Gaussian elimination

Jordi Cortadella
Department of Computer Science

System of Linear Equations

$$
\begin{array}{rcrcrcr}
2x & + & y & - & z & = & 8 \\
-3x & - & y & + & 2z & = & -11 \\
-2x & + & y & + & 2z & = & -3
\end{array}
$$

$$\Downarrow$$

$$
\begin{bmatrix} 2 & 1 & -1 \\ -3 & -1 & 2 \\ -2 & 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 8 \\ -11 \\ -3 \end{bmatrix}
$$

System of Linear Equations

$$\begin{bmatrix} 2 & 1 & -1 \\ -3 & -1 & 2 \\ -2 & 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 8 \\ -11 \\ -3 \end{bmatrix}$$

$$\Downarrow$$ Gaussian elimination

$$\begin{bmatrix} -3 & -1 & 2 \\ 0 & 5/3 & 2/3 \\ 0 & 0 & 1/5 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} -11 \\ 13/3 \\ -1/5 \end{bmatrix}$$

$$\Downarrow$$ Back-substitution

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \\ -1 \end{bmatrix}$$

Gaussian elimination

- An essential algorithm in Linear Algebra with multiple applications:
  - Solving linear systems of equations
  - Finding the inverse of a matrix
  - Computing determinants
  - Computing ranks and bases of vector spaces

- Named after Carl Friedrich Gauss (1777-1855), but known much before by the Chinese.

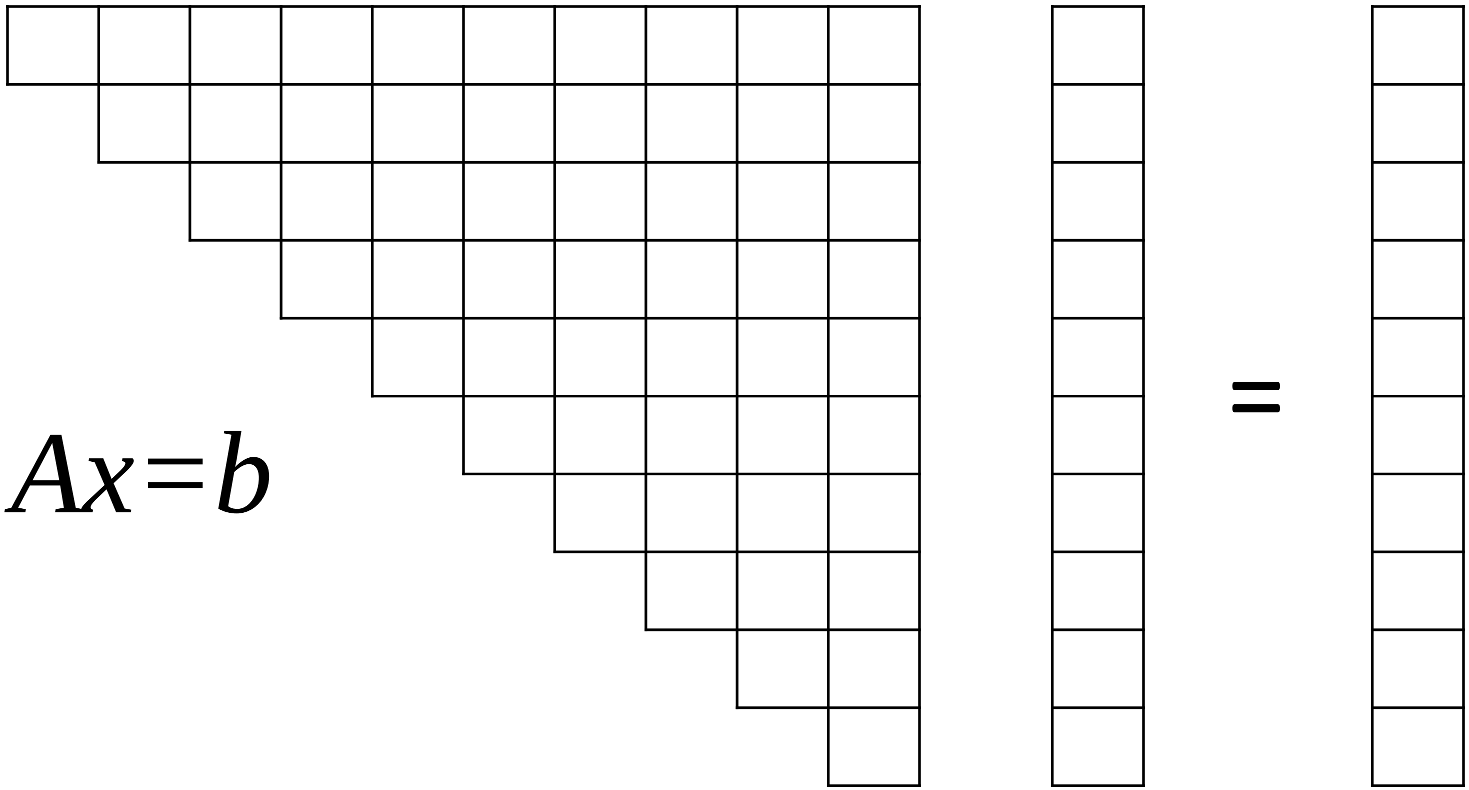
- Alan Turing contributed to provide modern numerical algorithms for computers (characterization of ill-conditioned systems).

System of Linear Equations

```
// Type definitions for real vectors and matrices
typedef vector<double> rvector;
typedef vector<rvector> rmatrix;

// Pre:  A is an n×n matrix, b is an n-element vector.
// Returns x such that Ax=b. In case A is singular,
// it returns a zero-sized vector.
// Post: A and b are modified.

rvector SystemEquations(rmatrix& A, rvector& b) {
    bool invertible = GaussElimination(A, b);
    if (not invertible) return rvector(0);
    // A is in row echelon form
    return BackSubstitution(A, b);
}
```

Back-substitution

$$Ax=b$$

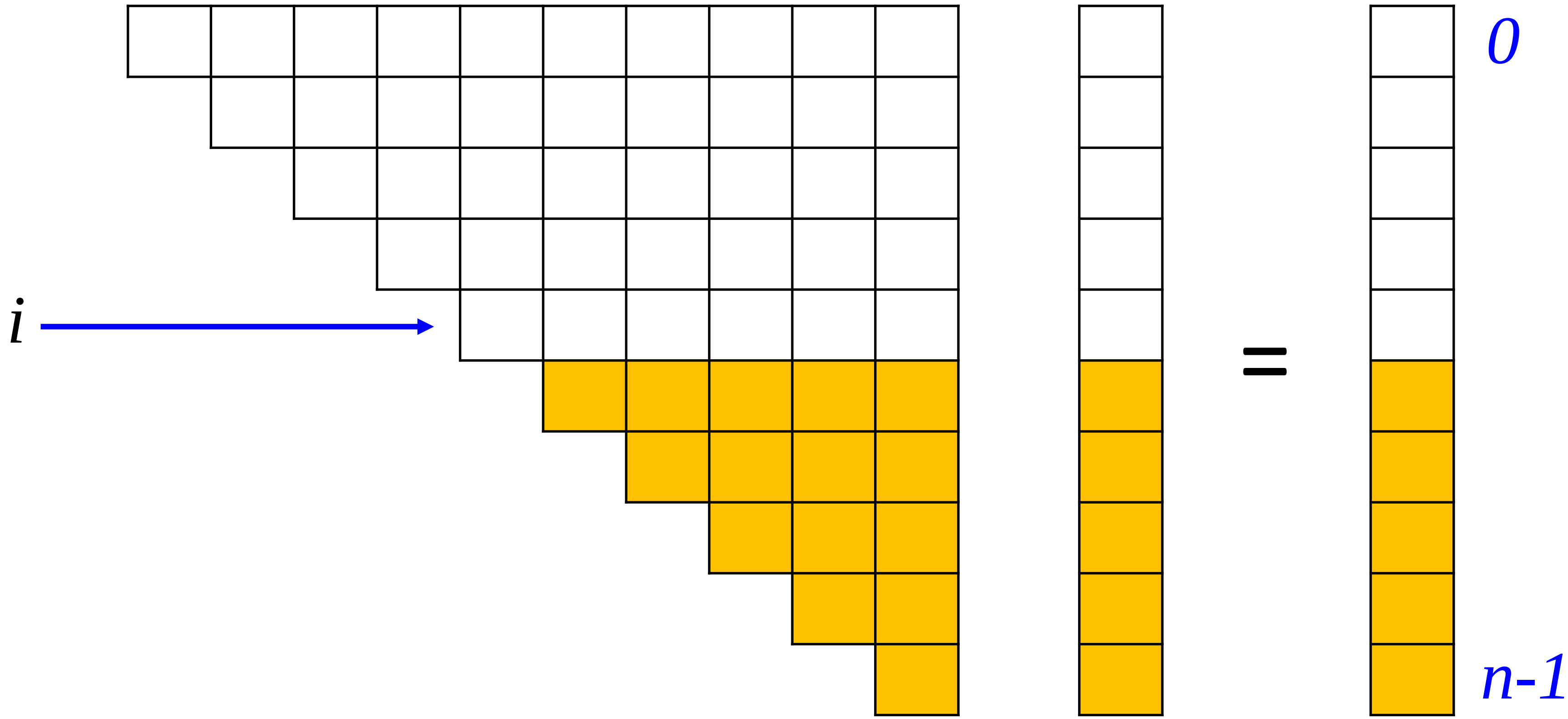


A $\quad$ x $\quad$ b

Assumption: triangular system of equations without zeroes in the diagonal

© Dept. CS, UPC

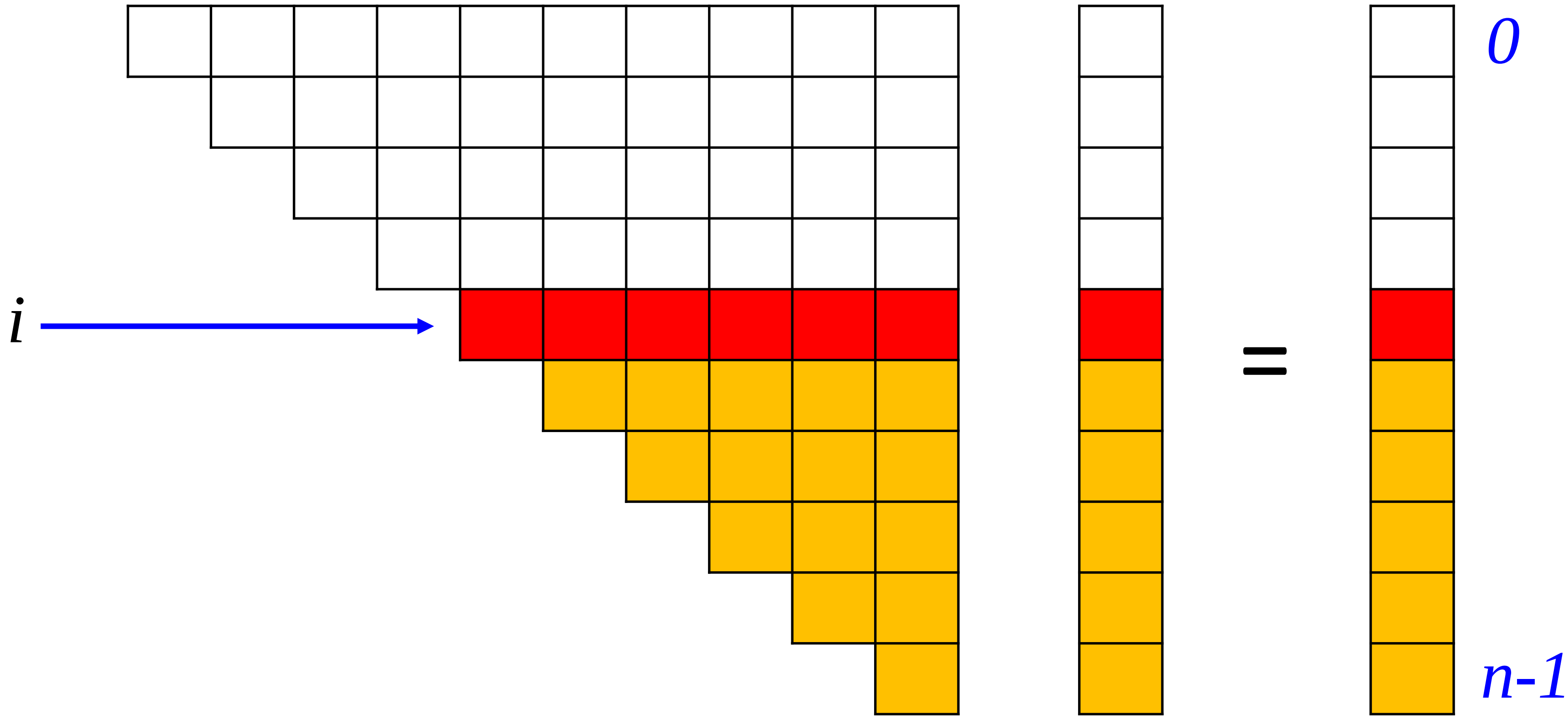
Back-substitution



Invariant: At iteration i, the values for $x_{i+1} \ldots x_{n-1}$ have already been calculated.

© Dept. CS, UPC

Back-substitution



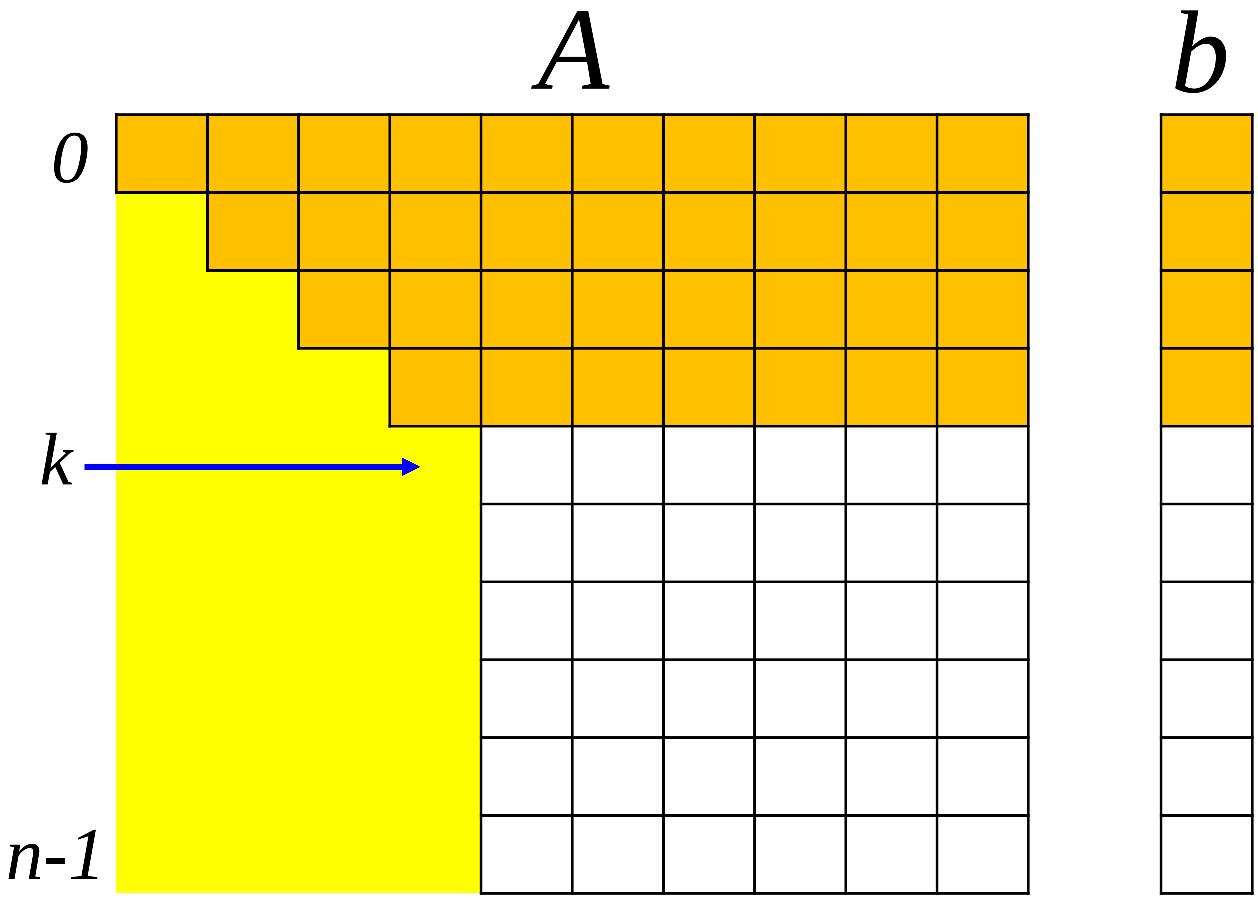
$$A_{i,i}\,x_i + A_{i,i+1}\,x_{i+1} + \cdots + A_{i,n-1}\,x_{n-1} = b_i$$

$$x_i = \frac{b_i - (A_{i,i+1}\,x_{i+1} + \cdots + A_{i,n-1}\,x_{n-1})}{A_{i,i}}$$

Back-substitution

```
// Pre:  A is an invertible n×n matrix in row echelon form,
//       b is a n-element vector
// Returns x such that Ax=b

rvector BackSubstitution(const rmatrix& A, const rvector& b) {
  int n = A.size();
  rvector x(n); // Creates the vector for the solution

  // Calculates x from x[n-1] to x[0]
  for (int i = n - 1; i >= 0; --i) {
    // The values x[i+1..n-1] have already been calculated
    double s = 0;
    for (int j = i + 1; j < n; ++j) s = s + A[i][j]*x[j];
    x[i] = (b[i] – s)/A[i][i];
  }

  return x;
}
```

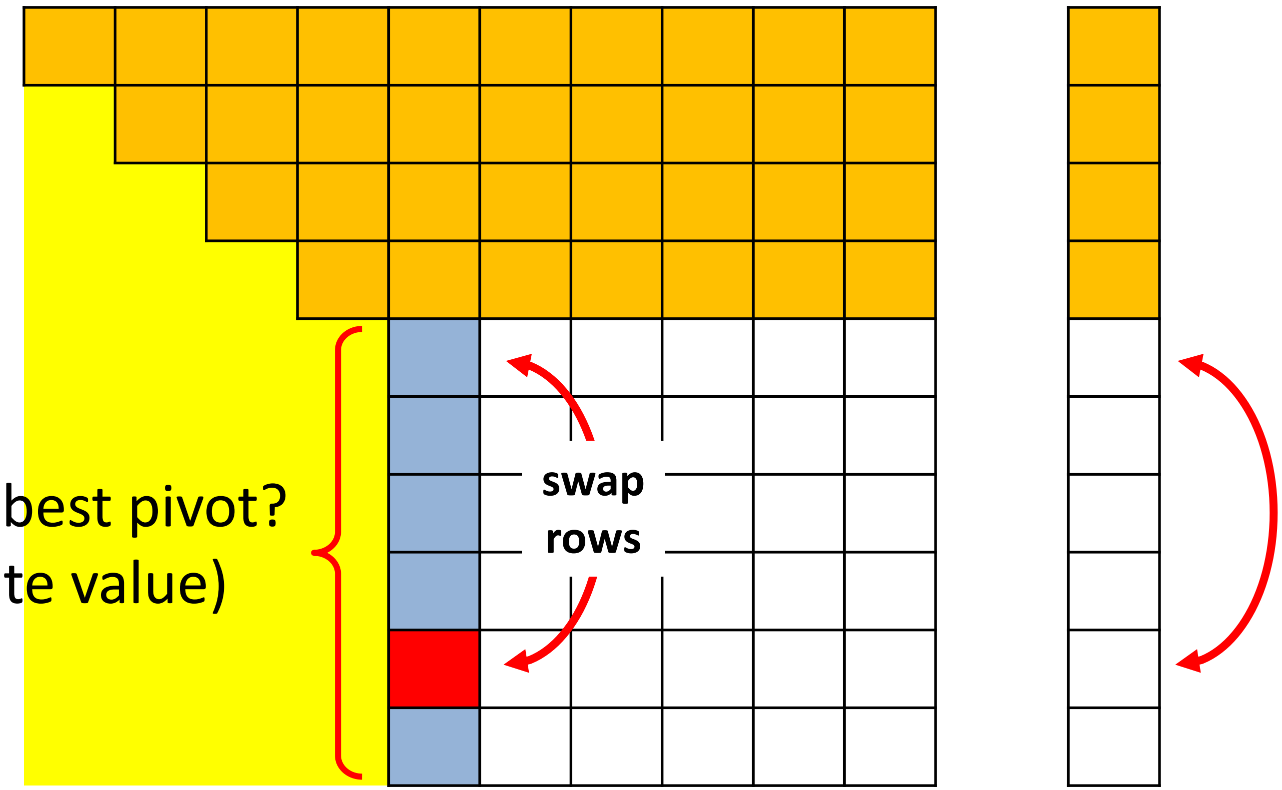
Gaussian elimination



Invariant: The rows 0..k have been reduced
(zeroes at the left of the diagonal)

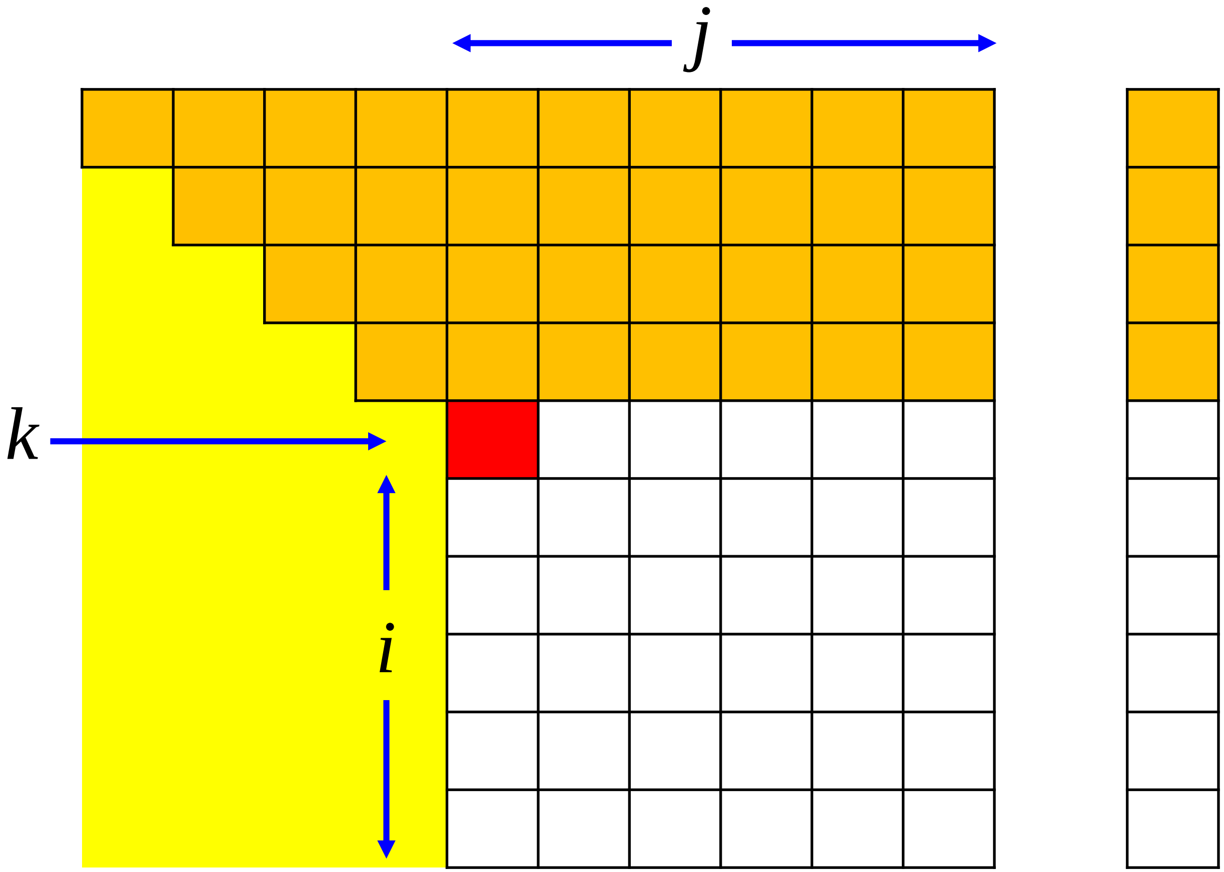
© Dept. CS, UPC

Gaussian elimination



What is the best pivot?
(max absolute value)

Gaussian elimination



For each row *i*, add a multiple of row *k* such that A[i][k] becomes zero.
The coefficient is –A[i][k]/A[k][k].

Gaussian elimination

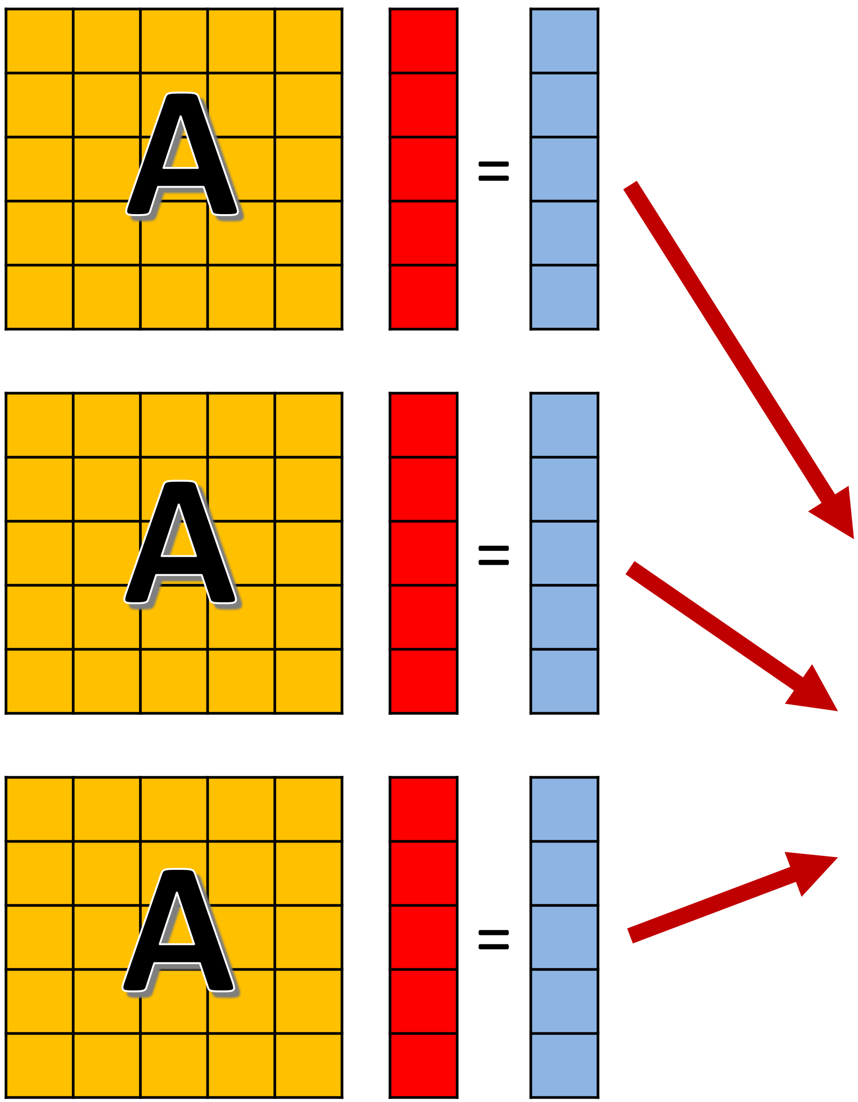
```
// Pre:  A is an n×n matrix, b is a n-element vector.
// Returns true if A is invertible, and false if A is singular.
// Post: If A is invertible, A and b are the result of
//       the Gaussian elimination (A is in row echelon form).

bool GaussianElimination(rmatrix& A, rvector& b) {
  int n = A.size();

  // Reduce rows 1..n-1 (use the pivot in previous row k)
  for (int k = 0; k < n - 1; ++k) {
    // Rows 0..k have already been reduced
    int imax = find_max_pivot(A, k); // finds the max pivot
    if (abs(A[imax][k]) < 1e-10) return false; // Singular matrix

    swap(A[k], A[imax]); swap(b[k], b[imax]); // Swap rows k and imax

    // Force 0's in column A[k+1..n-1][k]
    for (int i = k + 1; i < n; ++i) {
      double c = A[i][k]/A[k][k]; // coefficient to scale row
      A[i][k] = 0;
      for (int j = k + 1; j < n; ++j) A[i][j] = A[i][j] - c*A[k][j];
      b[i] = b[i] - c*b[k];
    }
  }

  return true; // We have an invertible matrix
}
```

Discussion: how large should a pivot be?

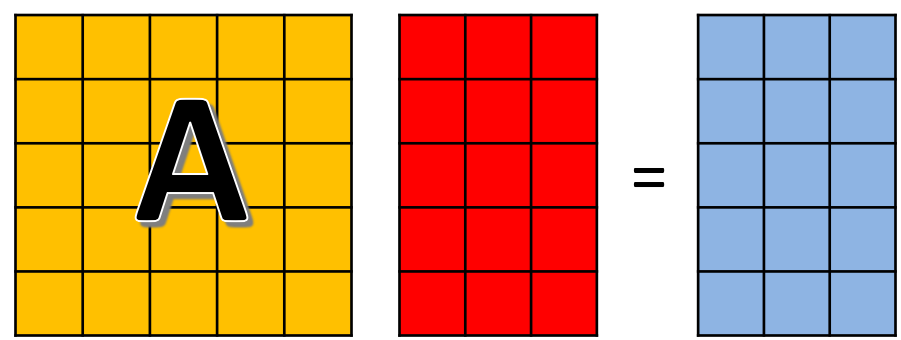
Gaussian elimination

```
// Pre:  A is an n×n matrix, k is the index of a row.
// Returns the index of the row with max absolute value
// for the subcolumn A[k..n-1][k].

int find_max_pivot(const rmatrix& A, int k) {
  int n = A.size();

  double imax = k; // index of the row with max pivot
  double max_pivot = abs(A[k][k]);

  for (int i = k + 1; i < n; ++i) {
    double a = abs(A[i][k]);
    if (a > max_pivot) {
      max_pivot = a;
      imax = i;
    }
  }

  return imax;
}
```

Solving multiple systems of equations



$$
\left.\begin{aligned}
Ax_1 &= b_1 \\
Ax_2 &= b_2 \\
&\vdots \\
Ax_m &= b_m
\end{aligned}\right\} AX = B
$$



Back-substitution

```
// Pre: A is an invertible n×n matrix in row echelon form,
//      B is a n×m matrix
// Returns X such that AX=B

rmatrix BackSubstitution(const rmatrix& A, const rmatrix& B) {
  int n = A.size();
  int m = B[0].size();
  rmatrix X(n, rvector(m)); // Creates the solution matrix

  // Calculates X from X[n-1] to X[0]
  for (int i = n - 1; i >= 0; --i) {
    // The values X[i+1..n-1] have already been calculated
    for (int k = 0; k < m; ++k) {
      double s = 0;
      for (int j = i + 1; j < n; ++j) s = s + A[i][j]*X[j][k];
      X[i][k] = (B[i][k] – s)/A[i][i];
    }
  }

  return X;
}
```

Gaussian elimination

```
// Pre:  A is an n×n matrix, B is an n×m matrix.
// Returns true if A is invertible, and false if A is singular.
// Post: If A is invertible, A and B are the result of
//       the Gaussian elimination (A is in row echelon form).
bool GaussianElimination(rmatrix& A, rmatrix& B) {
  int n = A.size();
  int m = B[0].size();

  // Reduce rows 1..n-1 (use the pivot in previous row k)
  for (int k = 0; k < n - 1; ++k) {
    // Rows 0..k have already been reduced
    int imax = find_max_pivot(A, k); // finds the max pivot
    if (abs(A[imax][k]) < 1e-10) return false; // Singular matrix
    swap(A[k], A[imax]); swap(B[k], B[imax]); // Swap rows k and imax

    // Force 0's in column A[k+1..n-1][k]
    for (int i = k + 1; i < n; ++i) {
      double c = A[i][k]/A[k][k]; // coefficient to scale row
      A[i][k] = 0;
      for (int j = k + 1; j < n; ++j) A[i][j] = A[i][j] – c*A[k][j];
      for (int l = 0; l < m; ++l) B[i][l] = B[i][l] – c*B[k][l];
    }
  }
  return true; // We have an invertible matrix
}
```

Inverse of a Matrix

Reduce the problem to a set of systems of linear equations:

$$
\begin{aligned}
A \cdot X &= I \\
&\Downarrow \\
X &= A^{-1}
\end{aligned}
$$

© Dept. CS, UPC

Computing determinants

- Rules of determinants:
  1. Swapping two rows: determinant multiplied by -1
  2. Multiplying a row by a scalar: the determinant is multiplied by the same scalar
  3. Adding to one row the scalar multiple of another row: determinant does not change

- Algorithm:
  - Do Gaussian elimination and remember the number of row swaps (odd or even).
  - The determinant is the product of the elements in the diagonal (negated in case of an odd number of swaps).
  - Rule 2 is not used, unless some row is scaled.

Summary

- Gaussian elimination is the most used algorithm in Linear Algebra.
- Complexity: $O(n^3)$.
- There are many good packages to solve Linear Algebra operations (LAPACK, LINPACK, Matlab, Mathematica, NumPy, R, …).

© Dept. CS, UPC