New Directions in Mathematics

Adérito Araújo
DMUC, University of Coimbra, Portugal

1 2 9 0
UNIVERSIDADE D COIMBRA

Coimbra, 2024

Module Overview

Numerical Linear Algebra

New Directions in Mathematics

Adérito Araújo (alma@mat.uc.pt)
March 15, 2024

Syllabus

- What is numerical linear algebra?
  - Solving linear algebra problems using efficient algorithms on computers
- Module topics: direct and iterative methods for solving simultaneous linear equations ($Ax = b$)
  - Matrix factorization and decomposition.
  - Stationary iterative methods: Jacobi, Gauss-Seidel and relaxation methods
  - Non stationary iterative methods: Arnoldi and GMRES methods
  - The two-grid/multigrid and domain decomposition methods

Syllabus

- Direct and iterative methods
  - Direct methods: solve the problem by a finite sequence of operations and in the absence of rounding errors, would deliver an exact solution; operate directly on elements of a matrix
  - Iterative methods: solve a problem by finding successive approximations to the solution starting from an initial guess, that hopefully converge to the true solution; often are easier to implement on parallel computers
- Prerequisite/co-requisite
  - Good knowledge in linear algebra
  - Programming experience in MATLAB (Fortran, C, C++)
  - Good numerical skils
- Required Textbook: Alfio Quarteroni, Riccardo Sacco, Fausto Saleri, Numerical Mathematics, Texts in Applied Mathematics Volume 37, 2007, ISBN: 978-1-4757-7394-1 (**Chapters 3 - 4**)
- Grading: Assignments ($5 \times 20\%$)

Lecture 0

Foundations of Matrix Analysis

New Directions in Mathematics

Adérito Araújo (alma@mat.uc.pt)
March 15, 2024

Orthogonal Vectors and Matrices, Norms

Transpose and Adjoint

- For real A, the transpose of A is obtained by interchanging rows/columns

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \Rightarrow A^T = \begin{bmatrix} a_{11} & a_{21} & a_{31} \\ a_{21} & a_{22} & a_{32} \end{bmatrix}$$

- The adjoint or hermitian conjugate also takes complex conjugates

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \Rightarrow A^* = \begin{bmatrix} \bar{a}_{11} & \bar{a}_{21} & \bar{a}_{31} \\ \bar{a}_{21} & \bar{a}_{22} & \bar{a}_{32} \end{bmatrix}$$

- A is symmetric (hermitian) if $A = A^T$ ($A = A^*$)

Inner Product

- Inner product of two column vectors $x, y \in \mathbb{C}^n$

$$x^* y = \sum_{i=1}^{n} \bar{x}_i y_i$$

- Euclidean length of x

$$\|x\| = \sqrt{x^* x} = \left(\sum_{i=1}^{n} |x_i|^2 \right)$$

- Angle α between x, y

$$\cos \alpha = \frac{x^* y}{\|x\| \|y\|}$$

Positive Definite Matrices

- A hermitian matrix A is symmetric (hermitian) positive definite if $x^T Ax > 0$ ($x^* Ax > 0$) for $x \neq 0$
- Exercise 0.1: $x^* Ax$ is always real.
- Exercise 0.2: If $A \in \mathbb{C}^{m \times m}$ is PD and X has full column rank, then $X^* AX$ is PD.
- Any principal submatrix of a PD matrix A is PD, and every diagonal entry $a_{ii} > 0$
- Exercise 0.3: PD matrices have positive real eigenvalues and orthogonal eigenvectors.

In MATLAB

Quantity	**MATLAB Syntax**	**Comment**
Transpose of A	`A.'`	Transpose only
Adjoint of A	`A'`	Transpose + complex conjugate
Inner product $x^* y$	`x'*y` `dot(x,y)`	`'* assumes column vector`
Lenght $\|x\|$	`sqrt(x'*x)` `norm(x)`	`'* assumes column vector`

Orthogonal Vectors

- The vectors $x, y \in \mathbb{C}^n$ are orthogonal if

$$x^* y = 0$$

- The sets os vectors X, Y are orthogonal of

$$\text{every } x \in X \text{ is orthogonal to every } y \in Y$$

- A set of (nonzero) vectors S is orthogonal if

  vectors pairwise orthogonal, i.e., for $x, y \in S$, $x \neq y \Rightarrow x^* y = 0$

  and orthonormal if, in addition

$$\text{every } x \in S \text{ has } \|x\| = 1$$

Orthogonal and Unitary Matrices

- A square matrix $Q \in \mathbb{C}^{n \times n}$ is unitary (orthogonal in real case) if

$$Q^* = Q^{-1}$$

- For unitary Q

$$Q^* Q = I \Leftrightarrow q_i^* q_j = \delta_{ij}$$

- Interpretation of unitary-times-vector product

$$\begin{aligned} x = Q^* b \quad &= \quad \text{solution to } Qx = b \\ &= \quad \text{the vector of coefficients of the expansion of } b \\ &\quad\quad \text{in the basis of columns of } Q \end{aligned}$$

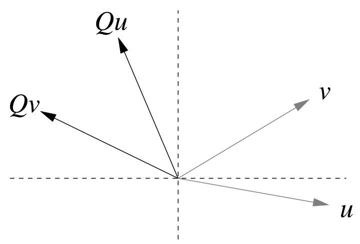
Preservation of Geometry Structure

- Inner product is preserved under multiplication by unitary Q

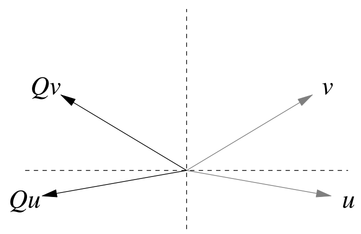
$$(Qx)^*(Qy) = x^*Q^*Qy = x^*y$$

- Therefore lengths of vectors and angles between vectors are preserved
- A real orthogonal Q is either a rigid rotation or reflection

Rotation



Reflection



Norms in MATLAB

Quantity	MATLAB Syntax
$\|x\|_1$	`sum(abs(x))` or `norm(x,1)`
$\|x\|_2$	`sqrt(x'*x)` or `norm(x)`
$\|x\|_p$	`sum(abs(x).^p).^(1/p)` or `norm(x,p)`
$\|x\|_\infty$	`max(abs(x))` or `norm(x,inf)`
$\|A\|_1$	`max(sum(abs(A),1))` or `norm(A,1)`
$\|A\|_2$	`norm(A)`
$\|A\|_\infty$	`max(sum(abs(A),2))` or `norm(A,inf)`
$\|A\|_F$	`sqrt(A(:)'*A(:))` or `norm(x,'fro')`

The Singular Value Decomposition

Diagonalizable Matrices

- A square matrix A is called diagonalizable or non-defective if it is similar to a diagonal matrix, i.e., there exists an invertible matrix P and a diagonal matrix D such that

$$P^{-1}AP = D$$

- Exercise 0.4: If $A \in \mathbb{C}^{n \times n}$ has n linear independent columns, there exists an eigenvalue decomposition (EVD)

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

- If A is real and symmetric, the EVD is always possible

$$A = U\Lambda U^T,$$

with U an unitary matrix

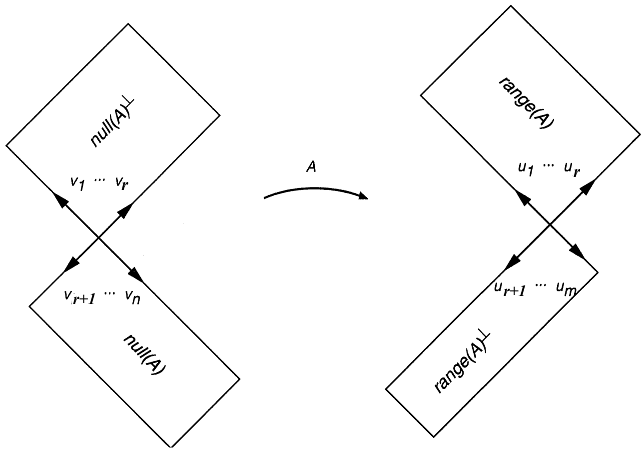
The SVD - Brief Description

- Suppose that $A \in \mathbb{C}^{m \times n}$ with $m \geqslant n$ and full rank ($r = n$)
- Choose orthonormal basis

$$v_1, \ldots, v_n \text{ for the row space}$$

$$u_1, \ldots, u_n \text{ for the column space}$$

such that Av_i is in the direction of u_i: $Av_i = \sigma_i u_i$



- The singular values $\sigma_1 \geqslant \sigma_2 \geqslant \cdots \geqslant \sigma_n > 0$

The SVD - Brief Description

- In matrix form, $Av_i = \sigma_i u_i$ becomes

$$AV = \hat{U}\hat{\Sigma} \Leftrightarrow A = \hat{U}\hat{\Sigma}V^*$$

where $\hat{\Sigma} = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_n)$

- This is the reduced singular value decomposition
- Add orthonormal extension to $\hat{U}$ and add rows to $\hat{\Sigma}$ to obtain the full sigular value decomposition

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

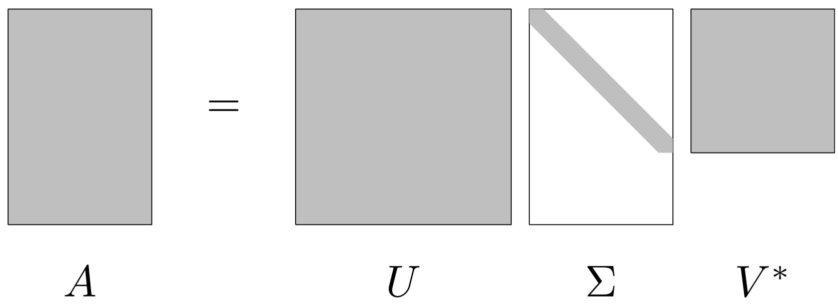
The Full Singular Value Decomposition

- Let A be an $m \times n$ matrix. The singular value decomposition of A is the factorization $A = U\Sigma V^*$ where

  U is $m \times m$ unitary (the left singular vectors of A)

  V is $n \times n$ unitary (the right singular vectors of A)

  U is $m \times m$ unitary (the left singular vectors of A)



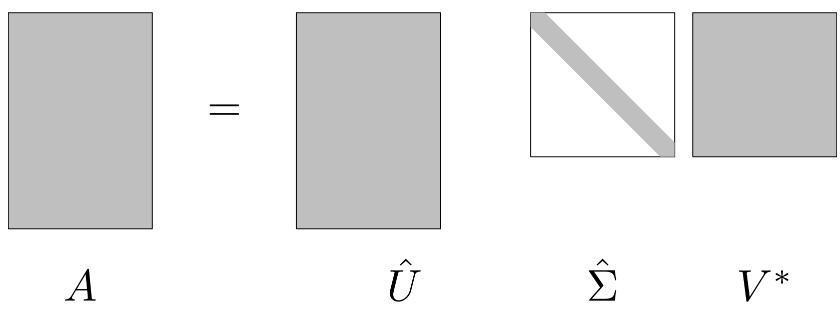
The Reduced Singular Value Decomposition

- A more compact representation is the reduced SVD, for $m \geqslant n$:

$$A = \hat{U}\hat{\Sigma}V^*$$

where

$$U \text{ is } m \times n, \qquad V \text{ is } n \times n, \qquad \Sigma \text{ is } n \times n$$



The SVD and The Eigenvalue Decomposition

- The eigenvalue decomposition $A = X\Lambda X^{-1}$
  - uses the same basis X for row and column space, but the SVD uses two different basis V and U
  - generally does not use an orthonormal basis, but the SVD does
  - is only defined for square matrices, but the SVD exists for all matrices
- For symmetric positive definite matrices A, the EVD and SVD are equal

Matrix Properties (Exercise 0.5)

1. The rank of A is r, the number of nonzero singular values
2. range $(A) = \langle u_1, \ldots, u_r \rangle$ and null $(A) = \langle v_{r+1}, \ldots, v_n \rangle$
3. $\|A\|_2 = \sigma_1$ and $\|A\|_F = \sqrt{\sigma_1^2 + \sigma_2^2 \cdots + \sigma_r^2}$
4. Nonzero eigenvalues of A^*A are nonzero σ_j^2, eigenvectors are v_j; Nonzero eigenvalues of AA^* are nonzero σ_j^2, eigenvectors are u_j
5. In $A = A^*$, $\sigma_i = |\lambda_j|$, where λ_j are eigenvalues of A
6. For square A, $|\det(A)| = \prod_{j=1}^{m} \sigma_j$

Existence and Uniqueness

Theorem 0.1: Existence

Every matrix $A \in \mathbb{C}^{m\times n}$ has a SVD.

Theorem 0.2: Uniqueness

The singular values $\{\sigma_j\}$ are uniquely determined. If A is square and the σ_j are distinct, the left and right singular vectors are uniquely determined **up to complex signs**.

Example: $A = \begin{bmatrix} 2 & 2 \\ 1 & -1 \end{bmatrix}$

- Prove that the eigenvalues of

$$A^T A = \begin{bmatrix} 5 & 3 \\ 3 & 5 \end{bmatrix},$$

are $\lambda_1 = 8$ and $\lambda_2 = 2$ and the (orthonormal) eigenvectors are

$$v_1 = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix} \quad \text{and} \quad v_2 = \begin{bmatrix} -1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$$

- Then

$$\Sigma = \begin{bmatrix} \sigma_1 & \\ & \sigma_2 \end{bmatrix} = \begin{bmatrix} 2\sqrt{2} & \\ & \sqrt{2} \end{bmatrix} \text{ and } V = \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} \\ 1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix}$$

Example: $A = \begin{bmatrix} 2 & 2 \\ 1 & -1 \end{bmatrix}$ (cont.)

- The columns of U are obtained by

$$\sigma_1 u_1 = Av_1 = \begin{bmatrix} 2 & 2 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix} = \begin{bmatrix} 2\sqrt{2} \\ 0 \end{bmatrix} \Rightarrow u_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

  and

$$\sigma_2 u_2 = Av_2 = \begin{bmatrix} 2 & 2 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} -1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix} = \begin{bmatrix} 0 \\ \sqrt{2} \end{bmatrix} \Rightarrow u_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

- The SVD of $A = U\Sigma V^T$ is

$$\begin{bmatrix} 2 & 2 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 2\sqrt{2} & \\ & \sqrt{2} \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ -1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix} \quad \square$$

- Exercise 0.6: Obtain the SVD of $A = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$.

Low-Rank Approximations

- The SVD can be written as a sum of rank-one matrices

$$A = \sum_{j=1}^{r} \sigma_j u_j v_j^*$$

- Eckart-Young (1936): The best rank η approximation in the 2-norm is

$$A_\eta = \sum_{j=1}^{\eta} \sigma_j u_j v_j^*$$

  with

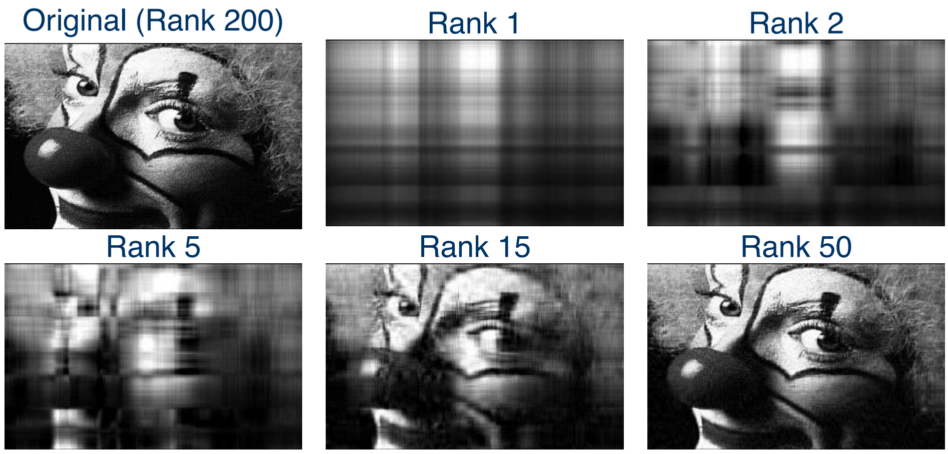
$$\|A - A_\eta\|_2 = \sigma_{\eta+1}$$

- Also true in the Frobenius norm, with

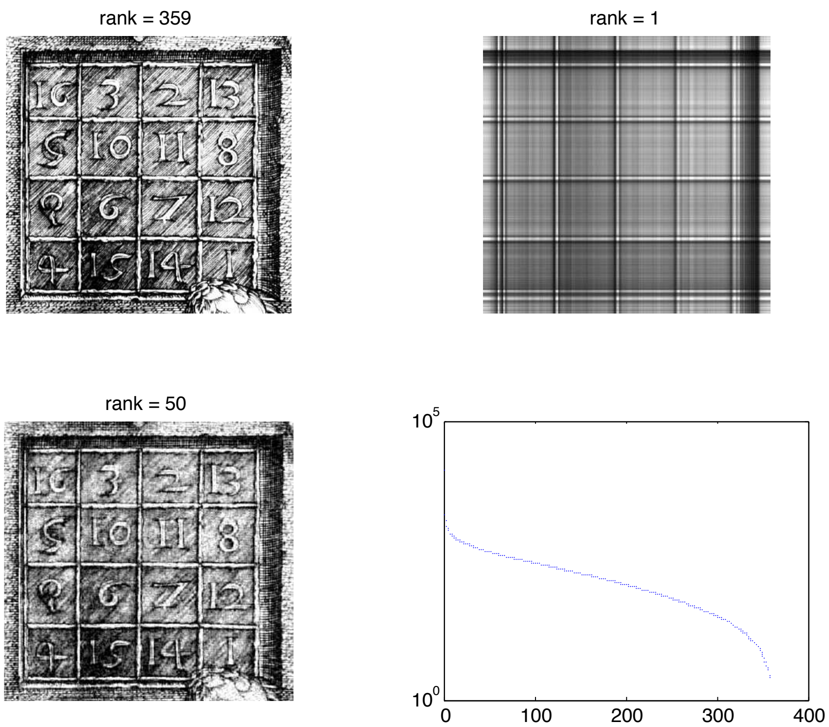
$$\|A - A_\eta\|_F = \sqrt{\sigma_{\eta+1}^2 + \cdots + \sigma_r^2}$$

Application: Image Compression

- View $m \times n$ image as a (real) matrix A, find best rank η approximation by SVD
- Storage $\eta \times (m+n)$ instead of $m \times n$



Application: Image Compression



Cleave Moler Textbooks: `www.mathworks.com/moler/`

Solving Systems of Linear Equations ($Ax = b$)

- Let $A = U\Sigma V^* = \hat{U}\hat{\Sigma}V^* \quad (\mathrm{rank}(A) = r)$
- $Ax = b$ is solvable iif $b \perp \mathrm{null}(A^*)$
- A solution of $Ax = b$, if exists, is given by

  $$\hat{x} = \hat{V}\hat{\Sigma}^{-1}\hat{U}^* b = V\Sigma^+ U^* b = A^+ b,$$

  where $A^+ = V\Sigma^+ U^*$ is the pseudo inverse of A
- The vector $\hat{x} = A^+ b$ represents the uniquely determined solution of $Ax = b$ with minimal euclidean norm
- If $Ax = b$ has no solution, $\hat{x} = A^+ b$ represents its least squares solution with minimal euclidean norm

The QR Factorization

The QR Factorization - Main Idea

- Find orthonormal vectors q_j that span the successive spaces spanned by the columns of A:

$$\langle a_1 \rangle \subseteq \langle a_1, a_2 \rangle \subseteq \langle a_1, a_2, a_2 \rangle \subseteq \cdots$$

- This means that (for full rank A)

$$\langle q_1, q_2, \dots q_j \rangle = \langle a_1, a_2, \dots a_j \rangle, \qquad \text{for } j = 1, \dots, n$$

The QR Factorization - Matrix Form

- In matrix form $\langle q_1, q_2, \dots q_j \rangle = \langle a_1, a_2, \dots a_j \rangle$ becomes

$$\left[\begin{array}{c|c|c|c} a_1 & a_2 & \cdots & a_n \end{array} \right] = \left[\begin{array}{c|c|c|c} q_1 & q_2 & \cdots & q_n \end{array} \right] \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ & r_{22} & \cdots & r_{2n} \\ & & \ddots & \vdots \\ & & & r_{nn} \end{bmatrix}$$

or

$$A = \hat{Q}\hat{R}$$

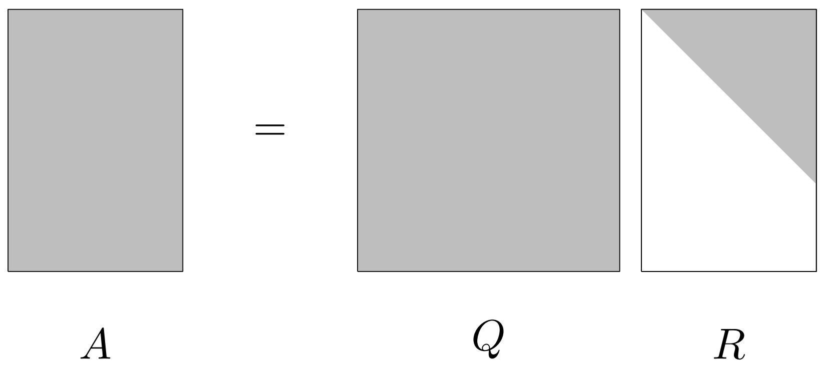
- This is the reduced QR factorization
- Add orthogonal extension to $\hat{Q}$ and add rows to $\hat{R}$ of obtain the full QR factorization

The Full QR Factorization

- Let A be an $m \times n$ matrix. The full QR factorization of A is the factorization $A = QR$, where

  Q is $m \times m$ unitary

  R is $m \times n$ upper-triangular

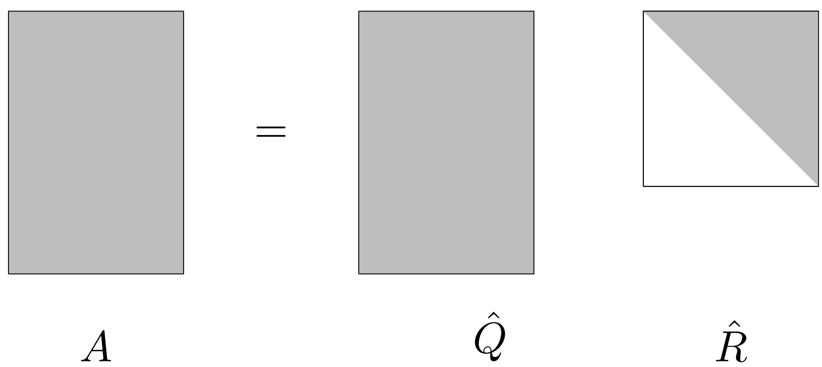


The Reduced QR Factorization

- A more compact representation is the reduced QR factorization $A = \hat{Q}\hat{R}$, where (for $m \geqslant n$)

  $\hat{Q}$ is $m \times n$ with orthonormal columns

  R is $n \times n$ upper-triangular



Gram-Schmidt Orthogonalization ($*$)

- Find new q_j orthogonal to $q_1, \dots, q_{j-1}$ by subtracting components along previous vectors

$$v_j = a_j - (q_1^* a_j)q_1 - (q_2^* a_j)q_2 - \cdots - (q_{j-1}^* a_j)q_{j-1}$$

- Normalize to get $q_j = v_j / \|v_j\|$
- We then obtain a reduced QR factorization $A = \hat{Q}\hat{R}$, with

$$r_{ij} = q_i^* a_j, \qquad (i \neq j)$$

and

$$|r_{jj}| = \left\| a_j - \sum_{i=1}^{j-1} r_{ij} q_i \right\|_2$$

- "Triangular Orthogonalization"

Classical Gram-Schmidt ($*$)

- Straight-forward application of Gram-Schmidt orthogonalization
- Numerically unstable
- Algorithm: Classical Gram-Schmidt

for $j = 1$ to n do
  $v_j = a_j$
  for $i = 1$ to $j - 1$ do
    $r_{ij} = q_i^* a_j$
    $v_j = v_j - r_{ij} q_i$
  end for
  $r_{jj} = \|v_j\|_2$
  $q_j = v_j / r_{jj}$
end for

Existence and Uniqueness

Theorem 0.3: Existence

Every $A \in \mathbb{C}^{m \times n}$ $(m \geqslant n)$ has a full QR factorization and a reduced QR factorization.

Proof: For full rank A, Gram-Schmidt process gives the existence of $A = \hat{Q}\hat{R}$. Otherwise, when $v_j = 0$ choose arbitrary vector orthogonal to previous $q_1, \dots, q_{j-1}$. For full QR, add orthogonal extension to Q (silent columns) and zero rows to R. $\square$

Theorem 0.4: Uniqueness

Each $A \in \mathbb{C}^{m \times n}$ $(m \geqslant n)$ of full rank has a unique $A = \hat{Q}\hat{R}$ with $r_{jj} > 0$.

Proof: Again Gram-Schmidt, $r_{jj} > 0$ determines the sign. $\square$

Classical vs Modified Gram-Schmidt $(*)$

- Some modifications of classical Gram-Schmidt gives modified Gram-Schmidt (but see next slide)
- Modified Gram-Schmidt is numerically stable (less sensitive to rounding errors)
- Algorithm: Classical/Modified Gram-Schmidt

for $j = 1$ to n do
  $v_j = a_j$
  for $i = 1$ to $j-1$ do
    $r_{ij} = q_i^* a_j$ (CGS)
    $r_{ij} = q_i^* v_j$ (MGS)
    $v_j = v_j - r_{ij} q_i$
  end for
  $r_{jj} = \|v_j\|_2$
  $q_j = v_j / r_{jj}$
end for

Implementation of Modified Gram-Schmidt ($*$)

- Algorithm: CGS

for $j = 1$ to n do
  $v_j = a_j$
  for $i = 1$ to $j-1$ do
    $r_{ij} = q_i^* a_j$
    $v_j = v_j - r_{ij} q_i$
  end for
  $r_{jj} = \|v_j\|_2$
  $q_j = v_j / r_{jj}$
end for

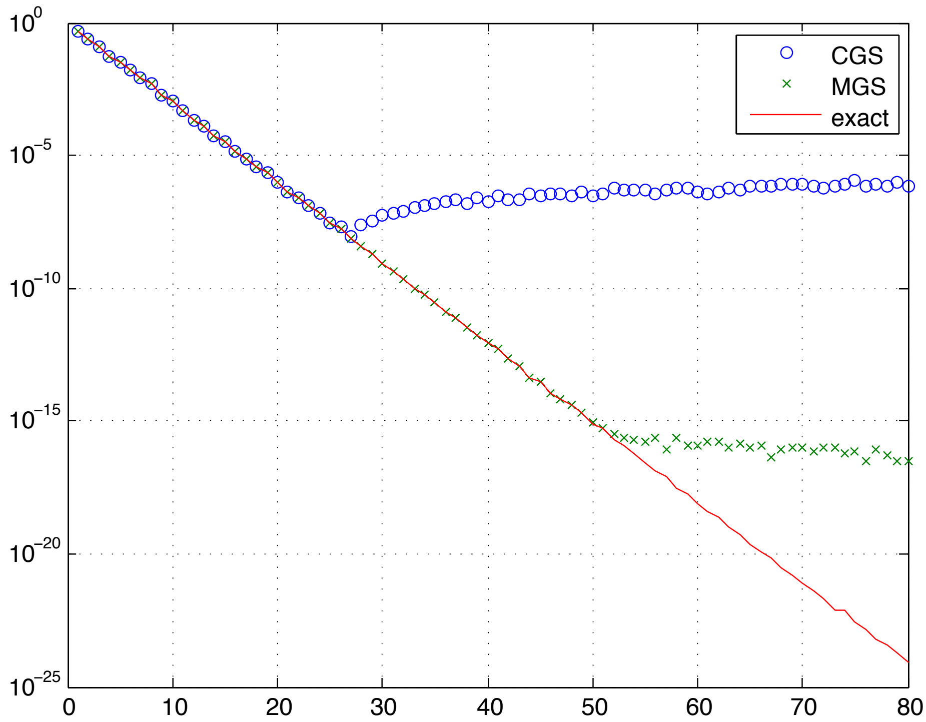
- Algorithm: MGS

for $i = 1$ to n do
  $v_i = a_i$
end for
for $i = 1$ to n do
  $r_{ii} = \|v_i\|_2$
  $q_i = v_i / r_{ii}$
  for $j = i+1$ to n do
    $r_{ij} = q_i^* v_j$
    $v_j = v_j - r_{ij} q_i$
  end for
end for

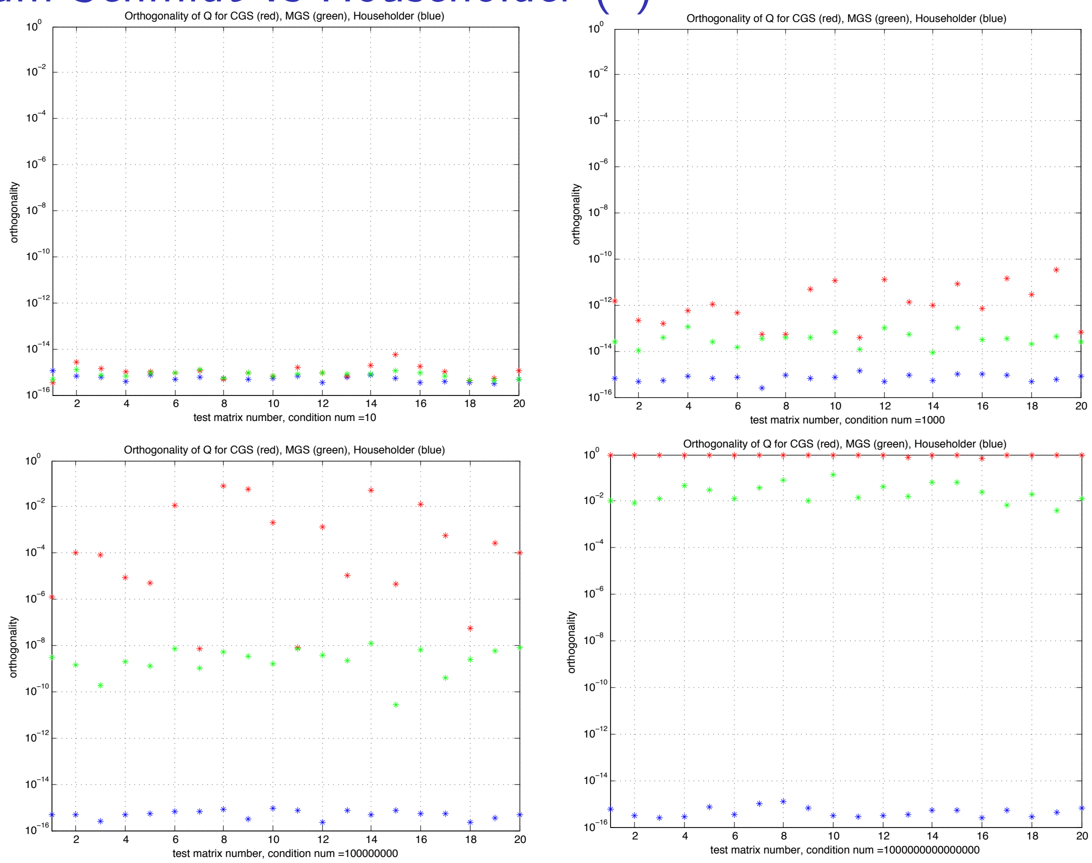
Example: Classical vs Modified Gram-Schmidt ($*$)

```
% Create a random orthogonal matrix Q
n = 80;
[Q,X] = qr(randn(n));
% Make an ill-conditioned R (with diagonal
% entries = 2^-j, j=1,...,n)
R = diag(2.^(-1:-1:-n))*triu(ones(n)+0.1*randn(n));
% Compute QR factorization with classical and with
% modified GS, compare diagonal elements of
% computed R's
A = Q*R;
[QC,RC] = clgs(A);
[QM,RM] = mgs(A);
semilogy(1:n,diag(RC),'o',1:n,diag(RM),'x',1:n,diag(R))
legend('CGS', 'MGS', 'exact')
grid on
```

Example: Classical vs Modified Gram-Schmidt (∗)



Gram-Schmidt vs Householder (∗)



The LU Factorization

The LU Factorization

- Transform $A = \mathbb{R}^{n \times n}$ into upper triangular U by subtracting multiples of rows
- Each L_i introduces zeros below diagonal of column i:

$$\underbrace{L_{n-1} \cdots L_2 L_1}_{L^{-1}} A = U \Rightarrow A = LU \text{ where } L = L_1^{-1} L_2^{-1} \cdots L_{n-1}^{-1}$$

$$\begin{bmatrix} \star & \star & \star & \star \\ \star & \star & \star & \star \\ \star & \star & \star & \star \\ \star & \star & \star & \star \end{bmatrix} \xrightarrow{L_1} \begin{bmatrix} \star & \star & \star & \star \\ 0 & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \end{bmatrix} \xrightarrow{L_2} \begin{bmatrix} \star & \star & \star & \star \\ & \star & \star & \star \\ & 0 & * & * \\ & 0 & * & * \end{bmatrix} \xrightarrow{L_3} \begin{bmatrix} \star & \star & \star & \star \\ & \star & \star & \star \\ & & \star & \star \\ & & 0 & * \end{bmatrix}$$

$A \qquad L_1 A \qquad L_2 L_1 A \qquad L_3 L_2 L_1 A$

- "Triangular triangularization"

The Matrices L_k

- At step k, eliminate elements below A_{kk}:

$$\begin{aligned} x_k &= \begin{bmatrix} x_{1k} & \cdots & x_{kk} & x_{k+1,k} & \cdots & x_{nk} \end{bmatrix}^T \\ L_k x_k &= \begin{bmatrix} x_{1k} & \cdots & x_{kk} & 0 & \cdots & 0 \end{bmatrix}^T \end{aligned}$$

- Each L_i introduces zeros below diagonal of column i:

$$\underbrace{L_{n-1} \cdots L_2 L_1}_{L^{-1}} A = U \Rightarrow A = LU \text{ where } L = L_1^{-1} L_2^{-1} \cdots L_{n-1}^{-1}$$

- The multipliers $\ell_{jk} = x_{jk}/x_{kk}$ appear in L_k:

$$L_k = \begin{bmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & -\ell_{k+1,k} & 1 & & \\ & & \vdots & & \ddots & \\ & & -\ell_{nk} & & & 1 \end{bmatrix}$$

Forming L

- The L matrix contains all the multipliers in one matrix (with plus signs)

$$L = L_1^{-1} L_2^{-1} \cdots L_{n-1}^{-1} = \begin{bmatrix} 1 & & & & \\ \ell_{21} & 1 & & & \\ \ell_{31} & \ell_{32} & 1 & & \\ \vdots & \vdots & \ddots & \ddots & \\ \ell_{n1} & \ell_{n2} & \cdots & \ell_{n,n-1} & 1 \end{bmatrix}$$

- Define $\ell_k = (0, \cdots, 0, \ell_{k+1,k}, \cdots, \ell_{nk})$. Then

$$L_k = I - \ell_k e_k^T,$$

  where e_k is the column vector with 1 in position k and 0 elsewhere
  - First, $L_k^{-1} = I + \ell_k e_k^T$, since $e_k^T \ell_k = 0$ and

$$(I - \ell_k e_k^T)(I + \ell_k e_k^T) = I - \ell_k e_k^T \ell_k e_k^T = I$$

  - Also, $L_k^{-1} L_{k+1}^{-1} = I + \ell_k e_k^T + \ell_{k+1} e_{k+1}^T$, since $e_k^T \ell_{k+1} = 0$ and

$$(I - \ell_k e_k^T)(I + \ell_{k+1} e_{k+1}^T) = I + \ell_k e_k^T + \ell_{k+1} e_{k+1}^T$$

Gaussian Elimination without Pivoting

- Factorize $A \in \mathbb{R}^{n \times n}$ into $A = LU$
- Algorithm: Gaussian Elimination (no pivoting)

$U = A$, $L = I$
for $k = 1$ to $n - 1$ do
    for $j = k + 1$ to n do
        $\ell_{jk} = u_{jk}/u_{kk}$
        $u_{j,k:n} = u_{j,k:n} - \ell_{jk} u_{k,k:n}$
    end for
end for

- The inner loop can be written using matrix operations instead of for-loop

Pivoting (*)

- At step k, we used matrix element k, k as pivot and introduced zeros in entry k of remaining rows

$$\begin{bmatrix} \star & \star & \star & \star & \star \\ & x_{kk} & * & * & * \\ & \star & \star & \star & \star \\ & \star & \star & \star & \star \\ & \star & \star & \star & \star \end{bmatrix} \rightarrow \begin{bmatrix} \star & \star & \star & \star & \star \\ & x_{kk} & \star & \star & \star \\ & 0 & * & * & * \\ & 0 & * & * & * \\ & 0 & * & * & * \end{bmatrix}$$

- But any other element $i \leqslant k$ in column k can be used as pivot:

$$\begin{bmatrix} \star & \star & \star & \star & \star \\ & \star & \star & \star & \star \\ & \star & \star & \star & \star \\ & x_{ik} & * & * & * \\ & \star & \star & \star & \star \end{bmatrix} \rightarrow \begin{bmatrix} \star & \star & \star & \star & \star \\ & 0 & * & * & * \\ & 0 & * & * & * \\ & x_{ik} & \star & \star & \star \\ & 0 & * & * & * \end{bmatrix}$$

Pivoting $(*)$

- Also, any other column $j \leqslant k$ can be used:

$$\begin{bmatrix} \star & \star & \star & \star & \star \\ & \star & \star & \star & \star \\ & \star & \star & \star & \star \\ & \ast & x_{ik} & \ast & \ast \\ & \star & \star & \star & \star \end{bmatrix} \rightarrow \begin{bmatrix} \star & \star & \star & \star & \star \\ & \ast & 0 & \ast & \ast \\ & \ast & 0 & \ast & \ast \\ & \star & x_{ik} & \star & \star \\ & \ast & 0 & \ast & \ast \end{bmatrix}$$

- Choosing different pivots means we can avoid zero or very small pivots
- Instead of using pivots at different entries, change rows or columns and use the standard triangular algorithm (pivoting)
- A computer code might account for the pivoting indirectly instead of actually moving the data

Partial Pivoting $(*)$

- Searching among all valid pivots is expensive (complete pivoting)
- Consider pivots in column k only and interchange rows (partial pivoting)

$$\begin{bmatrix} \star & \star & \star & \star & \star \\ & \star & \star & \star & \star \\ & \star & \star & \star & \star \\ & x_{ik} & \ast & \ast & \ast \\ & \star & \star & \star & \star \end{bmatrix} \xrightarrow{P_1} \begin{bmatrix} \star & \star & \star & \star & \star \\ & x_{ik} & \ast & \ast & \ast \\ & \star & \star & \star & \star \\ & \ast & \ast & \ast & \ast \\ & \star & \star & \star & \star \end{bmatrix} \xrightarrow{L_1} \begin{bmatrix} \star & \star & \star & \star & \star \\ & x_{ik} & \star & \star & \star \\ & 0 & \ast & \ast & \ast \\ & 0 & \ast & \ast & \ast \\ & 0 & \ast & \ast & \ast \end{bmatrix}$$

Pivot selection Row interchange Elimination

- In terms of matrices:

$$L_{n-1}P_{n-1}\cdots L_2P_2L_1P_1A = U$$

The $PA = LU$ Factorization $(*)$

- To combine all L_k and all P_k into matrices, rewrite as

$$L_{n-1}P_{n-1}\cdots L_2P_2L_1P_1A = U$$
$$(\bar{L}_{n-1}\cdots\bar{L}_2\bar{L}_1)(P_{n-1}\cdots P_2P_1)A = U$$

where
$$\bar{L}_k = P_{n-1}\cdots P_{k+1}L_kP_{k+1}^{-1}\cdots P_{n-1}^{-1}$$

- This gives the LU factorization of A

$$PA = LU$$

Gaussian Elimination with Partial Pivoting $(*)$

- Factorize $A \in \mathbb{R}^{n\times n}$ into $PA = LU$
- Algorithm: Gaussian Elimination (partial pivoting)

$U = A,\ L = I,\ P = I$
for $k = 1$ to $n-1$ do
    Select $i \geqslant k$ to maximize $|u_{ik}|$
    $u_{k,k:n} \leftrightarrow u_{i,k:n}$ % interchange two rows
    $\ell_{k,1:k-1} \leftrightarrow \ell_{i,1:k-1}$
    $p_{k,:} \leftrightarrow p_{i,:}$
    for $j = k+1$ to n do
        $\ell_{jk} = u_{jk}/u_{kk}$
        $u_{j,k:n} = u_{j,k:n} - \ell_{jk}u_{k,k:n}$
    end for
end for

Cholesky Factorization for SPD/HPD Matrices ($*$)

- Eliminate below pivot and to the right of pivot:

$$\begin{aligned} A &= \begin{bmatrix} a_{11} & \omega^* \\ \omega & K \end{bmatrix} = \begin{bmatrix} \alpha & 0 \\ \omega/\alpha & I \end{bmatrix} \begin{bmatrix} \alpha & \omega^*/\alpha \\ 0 & K - \omega\omega^*/a_{ii} \end{bmatrix} \\ &= \begin{bmatrix} \alpha & 0 \\ \omega/\alpha & I \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & K - \omega\omega^*/a_{ii} \end{bmatrix} \begin{bmatrix} \alpha & \omega^*/\alpha \\ 0 & I \end{bmatrix} \\ &= R_1^* A_1 R_1 \end{aligned}$$

  where $\alpha = \sqrt{a_{11}}$

- $K - \omega\omega^*/a_{11}$ is a principal submatrix of PD matrix $R_1^* A_1 R_1$, therefore its upper-left entry is positive

- Apply recursively to obtain

$$A = (R_1^* R_2^* \cdots R_n *)(R_n \cdots R_2 R_1) = R^* R, \qquad r_{jj} > 0$$

The Cholesky Factorization Algorithm

- Factorize hermitian positive definite $A \in \mathbb{R}^{n \times n}$ into $A = R^* R$

- Algorithm: Cholesky Factorization ($*$)

  $R = A$
  for $k = 1$ to n do
      for $j = k+1$ to n do
          $r_{j,j:n} = r_{j,j:n} - r_{k,j:n} r_{k,j}^* / r_{kk}$
      end for
      $r_{k,k:n} = r_{k,k:n} / \sqrt{r_{kk}}$
      end for
  end for

- Existence and uniqueness: Every PD matrix has a unique Choleskey factorization

Backslash in MATLAB

- `x=A\b` for dense `A` performs these steps (stopping when successful):
  1. If `A` is upper or lower triangular, solve by back/forward substitution
  2. If `A` is permutation of triangular matrix, solve by permuted back substitution (useful for `[L,U]=lu(A)` since `L` is permuted)
  3. If `A` is symmetric
     - Check if all diagonal elements are positive
     - Try Cholesky, if successful solve by back substitutions
  4. If `A` is Hessenberg (upper triangular plus one subdiagonal), reduce to upper triangular then solve by back substitution
  5. If `A` is square, factorize $PA = LU$ and solve by back substitutions
  6. If `A` is not square, run Householder `QR`, solve least squares problem

Conditioning and Condition Numbers

Conditioning

- Absolute Condition Number of a differentiable problem f at x:

$$\hat{k} = \lim_{\delta \to 0} \sup_{\|\delta x\| \leqslant \delta} \frac{\|\delta f\|}{\|\delta x\|} = \sup_{\delta x} \frac{\|\delta f\|}{\|\delta x\|} = \|J(x)\|,$$

  where the Jacobian $J_{ij} = \partial f_i / \partial x_j$, and the matrix norm is induced by the norms on δf and δx

- Relative Condition Number:

$$k = \sup_{\delta x} \left(\frac{\|\delta f\|}{\|f(x)\|} \Big/ \frac{\|\delta x\|}{\|x\|} \right) = \frac{\|J(x)\|}{\|f(x)\|/\|x\|}$$

Condition of Matrix-Vector Product

- Consider $f(x) = Ax$, with $A \in \mathbb{C}^{m \times n}$

$$k = \frac{\|J(x)\|}{\|f(x)\|/\|x\|} = \|A\| \frac{\|x\|}{\|Ax\|} = [Ax = b] = \|A\| \frac{\|x\|}{\|b\|}$$

- For A square and nonsingular, use $\|x\|/\|Ax\| \leqslant \|A^{-1}\|$:

$$k \leqslant \|A\| \|A^{-1}\|$$

  (equality achieved for the last right singular vector $x = v_m$)

- The condition number of Ax if ∞ if $x \in \text{null}(A)$

- Also the condition number for $f(b) = A^{-1}b$ (solution of linear system $Ax = b$):

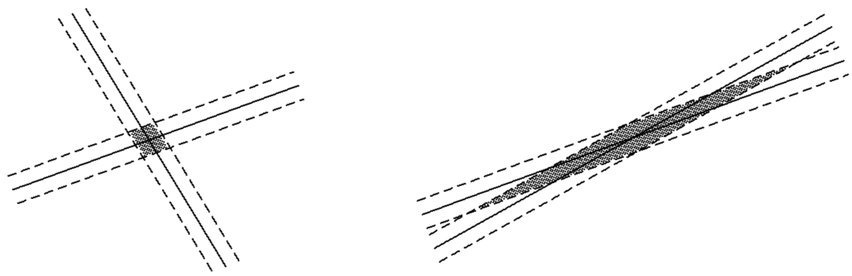
$$k = \|A^{-1}\| \frac{\|b\|}{\|x\|} \leqslant \|A\| \|A^{-1}\|$$

Condition Number of a Matrix

- Condition number of matrix A:

$$k(A) = \|A\|\|A^{-1}\| = [\text{for 2-norm}] = \frac{\sigma_1}{\sigma_m} \geqslant 1$$

- If A is singular we consider, by convention, $k(A) = \infty$
- Measure of uncertainty



well-conditioned ill-conditioned

Condition of System of Equations

- Exercise 0.7: For fixed A, consider $f(b) = A^{-1}b$. Prove that

$$k = \frac{\|\delta x\|}{\|x\|} \bigg/ \frac{\|\delta b\|}{\|b\|} \leqslant k(A).$$

Then, if the input data is accurate to the $\epsilon_{machine}$

$$\frac{\|\delta x\|}{\|x\|} \leqslant k(A)\epsilon_{machine}.$$

- Exercise 0.8 (Theorem 3.1 (QSS, page 62)): Let $A \in \mathbb{C}^{m\times m}$ be a non singular matrix and let $\delta A \in \mathbb{C}^{m\times m}$ be such that $\|A^{-1}\|\|\delta A\| < 1$. Let $Ax = b$ and $(A + \delta A)(x + \delta x) = b + \delta b$. Prove that

$$\frac{\|\delta x\|}{\|x\|} \leqslant \frac{k(A)}{1 - k(A)\frac{\|\delta A\|}{\|A\|}} \left(\frac{\|\delta A\|}{\|A\|} + \frac{\|\delta b\|}{\|b\|} \right),$$

where $k(A)$ is the condition number of the matrix A.

Example: Condition of Hilbert system

```
% Initialise settings, constants and vectors

clc; clear; close all;

N = 12; error = zeros(1,N-1); estimate = zeros(1,N-1);

% Loop on the order of the matrix

for n = 2:N
    H = hilb(n);
    x = ones(n,1); b = H*x;     % Exact values
    xbar = H\b; bbar = H*xbar; % Computed values
    % Compute error and error estimate
    error(n-1) = norm(x-xbar)/norm(x);
    estimate(n-1) = cond(H)*norm(b-bbar)/norm(b);
end

semilogy(2:n,error,'-o',2:n,estimate,'-x')

legend('error', 'estimate')

xlabel('order'), ylabel('relative error')
```

Example: Condition of Hilbert system