iterations. Large and sparse eigenvalue solvers are based on Lanczos and Arnoldi iterations (will not be discussed). In MATLAB: **eigs**
- Methods for finding all eigenvalues are based on orthogonal similarity transformations. In MATLAB: **eig**
- In general, algorithms for finding SVD are related to those for finding eigenvalues. In MATLAB: **svd**

# Outline

- Algorithms for a few eigenvalues
- Uses of eigenvalues and eigenvectors
- Uses of SVD
- (A taste of) algorithms for all eigenvalues and SVD

# Power method

- Given $\mathbf{v}_0$, expand using eigenpairs

$$\mathbf{v}_0 = \sum_{j=1}^{n} \beta_j \mathbf{x}_j,$$

where $A\mathbf{x}_j = \lambda_j \mathbf{x}_j, \qquad j = 1, \ldots, n.$

- Then

$$A\mathbf{v}_0 = \sum_{j=1}^{n} \beta_j A\mathbf{x}_j = \sum_{j=1}^{n} (\beta_j \lambda_j)\mathbf{x}_j.$$

- Multiplying by $A$ $k-1$ times gives

$$A^k\mathbf{v}_0 = \sum_{j=1}^{n} \beta_j \lambda_j^{k-1} A\mathbf{x}_j = \sum_{j=1}^{n} (\beta_j \lambda_j^k)\mathbf{x}_j.$$

# Power method (cont.)

Assuming:

- $A$ non-defective
- $|\lambda_1| > |\lambda_j|, \; j = 2, \ldots, n$
- $\beta_1 \neq 0$

Obtain

$$A^k \mathbf{v}_0 \;\to\; \mathbf{x}_1.$$

Given eigenvector, obtain eigenvalue from Rayleigh quotient

$$\lambda_1 = \frac{\mathbf{x}_1^T A \mathbf{x}_1}{\mathbf{x}_1^T \mathbf{x}_1}.$$

# Power method (cont.)

Assuming:

- $A$ non-defective
- $|\lambda_1| > |\lambda_j|, \; j = 2, \ldots, n$
- $\beta_1 \neq 0$

Obtain

$$A^k \mathbf{v}_0 \;\rightarrow\; \mathbf{x}_1.$$

Given eigenvector, obtain eigenvalue from Rayleigh quotient

$$\lambda_1 = \frac{\mathbf{x}_1^T A \mathbf{x}_1}{\mathbf{x}_1^T \mathbf{x}_1}.$$

# Power method (cont.)

Assuming:

- $A$ non-defective
- $|\lambda_1| > |\lambda_j|, \; j = 2, \ldots, n$
- $\beta_1 \neq 0$

Obtain

$$A^k \mathbf{v}_0 \;\rightarrow\; \mathbf{x}_1.$$

Given eigenvector, obtain eigenvalue from Rayleigh quotient

$$\lambda_1 = \frac{\mathbf{x}_1^T A \mathbf{x}_1}{\mathbf{x}_1^T \mathbf{x}_1}.$$

# Power method algorithm and properties

- Algorithm:
  For $k = 1, 2, \ldots$ until termination
  $$\tilde{\mathbf{v}} = A\mathbf{v}_{k-1}$$
  $$\mathbf{v}_k = \tilde{\mathbf{v}}/\|\tilde{\mathbf{v}}\|_2$$
  $$\lambda_1^{(k)} = \mathbf{v}_k^T A\mathbf{v}_k.$$

- Properties:
  - Simple, basic, can be slow.
  - Can be applied to large, sparse or implicit matrices.
  - Limiting assumptions.
  - Used as a building block for other, more robust algorithms.

# Power method algorithm and properties

- Algorithm:

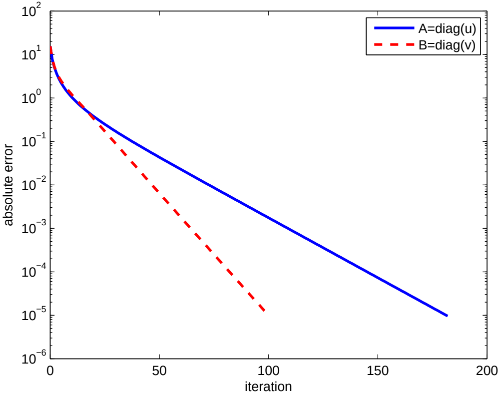
  For $k = 1, 2, \ldots$ until termination

$$\tilde{\mathbf{v}} = A\mathbf{v}_{k-1}$$
$$\mathbf{v}_k = \tilde{\mathbf{v}}/\|\tilde{\mathbf{v}}\|_2$$
$$\lambda_1^{(k)} = \mathbf{v}_k^T A \mathbf{v}_k.$$

- Properties:
  - Simple, basic, can be slow.
  - Can be applied to large, sparse or implicit matrices.
  - Limiting assumptions.
  - Used as a building block for other, more robust algorithms.

# Example: power method

```
u = [1:32]; v = [1:30, 30, 32];
A = diag(u); B = diag(v);
```



# Inverse power method

- Power method good only for a well-separated dominant eigenvalue. What about other eigenvalues? What about more general case of looking for a non-dominant eigenpair?
- If we know $\alpha$ that approximates well some simple eigenvalue $\lambda_s$ then $|\lambda_s - \alpha| \ll |\lambda_i - \alpha|, \text{ all } i \neq s$. Hence

$$|\lambda_s - \alpha|^{-1} \gg |\lambda_i - \alpha|^{-1}, \text{ all } i \neq s.$$

- These are the eigenvalues of $(A - \alpha I)^{-1}$ with the same eigenvectors!
- The parameter $\alpha$ is a shift.

# Inverse power method

- Power method good only for a well-separated dominant eigenvalue. What about other eigenvalues? What about more general case of looking for a non-dominant eigenpair?
- If we know $\alpha$ that approximates well some simple eigenvalue $\lambda_s$ then $|\lambda_s - \alpha| \ll |\lambda_i - \alpha|$, all $i \neq s$. Hence

$$|\lambda_s - \alpha|^{-1} \gg |\lambda_i - \alpha|^{-1}, \text{ all } i \neq s.$$

- These are the eigenvalues of $(A - \alpha I)^{-1}$ with the same eigenvectors!
- The parameter $\alpha$ is a shift.

# Inverse power method algorithm

- Algorithm: For $k = 1, 2, \ldots$ until termination

$$\text{Solve } (A - \alpha I)\tilde{\mathbf{v}} = \mathbf{v}_{k-1}$$
$$\mathbf{v}_k = \tilde{\mathbf{v}}/\|\tilde{\mathbf{v}}\|$$
$$\lambda^{(k)} = \mathbf{v}_k^T A \mathbf{v}_k.$$

- Properties:
  - Can apply to find different eigenvalues using different shifts.
  - But must solve (possibly many) linear systems.
  - If $\alpha$ is fixed then there is one matrix and many right hand sides: if a direct method can be applied then form LU decomposition of $A - \alpha I$ once. (See Chapter 5)
  - Alternatively, can set $\alpha = \alpha_k$ and learn it as the iteration proceeds using $\lambda_{k-1}$. Now each iteration is more expensive, but the algorithm may converge much faster.

# Inverse power method algorithm

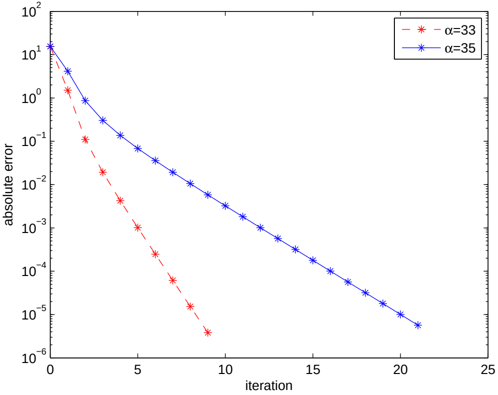
- Algorithm: For $k = 1, 2, \ldots$ until termination

$$\text{Solve } (A - \alpha I)\tilde{\mathbf{v}} = \mathbf{v}_{k-1}$$
$$\mathbf{v}_k = \tilde{\mathbf{v}}/\|\tilde{\mathbf{v}}\|$$
$$\lambda^{(k)} = \mathbf{v}_k^T A \mathbf{v}_k.$$

- Properties:
  - Can apply to find different eigenvalues using different shifts.
  - But must solve (possibly many) linear systems.
  - If $\alpha$ is fixed then there is one matrix and many right hand sides: if a direct method can be applied then form LU decomposition of $A - \alpha I$ once. (See Chapter 5)
  - Alternatively, can set $\alpha = \alpha_k$ and learn it as the iteration proceeds using $\lambda_{k-1}$. Now each iteration is more expensive, but the algorithm may converge much faster.

# Example: inverse power method

```
u = [1:32]; A = diag(u);
```



# Outline

- Algorithms for a few eigenvalues
- Uses of eigenvalues and eigenvectors
- Uses of SVD
- (A taste of) algorithms for all eigenvalues and SVD

# Uses of eigenvalues and eigenvectors

- The famous PageRank algorithm for enabling quick `www` searching amounts to finding the dominant eigenvector, corresponding to the largest eigenvalue $\lambda = 1$ in a huge matrix $A$ where relations among web pages are recorded.
- Often, the matrix $A$ represents a discretization of a differential equation (such as those in Chapters 7 and 16). Correspondingly, $A$ may be large and sparse. The eigenvector $\mathbf{x}$ in the relation $A\mathbf{x} = \lambda\mathbf{x}$ then corresponds to a discretization of an eigenfunction.
- Eigenvalues are used for determining $\ell_2$ condition numbers and naturally arise in many other analysis-related tasks.

# Outline

- Algorithms for a few eigenvalues
- Uses of eigenvalues and eigenvectors
- Uses of SVD
- (A taste of) algorithms for all eigenvalues and SVD

# Singular value decomposition (SVD)

- Recall material from Chapter 4 (where there is also a PCA example):
- For a real, $m \times n$ matrix $A$, the singular value decomposition (SVD) is given by

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

where $U$ $m \times m$ and $V$ $n \times n$ are orthogonal matrices and $\Sigma$ is "diagonal" consisting of zeros and singular values $\sigma_1 \geq \sigma_2, \cdots \geq \sigma_r > 0$, $r \leq \min(m,n)$.

- Singular values are square roots of eigenvalues of $A^T A$.
- If $r = n \leq m$ then

$$\kappa_2(A) = \frac{\sigma_1}{\sigma_n} = \sqrt{\kappa_2(A^T A)}.$$

- Note that, unlike for eigenvalues, $A$ need not be square, the singular values are real and nonnegative, and the transformation to "diagonal" form is always well-conditioned.

# Singular value decomposition (SVD)

- Recall material from Chapter 4 (where there is also a PCA example):
- For a real, $m \times n$ matrix $A$, the singular value decomposition (SVD) is given by

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

  where $U$ $m \times m$ and $V$ $n \times n$ are orthogonal matrices and $\Sigma$ is "diagonal" consisting of zeros and singular values $\sigma_1 \geq \sigma_2, \cdots \geq \sigma_r > 0$, $r \leq \min(m, n)$.
- Singular values are square roots of eigenvalues of $A^T A$.
- If $r = n \leq m$ then

$$\kappa_2(A) = \frac{\sigma_1}{\sigma_n} = \sqrt{\kappa_2(A^T A)}.$$

- Note that, unlike for eigenvalues, $A$ need not be square, the singular values are real and nonnegative, and the transformation to "diagonal" form is always well-conditioned.

# Solving the linear least squares problem using SVD

- $A = U\Sigma V^T$, so

$$\|\mathbf{b} - A\mathbf{x}\| = \|U^T\mathbf{b} - \Sigma\mathbf{y}\|, \quad \mathbf{x} = V\mathbf{y}.$$

- If $\kappa(A) = \sigma_1/\sigma_n$ is not too large then can proceed as with QR decomposition (Chapter 6).
- If $\kappa(A) = \sigma_1/\sigma_n$ is too large then proceed to determine an effective cutoff: going from $n$ backward find $r$, $1 \le r < n$, such that $\sigma_1/\sigma_r$ is not too large and set $\sigma_{r+1} = \cdots = \sigma_n = 0$, obtaining $\tilde{\Sigma}$. This changes the problem, from $A$ to $\tilde{A} = U\tilde{\Sigma}V^T$, regularizing it!
- Precisely the same procedure can be also applied for problems $A\mathbf{x} = \mathbf{b}$ where $A$ is square but too ill-conditioned (so the solution using GEPP may be meaningless).

# Solving the linear least squares problem using SVD

- $A = U\Sigma V^T$, so

$$\|\mathbf{b} - A\mathbf{x}\| = \|U^T\mathbf{b} - \Sigma\mathbf{y}\|, \quad \mathbf{x} = V\mathbf{y}.$$

- If $\kappa(A) = \sigma_1/\sigma_n$ is not too large then can proceed as with QR decomposition (Chapter 6).
- If $\kappa(A) = \sigma_1/\sigma_n$ is too large then proceed to determine an effective cutoff: going from $n$ backward find $r$, $1 \le r < n$, such that $\sigma_1/\sigma_r$ is not too large and set $\sigma_{r+1} = \cdots = \sigma_n = 0$, obtaining $\tilde{\Sigma}$. This changes the problem, from $A$ to $\tilde{A} = U\tilde{\Sigma}V^T$, regularizing it!
- Precisely the same procedure can be also applied for problems $A\mathbf{x} = \mathbf{b}$ where $A$ is square but too ill-conditioned (so the solution using GEPP may be meaningless).

# Solving the linear least squares problem using SVD

- $A = U\Sigma V^T$, so

$$\|\mathbf{b} - A\mathbf{x}\| = \|U^T\mathbf{b} - \Sigma\mathbf{y}\|, \quad \mathbf{x} = V\mathbf{y}.$$

- If $\kappa(A) = \sigma_1/\sigma_n$ is not too large then can proceed as with QR decomposition (Chapter 6).
- If $\kappa(A) = \sigma_1/\sigma_n$ is too large then proceed to determine an effective cutoff: going from $n$ backward find $r$, $1 \le r < n$, such that $\sigma_1/\sigma_r$ is not too large and set $\sigma_{r+1} = \cdots = \sigma_n = 0$, obtaining $\tilde{\Sigma}$. This changes the problem, from $A$ to $\tilde{A} = U\tilde{\Sigma}V^T$, regularizing it!
- Precisely the same procedure can be also applied for problems $A\mathbf{x} = \mathbf{b}$ where $A$ is square but too ill-conditioned (so the solution using GEPP may be meaningless).

# Regularization

Solve a nearby well-conditioned problem safely:

1. For $n, n-1\ldots$ until $r$ found s.t. $\frac{\sigma_1}{\sigma_r}$ is tolerable in size. This is the condition number of the problem that we actually solve:

$$\min\{\|\mathbf{x}\| \ ; \ \mathbf{x} = \text{argmin}\|\mathbf{b} - \tilde{A}\mathbf{x}\|\}$$

2. Let $\mathbf{u}_i$ be $i$th column vector of $U$; set $z_i = \mathbf{u}_i^T\mathbf{b}, \ i = 1, \ldots, r$.
3. Set $y_i = \sigma_i^{-1} z_i, \ i = 1, 2, \ldots, r, \quad y_i = 0, \ i = r+1, \ldots, n$.
4. Let $\mathbf{v}_i$ be $i$th column vector of $V$; set $\mathbf{x} = \sum_{i=1}^r y_i \mathbf{v}_i$.

# Regularization

Solve a nearby well-conditioned problem safely:

1. For $n, n-1\dots$ until $r$ found s.t. $\frac{\sigma_1}{\sigma_r}$ is tolerable in size. This is the condition number of the problem that we actually solve:

$$\min\{\|\mathbf{x}\| \ ; \ \mathbf{x} = \mathrm{argmin}\|\mathbf{b} - \tilde{A}\mathbf{x}\|\}$$

2. Let $\mathbf{u}_i$ be $i$th column vector of $U$; set $z_i = \mathbf{u}_i^T\mathbf{b}, \ i = 1, \dots, r$.
3. Set $y_i = \sigma_i^{-1}z_i, \ i = 1, 2, \dots, r, \quad y_i = 0, \ i = r+1, \dots, n$.
4. Let $\mathbf{v}_i$ be $i$th column vector of $V$; set $\mathbf{x} = \sum_{i=1}^r y_i\mathbf{v}_i$.

# Example

- The least squares problem $\min_{\mathbf{x}} \|C\mathbf{x} - \mathbf{b}\|$ is easily solved for

$$C = \begin{pmatrix} 1 & 0 & 1 \\ 2 & 3 & 5 \\ 5 & 3 & -2 \\ 3 & 5 & 4 \\ -1 & 6 & 3 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 4 \\ -2 \\ 5 \\ -2 \\ 1 \end{pmatrix}.$$

  Call the solution $\hat{\mathbf{x}}$.

- Next consider $\min_{\mathbf{x}} \|A\mathbf{x} - \mathbf{b}\|$, where we add to $C$ a column that is sum of the three previous ones:

$$A = \begin{pmatrix} 1 & 0 & 1 & 2 \\ 2 & 3 & 5 & 10 \\ 5 & 3 & -2 & 6 \\ 3 & 5 & 4 & 12 \\ -1 & 6 & 3 & 8 \end{pmatrix}.$$

- Note that $A$ has $m = 5,\ n = 4,\ r = 3$: can't reliably solve this using normal equations or even QR.
- But the truncated SVD method works: if $\mathbf{x}$ solves the problem with $A$, obtain $\|\mathbf{x}\| \approx \|\hat{\mathbf{x}}\|$, and $\|A\mathbf{x} - \mathbf{b}\| \approx \|C\hat{\mathbf{x}} - \mathbf{b}\|$.

# Example

- The least squares problem $\min_{\mathbf{x}} \|C\mathbf{x} - \mathbf{b}\|$ is easily solved for

$$C = \begin{pmatrix} 1 & 0 & 1 \\ 2 & 3 & 5 \\ 5 & 3 & -2 \\ 3 & 5 & 4 \\ -1 & 6 & 3 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 4 \\ -2 \\ 5 \\ -2 \\ 1 \end{pmatrix}.$$

  Call the solution $\hat{\mathbf{x}}$.

- Next consider $\min_{\mathbf{x}} \|A\mathbf{x} - \mathbf{b}\|$, where we add to $C$ a column that is sum of the three previous ones:

$$A = \begin{pmatrix} 1 & 0 & 1 & 2 \\ 2 & 3 & 5 & 10 \\ 5 & 3 & -2 & 6 \\ 3 & 5 & 4 & 12 \\ -1 & 6 & 3 & 8 \end{pmatrix}.$$

- Note that $A$ has $m = 5,\ n = 4,\ r = 3$: can't reliably solve this using normal equations or even QR.
- But the truncated SVD method works: if $\mathbf{x}$ solves the problem with $A$, obtain $\|\mathbf{x}\| \approx \|\hat{\mathbf{x}}\|$, and $\|A\mathbf{x} - \mathbf{b}\| \approx \|C\hat{\mathbf{x}} - \mathbf{b}\|$.

# Example

- The least squares problem $\min_{\mathbf{x}} \|C\mathbf{x} - \mathbf{b}\|$ is easily solved for

$$C = \begin{pmatrix} 1 & 0 & 1 \\ 2 & 3 & 5 \\ 5 & 3 & -2 \\ 3 & 5 & 4 \\ -1 & 6 & 3 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 4 \\ -2 \\ 5 \\ -2 \\ 1 \end{pmatrix}.$$

  Call the solution $\hat{\mathbf{x}}$.
- Next consider $\min_{\mathbf{x}} \|A\mathbf{x} - \mathbf{b}\|$, where we add to $C$ a column that is sum of the three previous ones:

$$A = \begin{pmatrix} 1 & 0 & 1 & 2 \\ 2 & 3 & 5 & 10 \\ 5 & 3 & -2 & 6 \\ 3 & 5 & 4 & 12 \\ -1 & 6 & 3 & 8 \end{pmatrix}.$$

- Note that $A$ has $m = 5,\ n = 4,\ r = 3$: can't reliably solve this using normal equations or even QR.
- But the truncated SVD method works: if $\mathbf{x}$ solves the problem with $A$, obtain $\|\mathbf{x}\| \approx \|\hat{\mathbf{x}}\|$, and $\|A\mathbf{x} - \mathbf{b}\| \approx \|C\hat{\mathbf{x}} - \mathbf{b}\|$.

# Truncated SVD and data compression

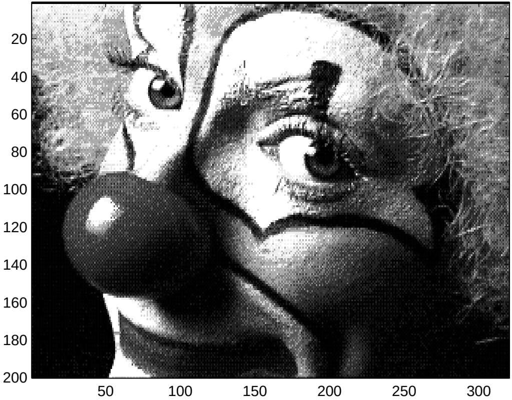
- Given an $m \times n$ matrix $A$, the best rank-$r$ approximation of $A = U\Sigma V^T$ is the matrix

$$A_r = \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^T.$$

- This is another example of truncated SVD (TSVD), so named because only the first $r$ columns of $U$ and $V$ are utilized.
- It is a model reduction technique, which is a best approximation in the sense that $\|A - A_r\|_2$ is minimal over all possible rank-$r$ matrices. The minimum residual norm is equal to $\sigma_{r+1}$.
- Note $A_r$ uses only $r(m + n + 1)$ storage locations – significantly fewer than $mn$ if $r \ll \min(m, n)$.

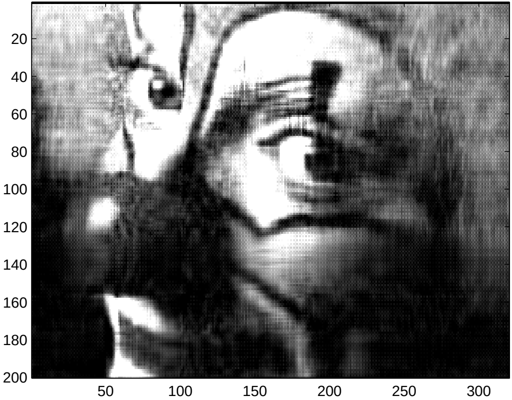
# Example

Consider the following clown image, taken from MATLAB's image repository:



# Compressed image of clown example

Here is what we get if we use $r = 20$.



# Assessment

- The original image requires $200 \times 320 = 64000$ matrix entries to be stored; the compressed image requires merely $20 \cdot (200 + 320 + 1) \approx 10000$ storage locations.
- By storing less than $16\%$ of the data, we get a reasonable image. It's not great, but all main the features of the clown are clearly shown.
- What are less clear in the compressed image are fine features (high frequency), such as the fine details of the clown's hair.
- Certainly, specific techniques such as DCT (Chapter 13) and wavelet are far superior to SVD for the task of image compression. Still, our example visually shows that most information is stored already in the leading singular vectors.

# Outline

- Algorithms for a few eigenvalues
- Uses of eigenvalues and eigenvectors
- Uses of SVD
- (A taste of) algorithms for all eigenvalues and SVD

# Finding all eigenvalues

- Consider $A$ real and square. Apply similarity orthogonal transformations

$$A_0 = A; \qquad A_{k+1} = Q_k^T A_k Q_k, \; k = 0, 1, 2, \ldots,$$

so that $A_k \rightarrow$ upper triangular (diagonal for symmetric case) form.

- QR algorithm (distinct from QR decomposition!): two stages.
- Stage I
  Given $A$, use Householder reflections (Chapter 6) to zero out elements in first diagonal: $Q^{(1)T} A = A^{(1)}$. However, to maintain similarity must multiply by $Q^{(1)}$ on the right, so

$$A_1 = Q^{(1)T} A Q^{(1)}.$$

# Finding all eigenvalues

- Consider $A$ real and square. Apply similarity orthogonal transformations

$$A_0 = A; \qquad A_{k+1} = Q_k^T A_k Q_k, \; k = 0, 1, 2, \ldots,$$

so that $A_k \;\rightarrow$ upper triangular (diagonal for symmetric case) form.

- QR algorithm (distinct from QR decomposition!): two stages.
- Stage I
  Given $A$, use Householder reflections (Chapter 6) to zero out elements in first diagonal: $Q^{(1)T} A = A^{(1)}$. However, to maintain similarity must multiply by $Q^{(1)}$ on the right, so

$$A_1 = Q^{(1)T} A Q^{(1)}.$$

# QR algorithm: stages I-II

- Define

$$A_1 = Q^{(1)T} A Q^{(1)}.$$

But can only zero elements *below main subdiagonal*.

- Leads to an upper Hessenberg form. (Tridiagonal if $A$ is symmetric.)
- Stage II

  Let $A_0$ be the given matrix, transformed into upper Hessenberg form;
  for $k = 0, 1, 2, \ldots$ until termination
  QR decomposition: $Q_k R_k = A_k - \alpha_k I$;
  Construct next iterate: $A_{k+1} = R_k Q_k + \alpha_k I$

# QR algorithm: stages I-II

- Define

$$A_1 = Q^{(1)T} A Q^{(1)}.$$

But can only zero elements *below main subdiagonal*.

- Leads to an upper Hessenberg form. (Tridiagonal if $A$ is symmetric.)
- Stage II

  Let $A_0$ be the given matrix, transformed into upper Hessenberg form;
  for $k = 0, 1, 2, \ldots$ until termination
  QR decomposition: $Q_k R_k = A_k - \alpha_k I$;
  Construct next iterate: $A_{k+1} = R_k Q_k + \alpha_k I$