Bi7740: Scientific computing

Systems of linear equations

Vlad Popovici
popovici@iba.muni.cz

Institute of Biostatistics and Analyses
Masaryk University, Brno

Additional references:

- Golub, Van Loan, *Matrix Computations*, Johns Hopkins Univ. Press, 3rd Ed. 1996

Outline

Outline

Outline

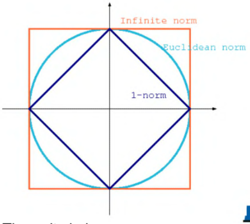
Vectors and norms

Let $\mathbf{x}$ be a vector, $\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = [x_1, \dots, x_n]^T$. The p–norm is defined as

$$||\mathbf{x}||_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{\frac{1}{p}}$$

Special cases:

- $p = 1$: (Manhattan or city-block norm) $||\mathbf{x}||_1 = \sum_i |x_i|$
- $p = 2$: (Euclidean norm) $||\mathbf{x}||_2 = \sqrt{\sum_i x_i^2}$
- $p \to \infty$: (∞–norm) $||\mathbf{x}||_\infty = \max_i |x_i|$



The unit circles.

Vector norms - properties

$\forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ and for any norm,

- $||\mathbf{x}|| \geq 0$ with $||\mathbf{x}|| = 0 \Leftrightarrow \mathbf{x} = 0$
- $||\alpha \mathbf{x}|| = |\alpha| \cdot ||\mathbf{x}||, \forall \alpha$
- $||\mathbf{x} + \mathbf{y}|| \leq ||\mathbf{x}|| + ||\mathbf{y}||$ (triangle inequality); also $|\,||\mathbf{x}|| - ||\mathbf{y}||\,| \leq ||\mathbf{x} - \mathbf{y}||$
- $||\mathbf{x}||_1 \geq ||\mathbf{x}||_2 \geq ||\mathbf{x}||_\infty$
- $||\mathbf{x}||_1 \leq \sqrt{n}||\mathbf{x}||_2$, $||\mathbf{x}||_2 \leq \sqrt{n}||\mathbf{x}||_\infty$ $\rightarrow$ norms differ by at most a constant, hence they are equivalent

MATLAB: `norm(x, p)`

Matrix norms

Let

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{1n} \\ & \dots & \\ a_{n1} & a_{n2} & a_{nn} \end{bmatrix}$$

be a square matrix.

- defined based on a vector norm
- $$||\mathbf{A}|| = \max_{\mathbf{x} \neq 0} \frac{||\mathbf{A}\mathbf{x}||}{||\mathbf{x}||}$$
- the maximum "stretching" applied to a vector by the matrix $\mathbf{A}$
- $||\mathbf{A}||_1 = \max_j \sum_{i=1}^{n} |a_{ij}|$ (maximum absolute column sum)
- $||\mathbf{A}||_\infty = \max_i \sum_{j=1}^{n} |a_{ij}|$ (maximum absolute row sum)
- $||\mathbf{A}||_2 =$? (we'll see it later)

Matrix norms - properties

Let **A** and **B** be two square matrices

- $||\mathbf{A}|| > 0$ if $\mathbf{A} \neq 0$
- $||\alpha\mathbf{A}|| = |\alpha| \cdot ||\mathbf{A}||$, for any scalar α
- $||\mathbf{A} + \mathbf{B}|| \leq ||\mathbf{A}|| + ||\mathbf{B}||$
- $||\mathbf{A} \cdot \mathbf{B}|| \leq ||\mathbf{A}|| \cdot ||\mathbf{B}||$
- $||\mathbf{Ax}|| \leq ||\mathbf{A}|| \cdot ||\mathbf{x}||$ for any vector $\mathbf{x}$

MATLAB: `norm(A, p)`

Outline

Linear systems

In general, a system of linear equations has the form:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\cdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m \end{aligned}$$

or, in matrix format,

$$\mathbf{Ax} = \mathbf{b}$$

where **A** is an $m \times n$ matrix (say, $\mathbf{A} \in \mathcal{M}_{m,n}(\mathbb{R})$), **b** and **x** are vectors with m and n elements, respectively.
In other words: *can the vector* **b** *be expressed as a linear combination of columns of matrix* **A**?

In MATLAB

- general operator "matrix division" \
- this is a wrapper for various algorithms - some we will discuss
- MATLAB: `x = A \ b`

Square matrices case ($m = n$)

$\mathbf{A} \in \mathcal{M}_{n,n}(\mathbb{R})$ is singular if it has any of the following *equivalent* properties:

- $\mathbf{A}$ has no inverse ($\mathbf{A}^{-1}$ does not exist)
- $\det(A) = 0$
- $\text{rank}(A) < n$ (rank: maximum number of rows or columns that are linearly independent)
- $\mathbf{Az} = \mathbf{0}$ for some vector $\mathbf{z} \neq \mathbf{0}$

Otherwise, the matrix is nonsingular.
If $\mathbf{A}$ is nonsingular, there is a unique solution; otherwise, depending on $\mathbf{b}$, there might be zero or infinitely many solutions.

Geometrical interpretation (2D):

- a linear equation defines a line
- if **A** is nonsingular, the two lines intersect
- if **A** is singular, the two lines may be parallel (no solution) or identical (infinitely many solutions)

If **A** is singular and $\mathbf{b} \in \text{span}(\mathbf{A})$ the system is *consistent* and has infinitely many solutions. ($\text{span}(A)$ is the vector space generated by the columns of **A**.)

Example - nonsingular matrix

- let $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ and $\mathbf{b} = \begin{bmatrix} -1 \\ -1 \end{bmatrix}$, then **A** is nonsingular and there is a unique solution, $\mathbf{x} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$

  Naive Matlab solution:

```
>> A=[1 2; 3 4]; b=[-1;-1];
>> x = inv(A)*b;
>> x

x =

    1.0000
   -1.0000
```

Example - singular matrix

- let $A = \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}$ and $\mathbf{b} = \begin{bmatrix} -1 \\ -1 \end{bmatrix}$, then **A** is nonsingular and there is a unique solution, $\mathbf{x} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$

  Naive Matlab solution:

```
>> A=[1 2; 2 4]; b=[-1;-1];
>> x = inv(A)*b
Warning: Matrix is singular to working precision.

x =

  -Inf
  -Inf
```

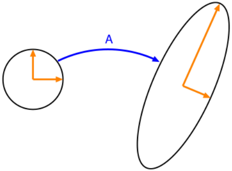
Outline

Singularity, norm and conditioning

- condition number of a nonsingular square matrix is

$$\text{cond}(A) = ||\mathbf{A}|| \cdot ||\mathbf{A}^{-1}||$$

- convention: $\text{cond}(\mathbf{A}) = \infty$ for singular $\mathbf{A}$
- ratio between maximum streching and maximum shrinking of a nonzero vector



- *large* $\text{cond}(\mathbf{A})$ indicates a matrix *close to singularity*
- small $\det(\mathbf{A})$ does not imply large $\text{cond}(\mathbf{A})$

Condition number - properties

- $\text{cond}(\mathbf{A}) \geq 1$
- $\text{cond}(I) = 1$ (I is the identity matrix - Matlab: `eye(n)`)
- $\text{cond}(\alpha\mathbf{A}) = \text{cond}(\mathbf{A})$, for any $\mathbf{A}$ and scalar α
- for a diagonal matrix $\mathbf{D} = diag(d_i)$, $d_i \neq 0$ we have $\text{cond}(\mathbf{D}) = \frac{\max|d_i|}{\min|d_i|}$
- condition number is used for assessing the accuracy of the solutions to linear systems

Condition number:

- exact computation requires matrix inverse:
  - $||\mathbf{A}||$ is easy to compute
  - computing at low cost $||\mathbf{A}^{-1}||$ is difficult $\rightarrow$ expensive (even more than finding the solutions to the problem) and prone to numerical instability
- in practice: estimated as a byproduct of the solution process

One approach: find lower bounds on $||\mathbf{A}^{-1}||$ and, thus, on cond($\mathbf{A}$). If $\mathbf{Ax} = \mathbf{y}$ it follows that

$$\frac{||\mathbf{x}||}{||\mathbf{y}||} \leq ||\mathbf{A}^{-1}||,$$

with "=" achieved for some optimal $\mathbf{y}$. So one needs to find $\mathbf{y}$ such that the lhs above is maximized to get a good estimate of $||\mathbf{A}^{-1}||$.

MATLAB: `cond()` and `condest()`.

Ill-conditioned matrices - example

Consider the *Hilbert matrix* **H** with elements $h_{ij} = \frac{1}{i+j-1}$. It arises, for example, from least square approximation of functions by polynomials, and

$$h_{ij} = \int_0^1 x^{i+j} dx$$

In MATLAB use the `hilb` and `invhilb` for **H** and $\mathbf{H}^{-1}$ respectively.

```
for n = 5:10
    H = hilb(n);
    invH = invhilb(n);  % exact inverse for n < 15!
    c  = cond(H);
    d1 = det(H) * det(inv(H));
    d2 = det(H) * det(invH);
    fprintf('n=%2d\tcond=%e\tdet1=%10.8f\tdet2=%10.8f\n', ...
        n, c, d1, d2);
end
```

```
n= 5 cond=4.766073e+05 det1=1.00000000 det2=1.00000000
n= 6 cond=1.495106e+07 det1=0.99999997 det2=1.00000000
n= 7 cond=4.753674e+08 det1=0.99977251 det2=1.00000000
n= 8 cond=1.525758e+10 det1=0.80314726 det2=1.00000007
n= 9 cond=4.931544e+11 det1=-31.86617838 det2=1.00000344
n=10 cond=1.602457e+13 det1=-150414807.25005761 ...
    det2=1.00011969
```

Outline

Accuracy of solutions

- condition number $\rightarrow$ error bounds
- let $\mathbf{x}$ be the solution to $\mathbf{Ax} = \mathbf{b}$ and $\hat{\mathbf{x}}$ the solution to $\mathbf{A}\hat{\mathbf{x}} = \mathbf{b} + \Delta\mathbf{b}$
- let $\Delta\mathbf{x} = \hat{\mathbf{x}} - \mathbf{x}$, then

$$\mathbf{b} + \Delta\mathbf{b} = \mathbf{Ax} + \mathbf{A}\Delta\mathbf{x},$$

from which

$$\frac{||\Delta\mathbf{x}||}{||\mathbf{x}||} \leq \text{cond}(\mathbf{A}) \frac{||\Delta\mathbf{b}||}{||\mathbf{b}||}$$

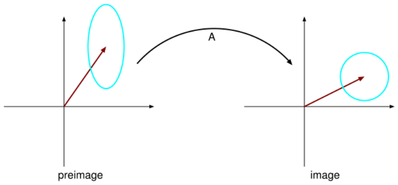
HOMEWORK: prove the above relation.

$$\frac{||\Delta \mathbf{x}||}{||\mathbf{x}||} \leq \text{cond}(\mathbf{A}) \frac{||\Delta \mathbf{b}||}{||\mathbf{b}||}$$

Relative change in solution

The condition number bounds the relative changes in the solution due to a relative change in rhs, *regardless of the algorithm used to compute the solution*.

The condition number cond($\mathbf{A}$) defines the uncertainty in $\mathbf{x}$, given the uncertainty in $\mathbf{b}$.



Similarly, if $(\mathbf{A}+\mathbf{D})\hat{\mathbf{x}} = \mathbf{b}$, then

$$\frac{||\Delta \mathbf{x}||}{||\hat{\mathbf{x}}||} \leq \text{cond}(\mathbf{A})\frac{||\mathbf{D}||}{||\mathbf{A}||}$$

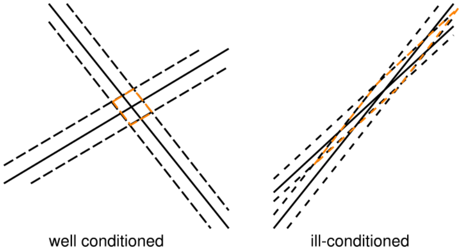
- if data ($\mathbf{A}$, $\mathbf{b}$) is accurate to machine precision, then the relative error in solution can be approximated by

$$\frac{||\hat{\mathbf{x}} - \mathbf{x}||}{||\mathbf{x}||} \approx \text{cond}(A)\epsilon_{\text{mach}}$$

  i.e. the solution loses about $\log_{10}(\text{cond}(\mathbf{A}))$ decimal digits of accuracy with respect to input data
- the analysis is about relative error in the *largest* components of the solution vector; relative error can be larger in the smaller components.

- the condition number is affected by the scaling of **A**, so one way of improving the solution is by rescaling - this does not improve a matrix near singularity.
- example: $\mathbf{A} = \begin{bmatrix} 1 & 0 \\ 0 & \epsilon \end{bmatrix}, \mathbf{b} = \begin{bmatrix} 1 \\ \epsilon \end{bmatrix}$
- the matrix **A** is ill-conditioned for small ϵ: $\text{cond}(\mathbf{A}) = 1/\epsilon$.
- by scaling the 2nd eq with $1/\epsilon$, the matrix becomes well conditioned.
- in general, it is more difficult...

Example:



Residuals

- residual vector: $\mathbf{r} = \mathbf{b} - \mathbf{A}\hat{\mathbf{x}}$ for $\hat{\mathbf{x}}$ being the approximate solution to $\mathbf{A}\mathbf{x} = \mathbf{b}$
- theoretically: if $\mathbf{A}$ is nonsingular then $||\hat{\mathbf{x}} - \mathbf{x}|| = 0 \Leftrightarrow ||\mathbf{r}|| = 0$
- practically, small residual is not necessarily equivalent to small error
- since

$$\frac{||\Delta \mathbf{x}||}{||\hat{\mathbf{x}}||} \leq \text{cond}(\mathbf{A}) \frac{||\mathbf{r}||}{||\mathbf{A}|| \cdot ||\hat{\mathbf{x}}||}$$

  small relative residual implies small relative error, *only if* $\mathbf{A}$ is well-conditioned

Residuals - backward error analysis

- let $\mathbf{D}$ be the "delta" matrix, such that $\hat{\mathbf{x}}$ is the exact solution of

$$(\mathbf{A} + \mathbf{D})\hat{\mathbf{x}} = \mathbf{b},$$

then

$$\frac{||\mathbf{r}||}{||\mathbf{A}|| \cdot ||\hat{\mathbf{x}}||} \leq \frac{||\mathbf{D}||}{||\mathbf{A}||}$$

- large *relative residual* implies large backward error and indicates an unstable algorithm
- stable algorithms yield small relative residuals, regardless conditioning of nonsingular $\mathbf{A}$

Outline

1. Systems of linear equations - reminder
   - Norms
   - Linear systems
   - Conditioning
   - Accuracy
2. Solving linear systems
   - Diagonal systems
   - Triangular systems
   - Gaussian elimination
3. Special cases
   - Symmetric positive definite systems
4. Examples and applications

General strategy

- transform the system (mainly **A**) such that the solution is easier to compute (but unchanged)
- if **M** is a nonsingular matrix the systems

$$\mathbf{Ax} = \mathbf{b}$$

and

$$\mathbf{MAx} = \mathbf{Mb}$$

have the same solution. HOMEWORK: prove it!
- trivial transformations:
  - permutation of rows in the system: use a permutation matrix (has exactly one 1 in each row and column, rest is 0).
  - diagonal scaling: may improve the accuracy

A few relevant functions in MATLAB

Please, use `help <name>` for details!

- `linsolve`: solves linear systems $\mathbf{Ax} = \mathbf{B}$ via various methods. You can specify the properties of $\mathbf{A}$.
- `\` operator is a wrapper for various methods
- `lu` computes LU factorization
- `triu` returns upper triangular part of a matrix
- `tril` returns lower triangular part of a matrix
- `diag` returns the diagonal of a matrix
- `cond, condest` used for estimating the condition number

Outline

Diagonal systems

The simplest linear system is

$$\begin{bmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & \dots & 0 \\ & & \ddots & \\ 0 & 0 & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

with obvious solution $\mathbf{x} = [b_i / a_{ii}]_i$.

```
function x = diagsolve(A, b)
% Solve A x = b for a diagonal matrix A.
  d = diag(A);
  if any(d == 0), error('A is singular!'), end
  x = b ./ d;
return
```

Outline

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &= b_1 \\ a_{22}x_2 + a_{23}x_3 &= b_2 \\ a_{33}x_3 &= b_3 \end{aligned}$$

which is equivalent to

$$\begin{array}{llllll} a_{11}x_1 & & & = b_1 & -a_{12}x_2 & -a_{13}x_3 \\ & a_{22}x_2 & & = b_2 & -a_{23}x_3 & \\ & & a_{33}x_3 & = b_3 & & \end{array}$$

Triangular systems

- **A** is *lower triangular* if $a_{ij} = 0$ for $i < j$ or *upper triangular* if $a_{ij} = 0$ for $i > j$
- solution is obtained by *back-substitution*: for

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ 0 & a_{22} & a_{23} & \dots & a_{2n} \\ 0 & 0 & a_{33} & \dots & a_{3n} \\ & & & \ddots & \\ 0 & 0 & 0 & \dots & a_{nn} \end{bmatrix}$$

$$x_n = b_n / a_{nn}$$

$$x_i = \left(b_i - \sum_{j=i+1}^{n} a_{ij} x_j \right) / a_{ii}, \text{for } i = n-1, n-2, \dots, 1$$

Back-substitution algorithm

(not vectorized!)

Algorithm: Back-substitution algorithm

for $j = n$ **to** 1 **do**
  if $a_{jj} = 0$ **then**
    `stop;`
  end if
  $x_j \leftarrow b_j / a_{jj}$;
  for $i = 1$ **to** $j - 1$ **do**
    $b_i \leftarrow b_i - a_{ij} x_j$;
  end for
end for

Exercise

- derive the forward substitution method for lower triangular matrices
- implement in MATLAB the functions `fwsolve` and `bksolve` for forward and backward substitution

Elementary elimination matrices

Goal

Find tranformations of nonsingular matrices that would lead to triangular systems.

Example: let $\mathbf{z} = [z_1, z_2]^T$ with $z_1 \neq 0$, then

$$\begin{bmatrix} 1 & 0 \\ -z_2/z_1 & 1 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} z_1 \\ 0 \end{bmatrix}$$

$\rightarrow$ use linear combinations or rows

In general,

$$\mathbf{M}_k\mathbf{z} = \begin{bmatrix} 1 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 1 & 0 & \dots & 0 \\ 0 & \dots & -m_{k+1} & 1 & \dots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & -m_n & 0 & \dots & 1 \end{bmatrix} \begin{bmatrix} z_1 \\ \vdots \\ z_k \\ z_{k+1} \\ \vdots \\ z_n \end{bmatrix} = \begin{bmatrix} z_1 \\ \vdots \\ z_k \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

where $m_i = z_i/z_k$, for $i = k+1, \dots, n$.

- pivot: z_k
- Gaussian transformation or elementary elimination transformation: $\mathbf{M}_k$

Properties of the Gaussian transformation

- $\mathbf{M}_k$ is nonsingular (it is lower triangular, full rank matrix)
- $\mathbf{M}_k = \mathbf{I} - \mathbf{m}\mathbf{e}_k^T$, where $\mathbf{m} = [0, \ldots, 0, m_{k+1}, \ldots, m_n]^T$ and $\mathbf{e}_k$ is the $k-$th column of the identity matrix
- $\mathbf{M}_k^{-1} = \mathbf{I} + \mathbf{m}\mathbf{e}_k^T$: just the sign is changed for the inverse. Denote $\mathbf{L}_k = \mathbf{M}_k$
- if $\mathbf{M}_j = \mathbf{I} - \mathbf{t}\mathbf{e}_j^T, j > k$, then

$$\mathbf{M}_k\mathbf{M}_j = \mathbf{I} - \mathbf{m}\mathbf{e}_k^T + \mathbf{t}\mathbf{e}_j^T,$$

  so the result is sort of "union" of the two matrices. Note that the order of multiplication is important.
- a similar result holds for the inverses

Outline

Gaussian elimination

- transform the system $\mathbf{Ax} = \mathbf{b}$ into a triangular system:
  - choose $\mathbf{M}_1$ with a_{11} as pivot to eliminate the 1st column below a_{11}. The new system is $\mathbf{M}_1\mathbf{Ax} = \mathbf{M}_1\mathbf{b}$. The solution stays the same.
  - next choose $\mathbf{M}_2$ with a_{22} as pivot to eliminate the 2nd colum below a_{22}. The new system is $\mathbf{M}_2\mathbf{M}_1\mathbf{Ax} = \mathbf{M}_2\mathbf{M}_1\mathbf{b}$. The solution stays the same.
  - ... until we get a triangular system
- solve the system

$$\mathbf{M}_{n-1}\dots\mathbf{M}_1\mathbf{Ax} = \mathbf{M}_{n-1}\dots\mathbf{M}_1\mathbf{b}$$

by back-substitution

LU factorization

- let $\mathbf{M} = \mathbf{M}_{n-1} \dots \mathbf{M}_1$ and $\mathbf{L} = \mathbf{M}^{-1}$
- $\mathbf{L} = (\mathbf{M}_{n-1} \dots \mathbf{M}_1)^{-1} = \mathbf{M}_1^{-1} \dots \mathbf{M}_{n-1}^{-1} = \mathbf{L}_1 \dots \mathbf{L}_{n-1}$ which is unit lower triangular.
- by design, $\mathbf{U} = \mathbf{MA}$ is upper triangular
- then $\boxed{\mathbf{A} = \mathbf{M}^{-1}\mathbf{U} = \mathbf{LU}}$ with $\mathbf{L}$ lower triangular and $\mathbf{U}$ upper triangular
- Gaussian elimination is a factorization of a matrix as a product of two triangular matrices: LU factorization
- LU factorization is *unique up to a scaling factor of diagonal scaling of factors*

- if **A** is factorized into **LU**, the system becomes $\mathbf{LUx} = \mathbf{b}$ and is solved by forward-substitution (reverse order of backward s.) in lower triangular system $\mathbf{Ly} = \mathbf{b}$ followed by back-substitution in $\mathbf{Ux} = \mathbf{y}$
- Gaussian elimination and LU factorization express the same solution process
- MATLAB example:

```
>> A = [0 1 1; 2 -1 -1; 1 1 -1]; b = [2 0 1]';
>> [L,U] = lu(A);
>> y = L \ b; x = U \ y;
```

- Note: $\boxed{\det(\mathbf{A}) = \det(\mathbf{L})\det(\mathbf{U})}$
- if at any stage, the leading entry on the diagonal is zero → cannot choose the pivot → interchange the row with some row below with a non-zero pivot
- if there is no way to choose a proper pivot, the matrix **U** will be singular
- but the factorization can be performed! the back-substitution will fail however.

Experiment

(from C. Van Loan, "Introduction to scientific computing")

Consider the system

$$\begin{bmatrix} \epsilon & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1+\epsilon \\ 2 \end{bmatrix}$$

with the solution $[1\ 1]^T$.
Write a MATLAB code to solve it using LU factorization, for $\epsilon = 10^{-2}, 10^{-4}, \ldots, 10^{-18}$.
Discuss the results!

Another application of LU decomposition

Consider you have to compute the scalar

$$\alpha = \mathbf{z}^T \mathbf{A}^{-1} \mathbf{b} \in \mathbb{R},$$

with $\mathbf{z}, \mathbf{b} \in \mathbb{R}^N$ and $\mathbf{A} \in \mathbb{R}^{n \times n}$ nonsingular.
But

$$\mathbf{x} = \mathbf{A}^{-1} \mathbf{b}$$

is the solution of the linear system $\mathbf{Ax} = \mathbf{b}$. So, you should use LU decomposition, compute $\mathbf{x}$ and then $\alpha = \mathbf{z}^T \mathbf{x}$. In MATLAB:

```
[L,U] = lu(A);
y = L \ b; x = U \ y;
alpha = z' * x;
```

Improving stability

- chose the pivot to minimize error propagation
- choose the entry of largest magnitude on or below the diagonal as pivot
- this is called partial pivoting
- each $\mathbf{M}_k$ is preceded by a permutation matrix $\mathbf{P}_k$ to interchange rows
- still $\mathbf{MA} = \mathbf{U}$, but $\mathbf{M} = \mathbf{M}_{n-1}\mathbf{P}_{n-1} \dots \mathbf{M}_1\mathbf{P}_1$
- $\mathbf{L} = \mathbf{M}^{-1}$ is triangular, but not necessarily *lower* triangular
- in general
$$(\mathbf{P}_{n-1} \dots \mathbf{P}_1)\mathbf{A} = \mathbf{PA} = \mathbf{LU}$$
- check again previous MATLAB example
- try `[L, U, P] = lu(A);`

- if the pivot is sought as the largest entry in the entire unreduced submatrix, then you have complete pivoting
- requires permutations or rows AND columns
- there are 2 permutations matrices, **P**, **Q**, such that

$$\mathbf{PAQ} = \mathbf{LU}$$

- better numerical stability, but much more expensive in computation
- in general, only partial pivoting is used with Gaussian elimination
- in MATLAB is implemented only for *sparse matrices*. See `help lu`

Pivoting is not required if:

- the matrix is *diagonally dominant*:

$$\sum_{i=1, i\neq j}^{n} |a_{ij}| < |a_{jj}|, \quad j = 1, \ldots, n$$

- the matrix is *symmetric positive definite*:

$$\mathbf{A} = \mathbf{A}^T \text{ and } \mathbf{x}^T\mathbf{A}\mathbf{x} > 0, \forall \mathbf{x} \neq \mathbf{0}$$

Examples of symmetric positive (semi-)definite matrices from practice?

Residuals

- $\mathbf{r} = \mathbf{b} - \mathbf{A}\hat{\mathbf{x}}$ where $\hat{\mathbf{x}}$ was obtained by Gaussian elimination
- it can be shown that

$$\frac{||\mathbf{r}||}{||\mathbf{A}||\ ||\hat{\mathbf{x}}||} \leq \frac{||\mathbf{E}||}{||\mathbf{A}||} \leq \rho n \epsilon_{\text{mach}}$$

  where $\mathbf{E}$ is the backward error in data matrix: $(\mathbf{A} + \mathbf{E})\hat{\mathbf{x}} = \mathbf{b}$ and $\rho = \max(u_{ij}) / \max(a_{ij})$ is the *growth factor*
- without pivoting, ρ is unbounded so the algorithm is unstable
- with partial pivoting, $\rho \leq 2^{n-1}$
- in practice, $\rho \approx 1$, so $\frac{||\mathbf{r}||}{||\mathbf{A}||\ ||\hat{\mathbf{x}}||} \lessapprox n\epsilon_{\text{mach}}$

Residuals, cont'd

- Gaussian elimination with partial pivoting yields small relative residuals, *regardless of the conditioning*
- however, computed solution is close to real solution only if the system is well-conditioned
- yet a smaller growth factor can be obtained with complete pivoting, but the extra cost may not be worth

Example: in a 3-digit decimal arithmetic, solve

$$\begin{bmatrix} 0.641 & 0.242 \\ 0.321 & 0.121 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0.883 \\ 0.442 \end{bmatrix}$$

- the exact solution is $[1\ 1]^T$
- the Gaussian elimination leads to $\hat{\mathbf{x}} = [0.782\ 1.58]^T$
- the exact residual is $\mathbf{r} = [-0.000622\ \ -0.000202]^T \rightarrow$ as small as can be expected with 3 digits precision
- the error is large: $||\hat{\mathbf{x}} - \mathbf{x}|| = 0.6196$ which is $\approx 62\%$ relative error!
- this is because of ill-conditioning, $\text{cond}(\mathbf{A}) > 4000$

What happend? The Gaussian elimination led to

$$\begin{bmatrix} 0.641 & 0.242 \\ 0 & 0.000242 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0.883 \\ -0.000383 \end{bmatrix}$$

so x_2 was the result of the division of quantities below ϵ_{mach}, yielding an arbitrary result. The x_1 is computed to satisfy the 1st eq., resulting in small residual but large error.

Implementation and complexity

The general form of the Gaussian elimination is

```
for i do
    for j do
        for k do
            a_ij ← a_ij − (a_ik / a_kk) a_kj
        end for
    end for
end for
```

- order of the loops is not important (for the final result)
- ...but, depending on the memory storage, they have different performance

Implementation and complexity (cont'd)

- there are about $n^3/3$ floating-point operations $\rightarrow$ the complexity is $\mathcal{O}(n^3)$
- the forward-/back-substitutions require about n^2 multiplications and n^2 additions (for a single $\mathbf{b}$)
- if you try to invert $\mathbf{A}$, $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$, you need n^3 operations $\rightarrow 3\times$ more the LU factorization
- inversion is less precise: difference between $3^{-1} \times 18$ and $18/3$ in fixed-precision arithmetic
- matrix iinversion is convenient in formulas, but in practice you do factorizations!
- Ex: $\mathbf{A}^{-1}\mathbf{B}$ should use LU factorization of $\mathbf{A}$ and then forward- and back-substitutions with columns of $\mathbf{B}$

Gauss-Jordan elimination

- idea: for each element of the diagonal, eliminate all the elements below AND above in the column using combinations of rows
- the elimination matrix has the form

$$\begin{bmatrix} 1 & \dots & 0 & -m_1 & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 1 & -m_{k-1} & 0 & \dots & 0 \\ 0 & \dots & 0 & 1 & 0 & \dots & 0 \\ 0 & \dots & 0 & -m_{k+1} & 1 & \dots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & -m_n & 0 & \dots & 1 \end{bmatrix}$$

  where $m_i = a_i/a_k$ for $i = 1, \dots, n$
- do the same to the right hand side term, too

Gauss-Jordan elimination, cont'd

- the result is a diagonal matrix on lhs
- the solution is obained by dividing the entries on the transformed rhs by the terms of the diagonal
- it requires $n^3/2$ multiplications and the same number of additions $\rightarrow$ 50% more expensive than LU decomposition
- despite being more expensive, it is sometimes preferred to LU decomposition for parallel implementations
- if the rhs is initialized with an identity matrix, after G-J elimination the rhs becomes $\mathbf{A}^{-1}$

Solving series of similar problems

- idea: try to reuse as much as possible from previous computations
- if only rhs changes, LU decomposition does not have to be recomputed
- if $\mathbf{A}$ suffers only rank one changes, one can still use pre-computed $\mathbf{A}^{-1}$ (Sherman-Morrison formula):

$$(\mathbf{A} - \mathbf{u}\mathbf{v}^T)^{-1} = \mathbf{A}^{-1} + \mathbf{A}^{-1}\mathbf{u}(1 - \mathbf{v}^T\mathbf{A}^{-1}\mathbf{u})^{-1}\mathbf{v}^T\mathbf{A}^{-1}$$

- this has a complexity of $\mathcal{O}(n^2)$ compared to $\mathcal{O}(n^3)$ that is needed by a new inversion

For a modified equation,

$$(\mathbf{A} - \mathbf{u}\mathbf{v}^T)\mathbf{x} = \mathbf{b}$$

the solution is

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b} + \mathbf{A}^{-1}\mathbf{u}(1 - \mathbf{v}^T\mathbf{A}^{-1}\mathbf{u})^{-1}\mathbf{v}^T\mathbf{A}^{-1}\mathbf{b}$$

and is solved by the following procedure

- solve $\mathbf{Az} = \mathbf{u}$, so $\mathbf{z} = \mathbf{A}^{-1}\mathbf{u}$
- solve $\mathbf{Ay} = \mathbf{b}$, so $\mathbf{y} = \mathbf{A}^{-1}\mathbf{b}$
- compute $\mathbf{x} = \mathbf{y} + ((\mathbf{v}^T\mathbf{y})/(1 - \mathbf{v}^T\mathbf{z}))\mathbf{z}$

If $\mathbf{A}$ is already factored, this approach has a complexity $\mathcal{O}(n^2)$

Comments on scaling

- theoretically, multiplying the terms on diagonal of **A** and corresponding entries of **b** would not change the solution
- in practice, it affects conditioning, choice of pivot and, by consequence, accuracy
- Example:

$$\begin{bmatrix} 1 & 0 \\ 0 & \epsilon \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ \epsilon \end{bmatrix}$$

  is ill-conditioned for small ϵ, since $\text{cond}(\mathbf{A}) = 1/\epsilon$. It becomes well-conditioned if the second equation is multiplied by $1/\epsilon$.

Iterative refinements

- let $\mathbf{x}_0$ be the approximate solution to $\mathbf{Ax} = \mathbf{b}$ and $\mathbf{r}_0 = \mathbf{b} - \mathbf{Ax}_0$ be the corresponding residual
- let then $\mathbf{z}_0$ be the solution to $\mathbf{Az} = \mathbf{r}_0$
- an improved approximate solution is then $\mathbf{x}_1 = \mathbf{x}_0 + \mathbf{z}_0$

  HOMEWORK: prove that $\mathbf{Ax}_1 = \mathbf{b}$
- repeat until convergence
- the process needs higher precision for computing a useful residual
- not often used, but sometimes useful

Outline

Special forms of linear systems

For some special cases of **A** storage and computation time can be saved.
For example, if **A** is

- symmetric: $\mathbf{A} = \mathbf{A}^T$, $a_{ij} = a_{ji}$ for all i, j
- positive definite: $\mathbf{z}^T\mathbf{A}\mathbf{z} > 0, \forall \mathbf{z} \neq \mathbf{0}$
- band diagonal: $a_{ij} = 0$ if $|i - j| > \beta$, where β is the bandwidth
- sparse: most of the elements of **A** are zero

Outline

Symmetric positive definite systems

- Cholesky decomposition:

$$\mathbf{A} = \mathbf{L}\mathbf{L}^T$$

  where **L** is lower triangular.
- **A** admits a Cholesky decomposition *if and only if* it is symmetric positive definite
- if the decomposition exists, it is unique

Cholesky decomposition algorithm with overwriting of **A**

Algorithm: Cholesky decomposition algorithm

for $j = 1$ **to** n **do**
  for $k = 1$ **to** $j - 1$ **do**
    for $i = j$ **to** n **do**
      $a_{ij} \leftarrow a_{ij} - a_{ik}\, a_{jk}$;
    end for
  end for
  $a_{jj} \leftarrow \sqrt{a_{jj}}$;
  for $k = j + 1$ **to** n **do**
    $a_{kj} \leftarrow a_{kj} / a_{jj}$;
  end for
end for

Cholesky decomposition - properties

- does not need pivoting to maintain stability
- only $n^3/6$ multiplications and $n^3/6$ additions are required
- for the algorithm presented, only the lower triangle of **A** is modified, and can be restored, if needed, from the upper triangle
- requires about half the computations and half of the memory compared with LU factorization
- there are variations of Cholesky decomposition for banded matrices, for positive semi-definite matrices (semi-Cholesky decomposition) and for symmetric indefinite matrices

Suggestions of methods to use

If **A** is a real dense square matrix...

- ...use LU decomposition with partial pivoting: $\mathbf{A} = \mathbf{PLU}$
- ...and is a band matrix, use LU decomposition with pivoting and row interchanges
- ...and is tridiagonal, use Gaussian elimination
- ...and is symmetric positive definite, use Cholesky decomposition
- ...and is symmetric tridiagonal, use special Cholesky with pivoting, $\mathbf{A} = \mathbf{LDL}^T$
- ...and is symmetric indefinite, use special Cholesky

In MATLAB, check the functions: `chol`, `ichol`, `ldl`, `lu`, `ilu`.

Outline

Polynomial interpolation

- a function $p(x)$ interpolates a set of points $\{(x_i, y_i) | i = 0, \dots, N\}$ if it satisfies $y_i = p(x_i)$ for all $i = 0, \dots, N$.
- this leads to a system of $N + 1$ equations. If $p(x)$ is a polynomial of degree M, $p(x) = a_M x^M + \cdots + a_1 x + a_0$, the system is of the form

$$\begin{aligned} a_0 + a_1 x_0 + \cdots + a_M x_0^M &= y_0 \\ &\cdots \\ a_0 + a_1 x_N + \cdots + a_M x_N^M &= y_N \end{aligned}$$

  where the unknowns are $a_0, \dots, a_M$.
- if $M = N \rightarrow$ *Vandermonde* matrix
- in MATLAB check the functions `polyfit` and `polyval`
- write the MATLAB function to solve the interpolation problem for $M = N$. Do NOT use MATLAB's own functions for interpolation!

1D Poisson problem

A two-point boundary problem,

$$-u''(x) = y(x), \quad x \in [0, 1], \quad u(0) = u(1) = 0,$$

where y is a given continuous function on $[0, 1]$. If y cannot be integrated exactly, approximate solutions are sought. Using finite differences,

$$u'(x) = \lim_{h \to 0} \frac{u(x + \frac{h}{2}) - u(x - \frac{h}{2})}{h}$$

$$u''(x) = \lim_{h \to 0} \frac{u(x + h) - 2u(x) + u(x - h)}{h^2}$$

Divide the interval $[0, 1]$ in $m + 1$ equal subintervals of length $h = 1/(m + 1)$ and let $x_i = ih$ be the limits of these subtintervals, $i = 0, \ldots, m + 1$.

Denote $y(x_i) = y(ih) = y_i$ and $u(x_i) = u(ih) = u_i$. Then, the problem becomes

$$-\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} = y_i, \qquad i = 1, \ldots, m, \; u_0 = u_{m+1} = 0.$$

This can be written as a linear system:

$$\mathbf{Tu} = \begin{bmatrix} 2 & -1 & 0 & & & & \\ -1 & 2 & -1 & & & & \\ 0 & \ddots & \ddots & \ddots & & & \\ & & & & & & 0 \\ & & & & -1 & 2 & -1 \\ & & & & 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{m-1} \\ u_m \end{bmatrix} = h^2 \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{m-1} \\ y_m \end{bmatrix}$$

where the matrix $\mathbf{T}$ is a *Toeplitz matrix*. The system can be solved using the Levinson algorithm - see `levinson` function in MATLAB.