Scientific Computing Software. Matlab

Lecture 7. Linear Algebra and Matrix Computations

Solving linear systems and eigenvalue problems. Condition number and perturbations

Nasedkina A. A.



Matrices and matrix algorithms in modern world

- Matrix algorithms pervade most areas of science and engineering
- Matrix algorithms are used by most disciplines requiring numerical computing
- Computations involving matrices is the topic becoming increasingly important in Computer Science
- Matrix algorithms in computer science are used for data mining, information retrieval, search engines, pattern recognition, graphics, ...

Example of a linear system

- **Problem**. A set of 12 coins containing nickels (5c each), dimes (10c each) and quarters (25c each) totals to \$1.45. In addition, the total without the nickels amounts to \$1.25. How many of each coin are there?
- **Solution.** Let x_n be the number of nickels, x_d be the number of dimes, x_q be the number of quarters

- **Equation form**

1^{st} equation: $x_n + x_d + x_q = 12$

2^{nd} equation: $5x_n + 10x_d + 25x_q = 145$

3^{rd} equation: $0x_n + 10x_d + 25x_q = 125$

What's the answer?



- **Matrix form**

$$\begin{pmatrix} 1 & 1 & 1 \\ 5 & 10 & 25 \\ 0 & 10 & 25 \end{pmatrix} \begin{pmatrix} x_n \\ x_d \\ x_q \end{pmatrix} = \begin{pmatrix} 12 \\ 145 \\ 125 \end{pmatrix}$$

Matrix $A = \begin{pmatrix} 1 & 1 & 1 \\ 5 & 10 & 25 \\ 0 & 10 & 25 \end{pmatrix}$

Right-hand side vector $b = \begin{pmatrix} 12 \\ 145 \\ 125 \end{pmatrix}$

Vector of unknowns $x = \begin{pmatrix} x_n \\ x_d \\ x_q \end{pmatrix}$

Matrix equation: $Ax = b$

Solve linear system in Matlab

- Matrix form:

$$\begin{pmatrix} 1 & 1 & 1 \\ 5 & 10 & 25 \\ 0 & 10 & 25 \end{pmatrix} \begin{pmatrix} x_n \\ x_d \\ x_q \end{pmatrix} = \begin{pmatrix} 12 \\ 145 \\ 125 \end{pmatrix}$$

- **Answer**: 4 nickels, 5 dimes and 3 quarters

- Question: is the order of equations important or not? Will the solution change?

Matrix left division **mldivide, \:** **A\B ~ inv(A)*B** solves the system of linear equations A*X = B for X

```
A=[1 1 1; 5 10 25; 0 10 25]
```

```
A = 3×3
     1     1     1
     5    10    25
     0    10    25
```

```
b=[12; 145; 125]
```

```
b = 3×1
    12
   145
   125
```

```
x=A\b
```

```
x = 3×1
     4
     5
     3
```

Ways to solve linear system in Matlab

Consider linear system Ax=b

1. Backslash: x=A\b or x=mldivide(A,b)
2. **linsolve** command: x=linsolve(A,b)
3. Use matrix inverse: x=inv(A)*b. This method should be used with caution as it is slower, less accurate, and can silently produce garbage for ill-conditioned matrices.
4. Use factorizations of A (LU, QR, Cholesky)

Backslash operator

- **The primary numerical solver in MATLAB.** For $Ax = b$, **x = A\b** automatically selects the best algorithm based on the structure of A

Matrix Structure	Algorithm Used
Triangular	Direct substitution
Symmetric positive definite	Cholesky
Square, general	LU with pivoting
Rectangular	QR (least squares)

```
A = [3 -2 5; -4.5 2 3; 5 1 -2.5]
```

```
A = 3×3
    3.0000   -2.0000    5.0000
   -4.5000    2.0000    3.0000
    5.0000    1.0000   -2.5000
```

```
b = [7.5; 5.5; 4.5]
```

```
b = 3×1
    7.5000
    5.5000
    4.5000
```

```
x = A \ b % Automatically chooses LU
```

```
x = 3×1
    1.2692
    2.8113
    1.8630
```

```
residual = norm(A*x - b)
```

```
residual = 1.9860e-15
```

Explicit Solver with Options: linsolve

- linsolve is similar to backslash but allows you to **declare the matrix structure** to skip checks and choose faster algorithms .
- **When to use:** when you know the structure and want to skip MATLAB's internal checks (marginal performance gain), or when solving **many systems** with the same structure.

```
A = [4 -2 1; -2 4 -2; 1 -2 4]
```

```
A = 3×3
     4    -2     1
    -2     4    -2
     1    -2     4
```

```
b = [11; -16; 17]
```

```
b = 3×1
    11
   -16
    17
```

```
% Default: same as backslash
x1 = linsolve(A, b)
```

```
x1 = 3×1
     1
    -2
     3
```

```
% Declare symmetric positive definite
% → uses Cholesky
opts.SYM = true;
opts.POSDEF = true;
x2 = linsolve(A, b, opts)
```

```
x2 = 3×1
    1.0000
   -2.0000
    3.0000
```

Matrix Factorizations

- LU
- QR
- Cholesky

Matrix Factorizations

- Solving $Ax = b$ by computing A^{-1} is **numerically unstable and inefficient.** Factorizations decompose Ainto structured pieces (triangular, orthogonal) that make solving **fast and stable.**

Factorization	Decomposition	Matrix Requirements
LU	$A = LU$ (with pivoting: $PA = LU$)	Square, general
QR	$A = QR$	Any rectangular matrix
Cholesky	$A = LL^T$ (or $R^T R$)	Symmetric positive definite

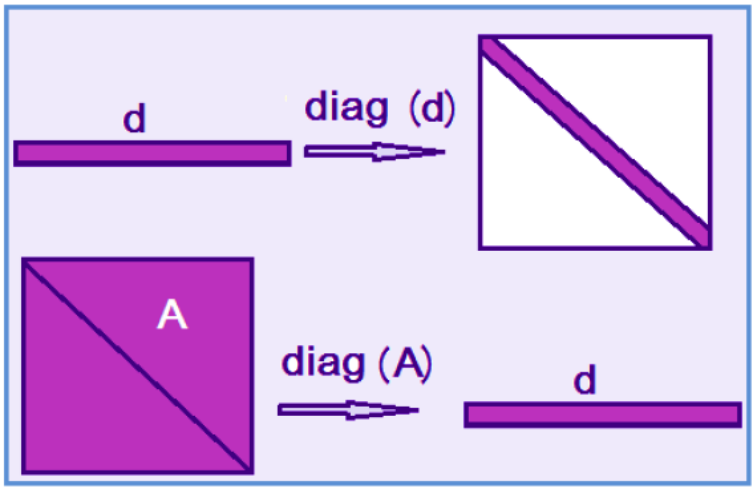
- MATLAB provides all three as built-in functions: **lu**, **qr**, and **chol**

Structures of square matrices with respect to location of zero entries: diagonal, upper and lower triangular

- Diagonal matrix $a_{ij} = 0 \quad \text{for } \forall j \neq i$

$\forall$ stands for "for every" or "for any"

Notation: $A = diag(a_{11}, a_{22}, ..., a_{nn})$



- Upper-triangular matrix

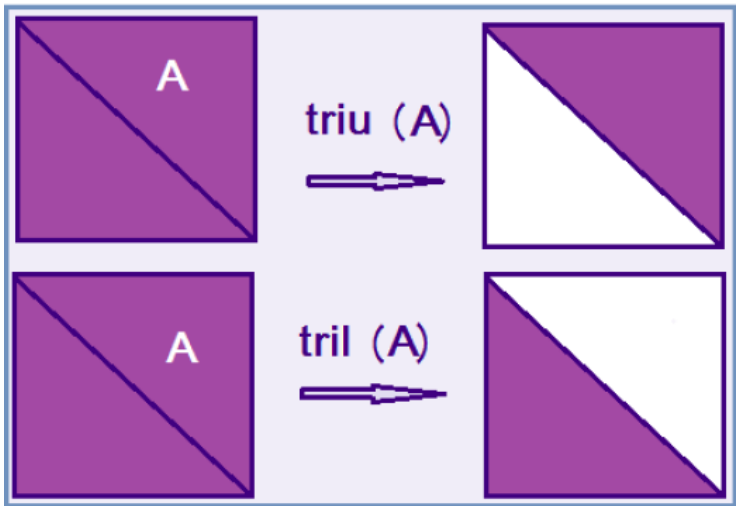
$a_{ij} = 0, \quad \text{for } \forall i > j$

$$\begin{matrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times \end{matrix}$$

- Lower-triangular matrix

$a_{ij} = 0, \quad \text{for } \forall i < j$

$$\begin{matrix} \times & 0 & 0 & 0 \\ \times & \times & 0 & 0 \\ \times & \times & \times & 0 \\ \times & \times & \times & \times \end{matrix}$$



Diagonal, upper and lower triangular matrices

```
A=[1 2 3; 4 5 6; 7 8 9]
diagA=diag(A)
v=[10 20 30]
D=diag(v)
U=triu(A)
L=tril(A)
```

```
A = 3×3
     1     2     3
     4     5     6
     7     8     9

diagA = 3×1
     1
     5
     9

v = 1×3
    10    20    30

D = 3×3
    10     0     0
     0    20     0
     0     0    30

U = 3×3
     1     2     3
     0     5     6
     0     0     9

L = 3×3
     1     0     0
     4     5     0
     7     8     9
```

LU Factorization

- **Concept:** Factor A into a lower triangular L and upper triangular U. With partial pivoting, $PA = LU$ where P is a permutation matrix.
- **Why useful:** Once factored, solving $Ax = b$ becomes two triangular solves: $Ly = Pb$, then $Ux = y$.

Use LU-factorization to solve the system $Ax = b$

If $A = LU \Rightarrow LUx = b$.

Denote $Ux = y \Rightarrow Ly = b$. We get two systems:

1) System $Ly = b$ with lower-triangular matrix

(can be solved using forward substitution)

$Ly = b \Rightarrow$ formal solution $y = L^{-1}b$

2) System $Ux = y$ with upper-triangular matrix

(can be solved using back substitution)

$Ux = y \Rightarrow$ formal solution $x = U^{-1}y$

Hence $x = U^{-1}L^{-1}b$ is the solution of initial system $Ax = b$

Matlab:

- [L,U]=lu(A)
- x=U\(L\b)

LU-factorization in Matlab

```
n=3;
A=rand(n)
[L,U]=lu(A) % computes upper triangular matrix U
% and a permuted lower triangular matrix L such that
% A = L*U.




L*U % gives A


isequal(A,L*U) % sometimes gives logical 1

norm(A-L*U)

[L1,U1,P]=lu(A) % computes permutation matrix P,
% unit lower triangular L1 and upper triangular U1,
% such that A = P'*L1*U1.
```

```
A = 3×3
    0.1890    0.3685    0.0811
    0.6868    0.6256    0.9294
    0.1835    0.7802    0.7757

L = 3×3
    0.2751    0.3203    1.0000
    1.0000         0         0
    0.2672    1.0000         0

U = 3×3
    0.6868    0.6256    0.9294
         0    0.6131    0.5274
         0         0   -0.3435

ans = 3×3
    0.1890    0.3685    0.0811
    0.6868    0.6256    0.9294
    0.1835    0.7802    0.7757


ans = logical
   0

ans = 2.7756e-17
L1 = 3×3
    1.0000         0         0
    0.2672    1.0000         0
    0.2751    0.3203    1.0000

U1 = 3×3
    0.6868    0.6256    0.9294
         0    0.6131    0.5274
         0         0   -0.3435

P = 3×3
     0     1     0
     0     0     1
     1     0     0
```

LU-factorization in Matlab

```
% A = P'*L1*U1
A-P'*L1*U1
norm(A-P'*L1*U1)

% L1 is P*L

PL=P*L

L1=L1
L1-P*L
```

```
ans = 3×3
10^-16 ×
         0         0         0
         0         0         0
   -0.2776         0         0

ans = 2.7756e-17
PL = 3×3
    1.0000         0         0
    0.2672    1.0000         0
    0.2751    0.3203    1.0000

L1 = 3×3
    1.0000         0         0
    0.2672    1.0000         0
    0.2751    0.3203    1.0000

ans = 3×3
     0     0     0
     0     0     0
     0     0     0
```

Use LU to solve linear system

```
A=[9 10 3;10 7 6;2 1 10]
```

```
A = 3×3
     9    10     3
    10     7     6
     2     1    10
```

```
b = [11; -16; 17]
```

```
b = 3×1
    11
   -16
    17
```

```
[L,U]=lu(A)
```

```
L = 3×3
    0.9000    1.0000         0
    1.0000         0         0
    0.2000   -0.1081    1.0000

U = 3×3
   10.0000    7.0000    6.0000
         0    3.7000   -2.4000
         0         0    8.5405
```

```
% Forward substitution
y = L \ b
```

```
y = 3×1
  -16.0000
   25.4000
   22.9459
```

```
% Back substitution
x = U \ y
```

```
x = 3×1
   -9.2373
    8.6076
    2.6867
```

```
%check Ax=b
A*x
```

```
ans = 3×1
   11.0000
  -16.0000
   17.0000
```

```
%check Ax=b
isequal(A*x,b)
```

```
ans = logical
   0
```

```
b-A*x
```

```
ans = 3×1
10^-13 ×
   -0.1066
    0.0711
         0
```

```
norm(b-A*x)
```

```
ans = 1.2809e-14
```

Use PLU to solve linear system

```
A=[9 10 3;10 7 6;2 1 10]
```

```
A = 3×3
     9    10     3
    10     7     6
     2     1    10
```

```
b = [11; -16; 17]
```

```
b = 3×1
    11
   -16
    17
```

```
[L, U, P] = lu(A)
```

```
L = 3×3
    1.0000         0         0
    0.9000    1.0000         0
    0.2000   -0.1081    1.0000

U = 3×3
   10.0000    7.0000    6.0000
         0    3.7000   -2.4000
         0         0    8.5405

P = 3×3
     0     1     0
     1     0     0
     0     0     1
```

```
% Forward substitution
y = L \ (P*b)
```

```
y = 3×1
   -16.0000
    25.4000
    22.9459
```

```
% Back substitution
x = U \ y
```

```
x = 3×1
    -9.2373
     8.6076
     2.6867
```

```
%check Ax=b
A*x
```

```
ans = 3×1
    11.0000
   -16.0000
    17.0000
```

```
b-A*x
```

```
ans = 3×1
10^-13 ×
   -0.1066
    0.0711
         0
```

```
% Compare with backslash
x_direct = A \ b
```

```
x_direct = 3×1
    -9.2373
     8.6076
     2.6867
```

```
norm(x - x_direct)
```

```
ans = 0
```

QR Factorization

- **Concept:** Factor A into an orthogonal matrix Q and upper triangular R: $A = QR$.

  Why useful:
- Works for **rectangular** matrices (unlike LU or Cholesky)
- Numerically stable for least-squares problems
- For overdetermined systems, A\b internally uses QR to compute the least-squares solution

If $Ax = b \Rightarrow x = A^{-1}b$ (this is formal writing, as we shall never compute the inverse A^{-1} in numerical algorithms).

If $A = QR \Rightarrow x = (QR)^{-1}b = R^{-1}Q^{-1}b = R^{-1}Q^{T}b$

($Q^{-1} = Q^{T}$, as Q is orthogonal).

Hence $x = R^{-1}Q^{T}b$ is the solution of $Rx = Q^{T}b$

R is upper triangular matrix, so this system is faster to solve.

Matlab:

- [Q,R]=qr(A)
- x=R\(Q'*b)

QR-factorization in Matlab

QR-factorization for rectangular matrix n>m

```
n=5;m=3;
A=rand(n,m)


[Qf,Rf]=qr(A) % full qr




[Qt,Rt]=qr(A,0) % thin qr




Qf*Rf % gives A
```

```
A = 5×3
    0.5285    0.6892    0.9133
    0.1656    0.7482    0.1524
    0.6020    0.4505    0.8258
    0.2630    0.0838    0.5383
    0.6541    0.2290    0.9961

Qf = 5×5
   -0.4894   -0.3591    0.5737   -0.5021   -0.2240
   -0.1534   -0.8278   -0.2394    0.2979    0.3810
   -0.5575    0.0406   -0.4975    0.2243   -0.6243
   -0.2435    0.1696    0.5783    0.7589    0.0403
   -0.6057    0.3943   -0.1776   -0.1813    0.6429

Rf = 5×3
   -1.0799   -0.8623   -1.6652
         0   -0.7440    0.0635
         0         0    0.2111
         0         0         0
         0         0         0

Qt = 5×3
   -0.4894   -0.3591    0.5737
   -0.1534   -0.8278   -0.2394
   -0.5575    0.0406   -0.4975
   -0.2435    0.1696    0.5783
   -0.6057    0.3943   -0.1776

Rt = 3×3
   -1.0799   -0.8623   -1.6652
         0   -0.7440    0.0635
         0         0    0.2111

ans = 5×3
    0.5285    0.6892    0.9133
    0.1656    0.7482    0.1524
    0.6020    0.4505    0.8258
    0.2630    0.0838    0.5383
    0.6541    0.2290    0.9961
```

```
isequal(A,Qf*Rf)
norm(A-Qf*Rf)
norm(A-Qt*Rt)
```

```
ans = logical
   0

ans = 3.6653e-16

ans = 3.6653e-16
```

QR-factorization in Matlab

QR-factorization for square matrix n=m

```
n=4;
A=rand(n)


[Qf,Rf]=qr(A) % full qr




[Qt,Rt]=qr(A,0) % thin qr
```

```
A = 4x4
    0.1419    0.9595    0.9340    0.3922
    0.4218    0.6557    0.6787    0.6555
    0.9157    0.0357    0.7577    0.1712
    0.7922    0.8491    0.7431    0.7060

Qf = 4x4
   -0.1100   -0.7517    0.5635    0.3246
   -0.3269   -0.3239    0.0155   -0.8877
   -0.7099    0.5041    0.4844    0.0860
   -0.6141   -0.2756   -0.6691    0.3150

Rf = 4x4
   -1.2900   -0.8667   -1.3189   -0.8125
         0   -1.1497   -0.7448   -0.6154
         0         0    0.4066   -0.1583
         0         0         0   -0.2174

Qt = 4x4
   -0.1100   -0.7517    0.5635    0.3246
   -0.3269   -0.3239    0.0155   -0.8877
   -0.7099    0.5041    0.4844    0.0860
   -0.6141   -0.2756   -0.6691    0.3150

Rt = 4x4
   -1.2900   -0.8667   -1.3189   -0.8125
         0   -1.1497   -0.7448   -0.6154
         0         0    0.4066   -0.1583
         0         0         0   -0.2174
```

QR-factorization in Matlab

QR-factorization for rectangular matrix n<m

```
n=3;m=4;
A=rand(n,m)


[Qf,Rf]=qr(A) % full qr




[Qt,Rt]=qr(A,0) % thin qr
```

```
A = 3x4
    0.0318    0.0971    0.3171    0.4387
    0.2769    0.8235    0.9502    0.3816
    0.0462    0.6948    0.0344    0.7655

Qf = 3x3
   -0.1127   -0.0142   -0.9935
   -0.9801   -0.1629    0.1135
   -0.1634    0.9865    0.0044

Rf = 3x4
   -0.2825   -0.9316   -0.9727   -0.5485
         0    0.5500   -0.1253    0.6868
         0         0   -0.2071   -0.3892

Qt = 3x3
   -0.1127   -0.0142   -0.9935
   -0.9801   -0.1629    0.1135
   -0.1634    0.9865    0.0044

Rt = 3x4
   -0.2825   -0.9316   -0.9727   -0.5485
         0    0.5500   -0.1253    0.6868
         0         0   -0.2071   -0.3892
```

Use QR to solve linear system

```
A=[9 10 3;10 7 6;2 1 10]
```

```
A = 3×3
     9    10     3
    10     7     6
     2     1    10
```

```
b = [11; -16; 17]
```

```
b = 3×1
    11
   -16
    17
```

```
[Q,R]=qr(A)
```

```
Q = 3×3
   -0.6617    0.7427    0.1031
   -0.7352   -0.6157   -0.2835
   -0.1470   -0.2633    0.9534

R = 3×3
  -13.6015  -11.9105   -7.8668
         0    2.8532   -4.0998
         0         0    8.1428
```

```
x1=R\(Q\b)
```

```
x1 = 3×1
   -9.2373
    8.6076
    2.6867
```

```
x2=R\(Q'*b)
```

```
x2 = 3×1
   -9.2373
    8.6076
    2.6867
```

```
b-A*x1
```

```
ans = 3×1
10^-13 ×
   -0.3908
   -0.2132
   -0.0711
```

```
norm(b-A*x1)
```

```
ans = 4.5079e-14
```

```
b-A*x2
```

```
ans = 3×1
10^-13 ×
   -0.3730
   -0.1776
         0
```

```
norm(b-A*x2)
```

```
ans = 4.1317e-14
```

Cholesky factorization

- **Concept:** For **symmetric positive definite** (SPD) matrices, $A = LL^T$ where L is lower triangular. This is twice as efficient as LU and guarantees stability without pivoting.
- **Checking SPD:** All eigenvalues positive, or equivalently chol succeeds.
- Cholesky is the standard choice for SPD systems arising from finite elements, covariance matrices, and normal equations.

When A is symmetric, i.e. $A = A^T$, we can try to compute U as L^T to preserve symmetry: $A = LL^T$ or $A = U^T U$

Matlab:

- U=chol(A)
- x=U\(U'\b)

Cholesky factorization in Matlab

```
n=3;
A=rand(n);
A=A*A'
R=chol(A) % for symmetric positive definite matrix
% computes upper triangular R, such that A=R'*R
A-R'*R
norm(A-R'*R)
```

```
L=chol(A,'lower') % for symmetric positive definite matrix
% computes lower triangular such that A=L*L'
A-L*L'
norm(A-L*L')
```

```
[R,flag] = chol(A) % also returns the output flag
% indicating whether A is symmetric positive definite.
% For symmetric positive definite flag = 0,
% otherwise flag is positive.
```

```
A = 3×3
    0.7209    0.8453    0.6216
    0.8453    1.3159    0.7771
    0.6216    0.7771    0.5943

R = 3×3
    0.8490    0.9956    0.7321
         0    0.5698    0.0846
         0         0    0.2261

ans = 3×3
10^-15 ×
         0   -0.1110         0
   -0.1110         0         0
         0         0         0

ans = 1.1102e-16
L = 3×3
    0.8490         0         0
    0.9956    0.5698         0
    0.7321    0.0846    0.2261

ans = 3×3
10^-15 ×
         0   -0.1110         0
   -0.1110         0         0
         0         0         0

ans = 1.1102e-16

R = 3×3
    0.8490    0.9956    0.7321
         0    0.5698    0.0846
         0         0    0.2261

flag = 0
```

Use Cholesky to solve linear system

```
A = randi(10,3)
```

```
A = 3×3
     7     1     8
     4     5     8
    10     4     2
```

```
A=A*A'
```

```
A = 3×3
   114    97    90
    97   105    76
    90    76   120
```

```
issymmetric(A)
```

```
ans = logical
   1
```

```
b = [11; -16; 17]
```

```
b = 3×1
    11
   -16
    17
```

```
eig(A) % positive eigenvalues
```

```
ans = 3×1
   11.0495
   39.2893
  288.6611
```

```
U=chol(A)
```

```
U = 3×3
   10.6771    9.0849    8.4293
         0    4.7397   -0.1221
         0         0    6.9952
```

```
x=U\(U'\b)
```

```
x = 3×1
    0.9300
   -1.1248
    0.1566
```

```
L=chol(A,"lower")
```

```
L = 3×3
   10.6771         0         0
    9.0849    4.7397         0
    8.4293   -0.1221    6.9952
```

```
x=L'\(L\b)
```

```
x = 3×1
    0.9300
   -1.1248
    0.1566
```

Existence of solution for a linear system

Consider a linear system $Ax = b$ for a square matrix A of the size $n \times n$

b of the size $n \times 1$ is the right-hand side vector

x of the size $n \times 1$ is the vector of unknowns

$r = b - Ax$ is called residual vector

Usually we need the norm of this vector (residual norm $\|r\|_2$ for estimation.

When $r = 0,\ x$ is exact solution

When $r \approx 0,\ x$ is approximate solution

- The system is called consistent, if it has a least one solution.
- The system is called inconsistent, if it has no solutions.

There are three cases of the solution existence:

1) unique solution

2) many solutions

3) no solutions

Existence of solution: unique solution

1) unique solution

when A is nonsingular (exists A^{-1}, $\det(A) \neq 0$)

Exists unique $x = A^{-1}b$ is the solution

Example of consistent system with unique solution.

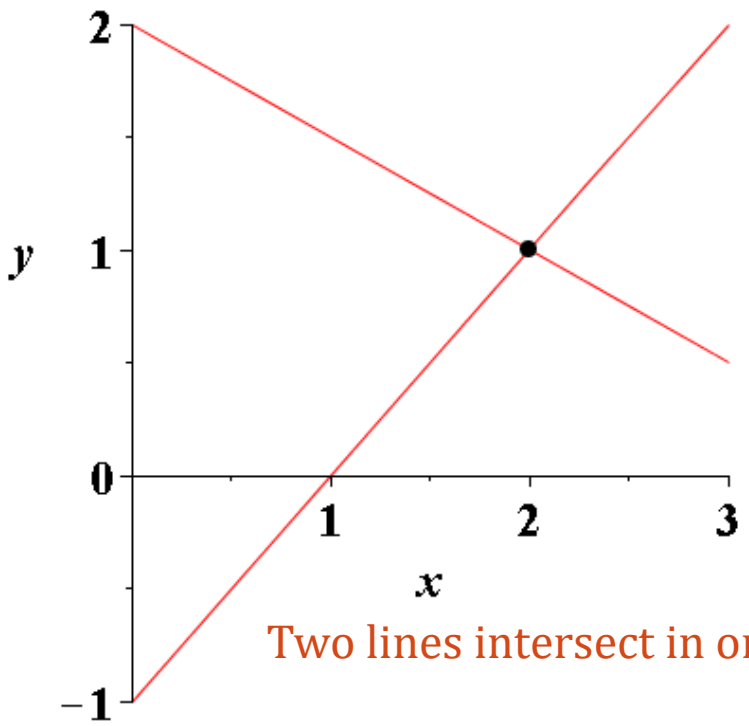
$$\begin{cases} x - y = 1 \\ x + 2y = 4 \end{cases}$$

$x = 2;\ y = 1$ is the solution

Matrix form:

$$\mathbf{A} = \begin{pmatrix} 1 & -1 \\ 1 & 2 \end{pmatrix},\ \mathbf{b} = \begin{pmatrix} 1 \\ 4 \end{pmatrix}$$

$$\mathbf{x} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$$ is the solution



Two lines intersect in one point

Existence of solution: many solutions

2) infinitely many solutions

when A is singular ($\det(A) = 0$) and $b \in Ran(A) = \{Au \mid u \in \mathbb{R}^n\}$

Exists x_0 : $Ax_0 = b \Rightarrow x_0 + v$ is also the solution for any $v \in Null(A) = \{u \in \mathbb{R}^n \mid Au = 0\}$

Example of consistent system with many solutions.

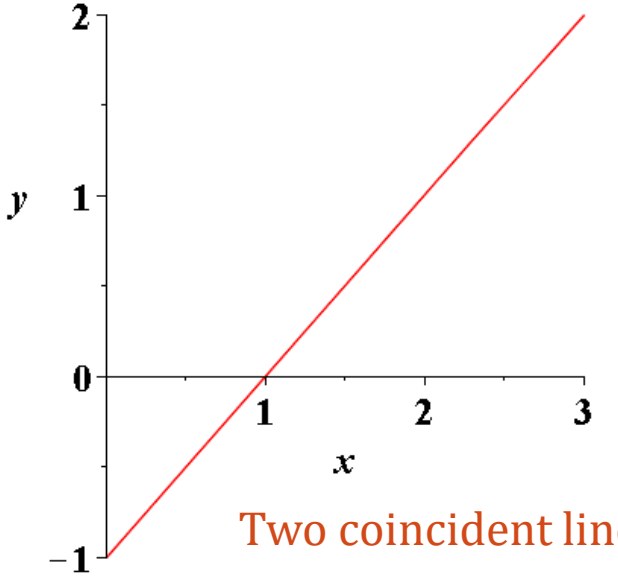
$$\begin{cases} x - y = 1 \\ -2x + 2y = -2 \end{cases}$$

$y = x - 1$ is the solution

Matrix form:

$$\mathbf{A} = \begin{pmatrix} 1 & -1 \\ -2 & 2 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 1 \\ -2 \end{pmatrix}$$

$$\mathbf{x} = \begin{pmatrix} x \\ x - 1 \end{pmatrix}$$ is the solution



Two coincident lines

Existence of solution: no solutions

3) no solutions (inconsistent system)

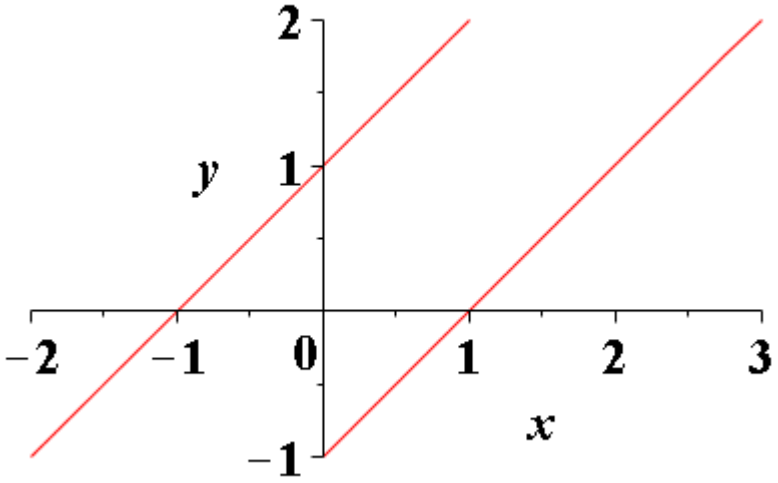
when A is singular ($\det(A) = 0$) and $b \notin Ran(A) = \{Au \mid u \in \mathbb{R}^n\}$

Example of inconsistent system (no solutions).

$$\begin{cases} x - y = 1 \\ x - y = -1 \end{cases}$$

Matrix form:

$$\mathbf{A} = \begin{pmatrix} 1 & -1 \\ 1 & -1 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$



Two parallel lines

Symbolic Solutions

- The Symbolic Math Toolbox provides **exact** solutions (rational numbers, symbolic expressions), useful for deriving formulas, checking numerical results, and handling ill-conditioned or rank-deficient systems
- **Symbolic backslash:** A\b also works for symbolic matrices but **does not compute least-squares solutions** and warns when the system is rank-deficient or inconsistent .