Linear algebra

• In scientific computing many tasks eventually take the form of a linear algebra problem.

- Systems of linear equations

- General least squares fitting:

  - $y(x) = \sum_{k=1}^{M} a_k X_k(x)$, minimize $\chi^2 \to (\mathbf{A}^T\mathbf{A})\mathbf{a} = \mathbf{A}^T\mathbf{b}$, $A_{ij} = \dfrac{X_j(x_i)}{\sigma_i}$, $b_i = \dfrac{y_i}{\sigma_i}$

- Solving partial differential equations using the finite element method (FEM)
  - May results in a huge but sparse matrix.

- Determination of eigenvalues and eigenvectors

- Energy eigenvalues in a quantum mechanical system
  - Express wave function in terms of atom-like orbitals $|\psi\rangle = \sum_p a_p |p\rangle$
  - Minimize $\langle\psi|H|\psi\rangle - E\langle\psi|\psi\rangle$ ($H_{pq} = \langle p|H|q\rangle$) with respect to a_p: $\sum_q (H_{pq} - E\delta_{pq}) = 0$. In matrix form $\mathbf{Ha} = E\mathbf{a}$
- Vibration modes in molecules and solids
  - $\mathbf{Ku} = m\omega^2\mathbf{u}$ where $\mathbf{u}$ contains the displacement vectors of all atoms

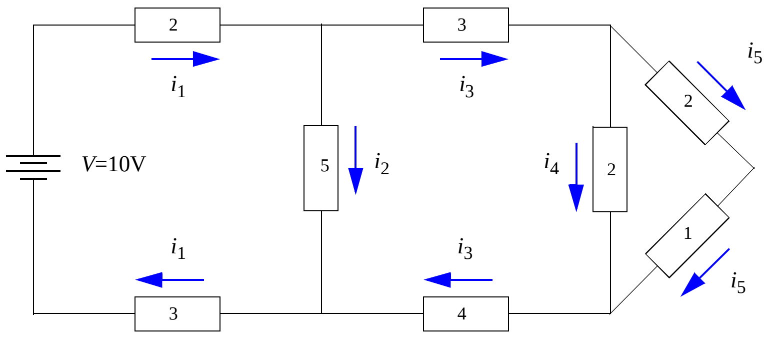
    and $\mathbf{K}$ is the dynamical matrix $K_{ij} = \dfrac{\partial^2 E_{\text{pot}}}{\partial x_i \partial x_j}$
  - All eigenvalues $\omega_i \Rightarrow$ density of states $N(\omega) \Rightarrow$ vibrational free energy F in harmonic approximation

$$F = E_{\text{P}} + k_{\text{B}}T\int_0^\infty N(\omega)\ln\left[2\sinh\left(\frac{h\omega}{4\pi k_{\text{B}}T}\right)\right]d\omega$$

Systems of linear equations

• Systems of linear equations solved in many problems in science and tehcnology

- Example: determining currents in an electrical circuit



- Applying Kirchoff's and Ohm's law in various loops and points in the circuit we get the following equations

$$\begin{cases} 5i_1 + 5i_2 = 10 \\ i_3 - i_4 - i_5 = 0 \\ 2i_4 - 3i_5 = 0 \\ i_1 - i_2 - i_3 = 0 \\ 5i_2 - 7i_3 - 2i_4 = 0 \end{cases}$$

where i_k are the currents in the loops.

Systems of linear equations

- In matrix form this is

$$\begin{bmatrix} 5 & 5 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & -1 \\ 0 & 0 & 0 & 2 & -3 \\ 1 & -1 & -1 & 0 & 0 \\ 0 & 5 & -7 & -2 & 0 \end{bmatrix} \begin{bmatrix} i_1 \\ i_2 \\ i_3 \\ i_4 \\ i_5 \end{bmatrix} = \begin{bmatrix} 10 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$ or $\mathbf{Ri} = \mathbf{v}$, where $\mathbf{R} = \begin{bmatrix} 5 & 5 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & -1 \\ 0 & 0 & 0 & 2 & -3 \\ 1 & -1 & -1 & 0 & 0 \\ 0 & 5 & -7 & -2 & 0 \end{bmatrix}$, $\mathbf{i} = \begin{bmatrix} i_1 \\ i_2 \\ i_3 \\ i_4 \\ i_5 \end{bmatrix}$ and $\mathbf{v} = \begin{bmatrix} 10 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$

- Solution by Matlab

```
>> R=[5 5 0 0 0; 0 0 1 -1 -1; 0 0 0 2 -3; 1 -1 -1 0 0; 0 5 -7 -2 0]
R =
     5     5     0     0     0
     0     0     1    -1    -1
     0     0     0     2    -3
     1    -1    -1     0     0
     0     5    -7    -2     0
>> v=[10 0 0 0 0]'
v =
    10
     0
     0
     0
     0
>> i=R\v
i =
         1.23364485981308
         0.766355140186916
         0.467289719626168
         0.280373831775701
         0.186915887850467
```

Check:
```
>> R*i
ans =
                           10
        -2.77555756156289e-17
                            0
         2.22044604925031e-16
         1.11022302462516e-16
```

Systems of linear equations

- Another simple example: cut-off of Lennard-Jones potential in molecular dynamics simulations

$$V_{LJ}(r) = 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right]$$

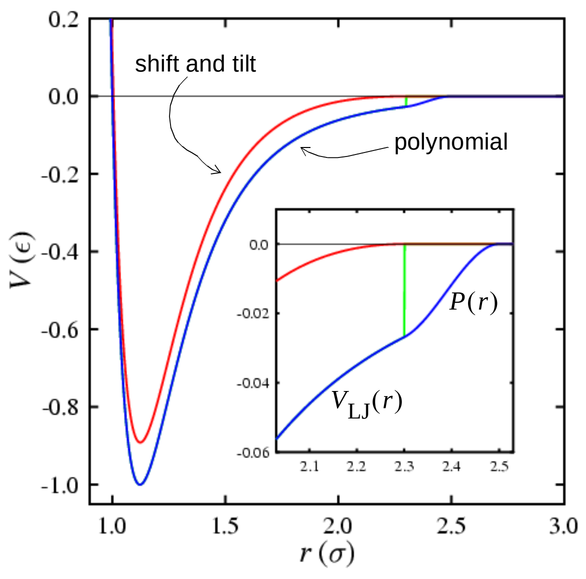
- Shift and tilt the potential: $V(r)$ and $V'(r)$ continuous at r_c:

$$V(r) = V_{LJ}(r) - (r - r_c)V'_{LJ}(r_c) - V_{LJ}(r_c)$$

- Problem: may change the potential at smaller r values

- Fit a polynomial $P(r) = ar^3 + br^2 + cr + d$ from $[r_c, r_c + \Delta r_c]$:

$r_c = 2.3$ Å $\quad \Delta r_c = 0.2$ Å



$$\begin{cases} P(r_c) = V_{LJ}(r_c) \\ P'(r_c) = V'_{LJ}(r_c) \\ P(r_c + \Delta r_c) = 0 \\ P'(r_c + \Delta r_c) = 0 \end{cases} \rightarrow \begin{bmatrix} r_c^3 & r_c^2 & r_c & 1 \\ 3r_c^3 & 2r_c^2 & 1 & 0 \\ (r_c + \Delta r_c)^3 & (r_c + \Delta r_c)^2 & (r_c + \Delta r_c) & 1 \\ 3(r_c + \Delta r_c)^2 & 2(r_c + \Delta r_c) & 1 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} V_{LJ}(r_c) \\ V'_{LJ}(r_c) \\ 0 \\ 0 \end{bmatrix}$$

Systems of linear equations

• In general a system of linear equations is written in the form

$$\mathbf{A}\mathbf{x} = \mathbf{b}$$

and expanded

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1N}x_N = b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \ldots + a_{2N}x_N = b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + \ldots + a_{3N}x_N = b_3 \\ \ldots \\ a_{M1}x_1 + a_{M2}x_2 + a_{M3}x_3 + \ldots + a_{MN}x_N = b_M \end{cases}$$

- N unknowns $x_j, j = 1, 2, \ldots, N$ coupled with M equations
- Coefficients a_{ij} and b_i known

- If $N = M$ the equation may a unique solution $\mathbf{x}$ provided that
  - None of the M equations is a linear combination of another (row degeneracy).
  - All equations do not contain certain variables only in the exactly same linear combinations (column degeneracy).
  - This is equivalent to existence of the inverse of $\mathbf{A}$ ($\mathbf{A}^{-1}$) or that $\det(\mathbf{A}) \neq 0$, or that the only solution to $\mathbf{A}\mathbf{x} = 0$ is $\mathbf{x} = 0$ or that $\mathrm{rank}(\mathbf{A}) = N$.
  - Otherwise matrix $\mathbf{A}$ is *singular*.

  - Roundoff errors in numerical calculations:
    - *near-degeneracy* $\rightarrow$ degeneracy
    - solution found but wrong one (does not solve the original equation)

Systems of linear equations

- If $M > N$ generally no solution exists.
  - We may try to find $\mathbf{x}$ that is nearest to the solution, e.g. in the sense of least squares.

- Different problems related systems of linear equations:

1. Find a solution vector $\mathbf{x}$ for the equation $\mathbf{A}\mathbf{x} = \mathbf{b}$ where $\mathbf{A}$ is a square matrix.

2. Find solutions to many systems in the same calculation: $\mathbf{A}\mathbf{x}_k = \mathbf{b}_k$.
   - Every unknown vector $\mathbf{x}_k$ has a corresponding right-hand side vector $\mathbf{b}_k$.
   - Matrix $\mathbf{A}$ is the same for all equations.

3. Compute the inverse $\mathbf{A}^{-1}$.
   - $\mathbf{A}\mathbf{A}^{-1} = \mathbf{1}$

4. Compute the determinant of a square matrix $\mathbf{A}$.

Elements of the inverse can be expressed as

$$A^{-1}_{ij} = \frac{C_{ji}}{\det(\mathbf{A})}, \quad C_{ji} = ji^{\mathrm{th}} \text{ cofactor}$$

This is not the way to calculate the inverse numerically. It scales badly and is prone to roundoff errors.

Systems of linear equations

• For general (not too large) matrices:

- Methods based on Gauss elimination
- Factorization if many solutions needed

• Large matrices:

- Iterative methods

• Sparse matrices vs. dense matrices

Systems of linear equations: naive Gauss elimination

• A simple method to solve linear equations

- A numerical example

$$\begin{cases} 6x_1 - 2x_2 + 2x_3 + 4x_4 = 16 \\ 12x_1 - 8x_2 + 6x_3 + 10x_4 = 26 \\ 3x_1 - 13x_2 + 9x_3 + 3x_4 = -19 \\ -6x_1 + 4x_2 + x_3 - 18x_4 = -34 \end{cases}$$

1. Subtract equation 1 from other equations so that x_1 is eliminated from them

$$\begin{cases} 6x_1 - 2x_2 + 2x_3 + 4x_4 = 16 \\ -4x_2 + 2x_3 + 2x_4 = -6 \\ -12x_2 + 8x_3 + x_4 = -27 \\ 2x_2 + 3x_3 - 14x_4 = -18 \end{cases}$$

2. Subtract equation 2 from equations 3 and 4 so that x_2 is eliminated

$$\begin{cases} 6x_1 - 2x_2 + 2x_3 + 4x_4 = 16 \\ -4x_2 + 2x_3 + 2x_4 = -6 \\ 2x_3 - 5x_4 = -9 \\ 4x_3 - 13x_4 = -21 \end{cases}$$

Systems of linear equations: naive Gauss elimination

3. Finally eliminate x_3 from the 4th equation

$$\begin{cases} 6x_1 - 2x_2 + 2x_3 + 4x_4 = 16 \\ \quad - 4x_2 + 2x_3 + 2x_4 = -6 \\ \qquad\qquad 2x_3 - 5x_4 = -9 \\ \qquad\qquad\qquad -3x_4 = -3 \end{cases}$$

4. The equation is now in upper triangular form and unknowns can readily be solved by backsubstitution

$$x_4 = \frac{-3}{-3} = 1$$

$$2x_3 - 5 = -9 \Rightarrow x_3 = -2$$

$$-4x_2 - 4 + 2 = -6 \Rightarrow x_2 = 1$$

$$6x_1 - 2 - 4 + 4 = 16 \Rightarrow x_1 = 3$$

- In matrix form

$$\begin{bmatrix} 6 & -2 & 2 & 4 \\ 12 & -8 & 6 & 10 \\ 3 & -13 & 9 & 3 \\ -6 & 4 & 1 & -18 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 16 \\ 26 \\ -19 \\ -34 \end{bmatrix} \text{ is transformed to } \begin{bmatrix} 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \\ 0 & 0 & 2 & -5 \\ 0 & 0 & 0 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 16 \\ -6 \\ -9 \\ -3 \end{bmatrix}$$

Systems of linear equations: naive Gauss elimination

- Check by Matlab

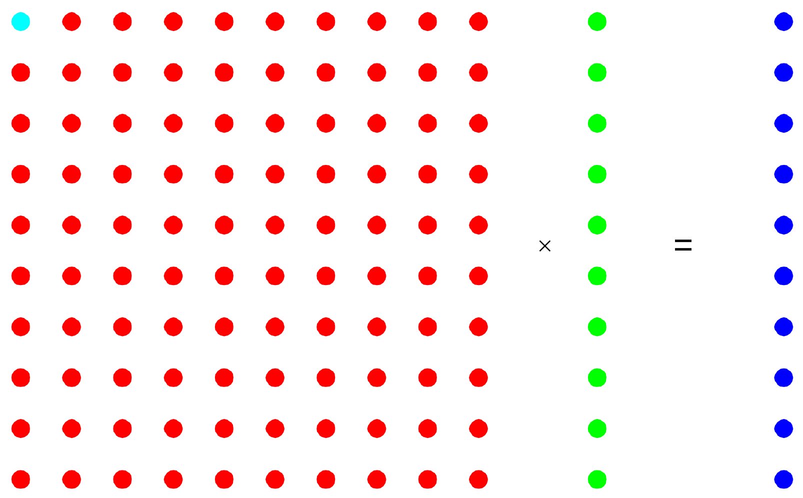
```
>> A=[6 -2 2 4 ; 12 -8 6 10; 3 -13 9 3; -6 4 1 -18]
A =
     6    -2     2     4
    12    -8     6    10
     3   -13     9     3
    -6     4     1   -18
>> b=[16 26 -19 -34]'
b =
    16
    26
   -19
   -34
>> A\b
ans =
                     3
     0.999999999999999
                    -2
                     1
```

```
>> A1=[6 -2 2 4; 0 -4 2 2; 0 0 2 -5; 0 0 0 -3]
A1 =
     6    -2     2     4
     0    -4     2     2
     0     0     2    -5
     0     0     0    -3
>> b1=[16 -6 -9 -3]'
b1 =
    16
    -6
    -9
    -3
>> A1\b1
ans =
     3
     1
    -2
     1
```

```
>> help mldivide
    Backslash or left matrix divide.
    A\B is the matrix division of A into B, which is roughly the
    same as INV(A)*B , except it is computed in a different way.
    If A is an N-by-N matrix and B is a column vector with N
    components, or a matrix with several such columns, then
    X = A\B is the solution to the equation A*X = B computed by
    Gaussian elimination. A warning message is printed if A is
    badly scaled or nearly singular.  A\EYE(SIZE(A)) produces the
    inverse of A.
```

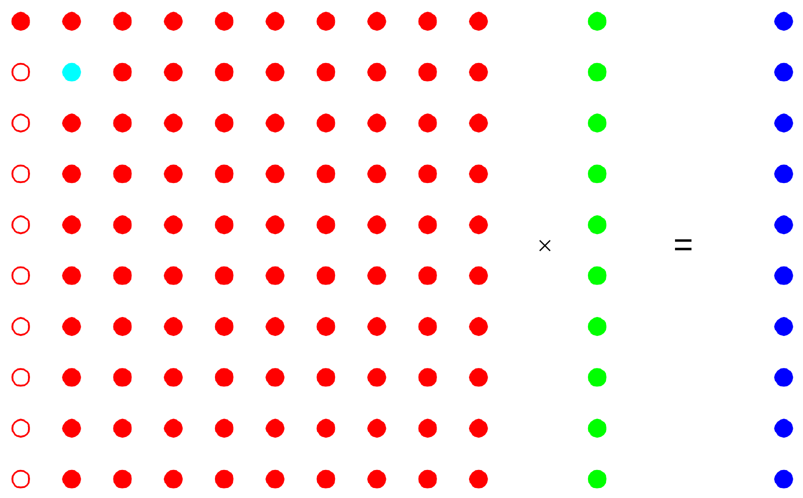
Systems of linear equations: naive Gauss elimination

- Graphically



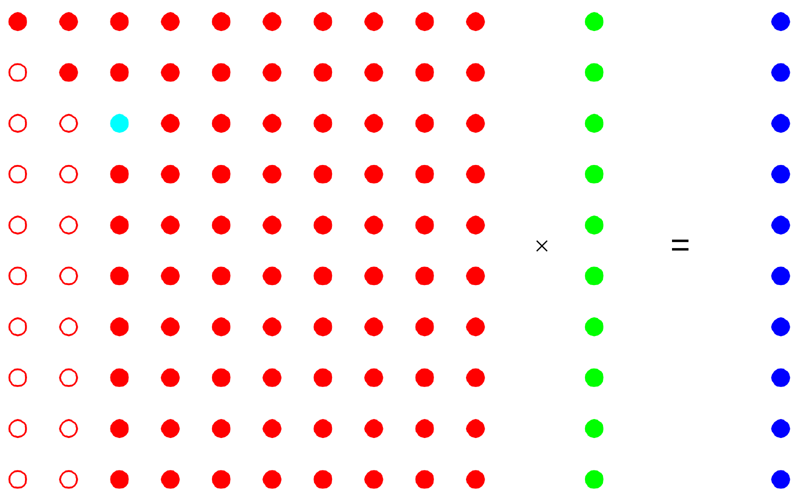
Systems of linear equations: naive Gauss elimination

- Graphically



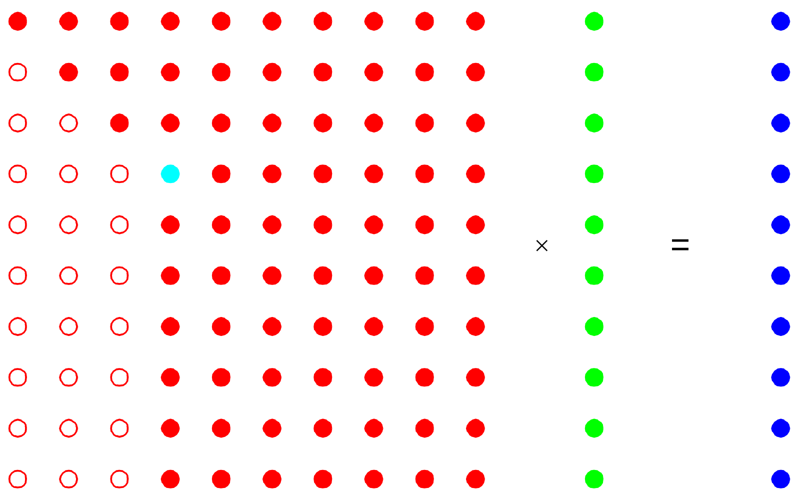
Systems of linear equations: naive Gauss elimination

- Graphically



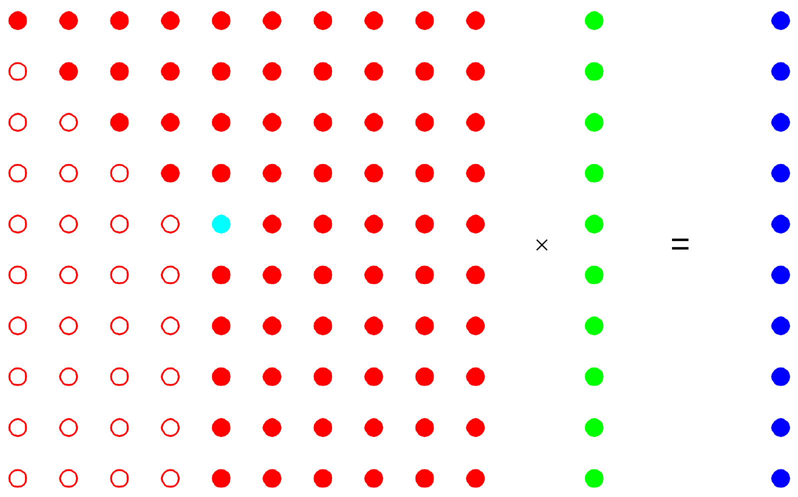
Systems of linear equations: naive Gauss elimination

- Graphically



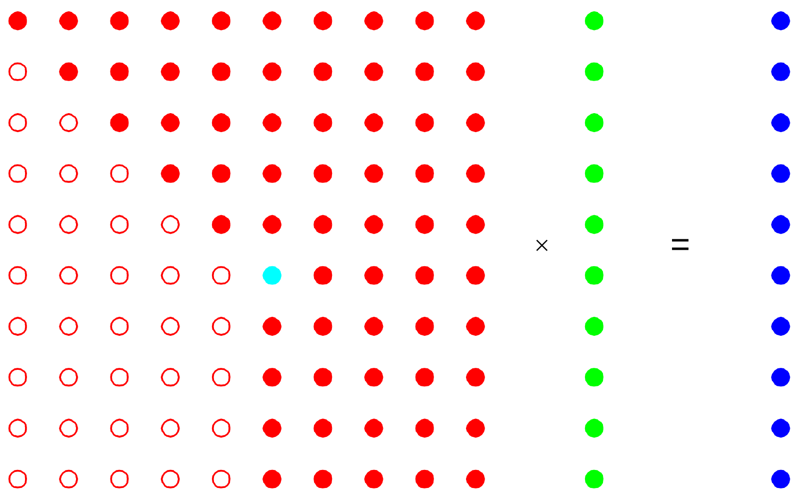
Systems of linear equations: naive Gauss elimination

- Graphically



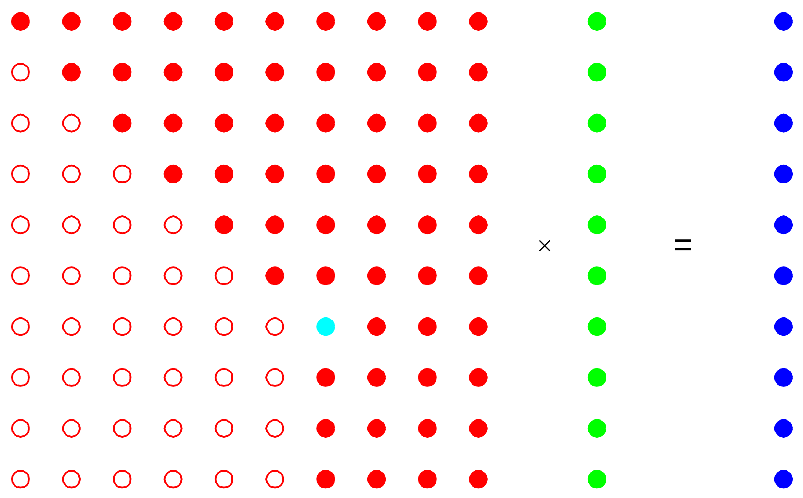
Systems of linear equations: naive Gauss elimination

- Graphically



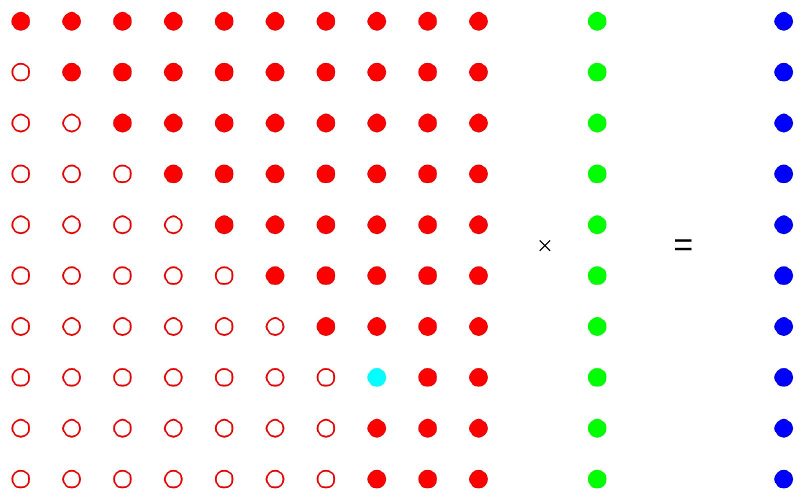
Systems of linear equations: naive Gauss elimination

- Graphically



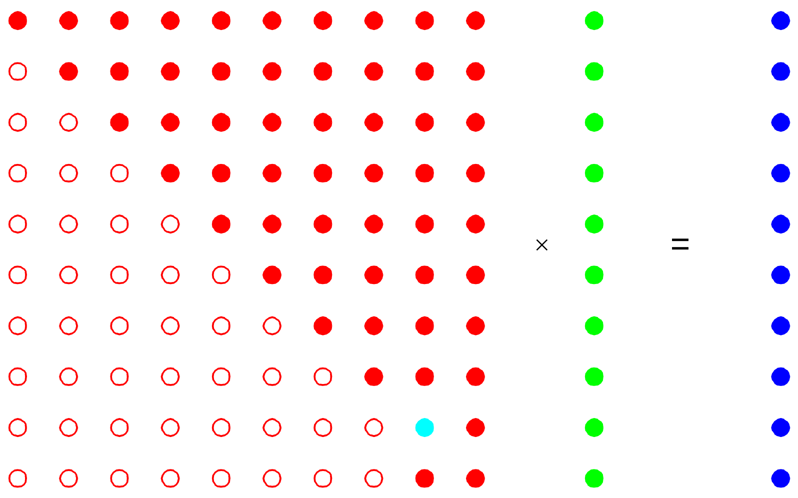
Systems of linear equations: naive Gauss elimination

- Graphically



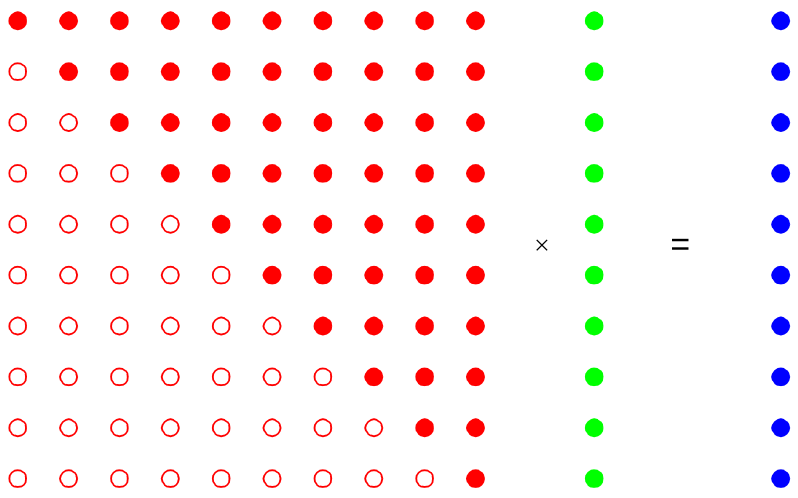
Systems of linear equations: naive Gauss elimination

- Graphically



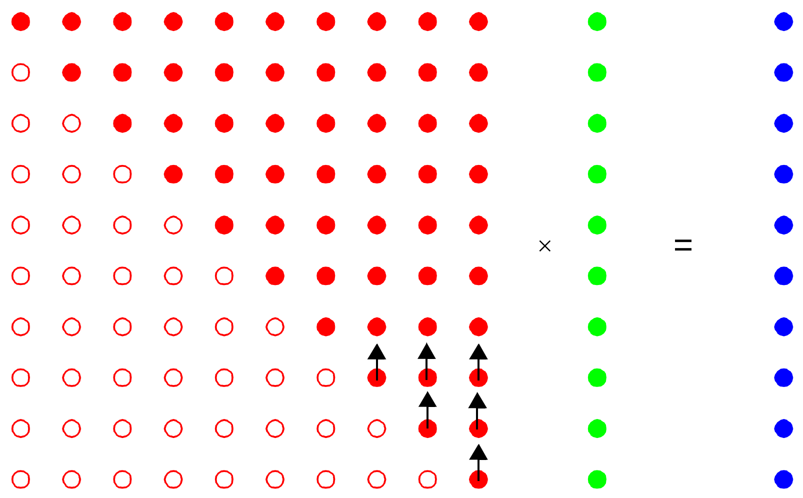
Systems of linear equations: naive Gauss elimination

- Graphically



Systems of linear equations: naive Gauss elimination

- Graphically: backsubstitution



Systems of linear equations: naive Gauss elimination

• Naive Gauss elimination as an algorithm

- Equation has the form

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1N}x_N = b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \ldots + a_{2N}x_N = b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + \ldots + a_{3N}x_N = b_3 \\ \ldots \\ a_{N1}x_1 + a_{N2}x_2 + a_{N3}x_3 + \ldots + a_{NN}x_N = b_N \end{cases}$$

- Elimination consists of $N-1$ steps.
  - At the kth step ($k = 1, \ldots, N-1$) the following substitutions are done to equation i ($k+1 \le i \le N$)

$$\begin{cases} a_{ij} \leftarrow a_{ij} - \left(\frac{a_{ik}}{a_{kk}}\right) a_{kj} \\ b_i \leftarrow b_i - \left(\frac{a_{ik}}{a_{kk}}\right) b_k \end{cases}, k \le j \le N$$

- Backsubstitution

$$x_i = \frac{1}{a_{ii}}\left[b_i - \sum_{j=i+1}^{N} a_{ij}x_j\right], \quad i = N, N-1, \ldots, 1$$

Systems of linear equations: naive Gauss elimination

- Then the bad news: in practice naive Gauss elimination can fail badly.
  - Take for example the following equation

$$\begin{cases} 0x_1 + x_2 = 1 \\ x_1 + x_2 = 2 \end{cases}$$

  - Zero coefficient in the first line prevents the application of Gauss elimination.

  - In numerical computation it need not be exactly zero:

$$\begin{cases} \varepsilon x_1 + x_2 = 1 \\ x_1 + x_2 = 2 \end{cases}$$

    - Here ε is something very small.

    - After the first elimination step

$$\begin{cases} \varepsilon x_1 + x_2 = 1 \\ \left(1 - \frac{1}{\varepsilon}\right) x_2 = 2 - \frac{1}{\varepsilon} \end{cases}$$

    - After backsubstitution

$$x_2 = \frac{\left(2 - \frac{1}{\varepsilon}\right)}{\left(1 - \frac{1}{\varepsilon}\right)}, \qquad x_1 = \frac{1 - x_2}{\varepsilon}$$

Note: We can not substitute expression for x_2 to that of x_1 and simplify it because computer does not do that.

Systems of linear equations: naive Gauss elimination

- ε small $\rightarrow$ $1/\varepsilon$ large $\rightarrow$ $2 - \frac{1}{\varepsilon} \approx 1 - \frac{1}{\varepsilon} \approx -\frac{1}{\varepsilon}$

- For the solution we get $x_1 \approx 0$, $x_2 \approx 1$

- Right solution $x_1 = \frac{1}{1-\varepsilon} \approx 1$, $x_2 = \frac{1-2\varepsilon}{1-\varepsilon} \approx 1$

- Error of 100% !

- Change the order of equations: a working solution

$$\begin{cases} x_1 + x_2 = 2 \\ \varepsilon x_1 + x_2 = 1 \end{cases} \Rightarrow \begin{cases} x_1 + x_2 = 2 \\ (1-\varepsilon) x_2 = 1 - 2\varepsilon \end{cases} \Rightarrow \begin{cases} x_2 = \frac{1-2\varepsilon}{1-\varepsilon} \approx 1 \\ x_1 = 2 - x_2 \approx 1 \end{cases}$$

- Order of elimination matters $\rightarrow$ pivoting.

Systems of linear equations: pivoting

- When the element of the matrix $\mathbf{A}$ that is to be eliminated (a_{kk}, so called *pivot element*) happens to be zero elimination can not be done.

$$\begin{cases} a_{ij} \leftarrow a_{ij} - \left(\frac{a_{ik}}{a_{kk}}\right) a_{kj} \\ b_i \leftarrow b_i - \left(\frac{a_{ik}}{a_{kk}}\right) b_k \end{cases}, k \le j \le N$$

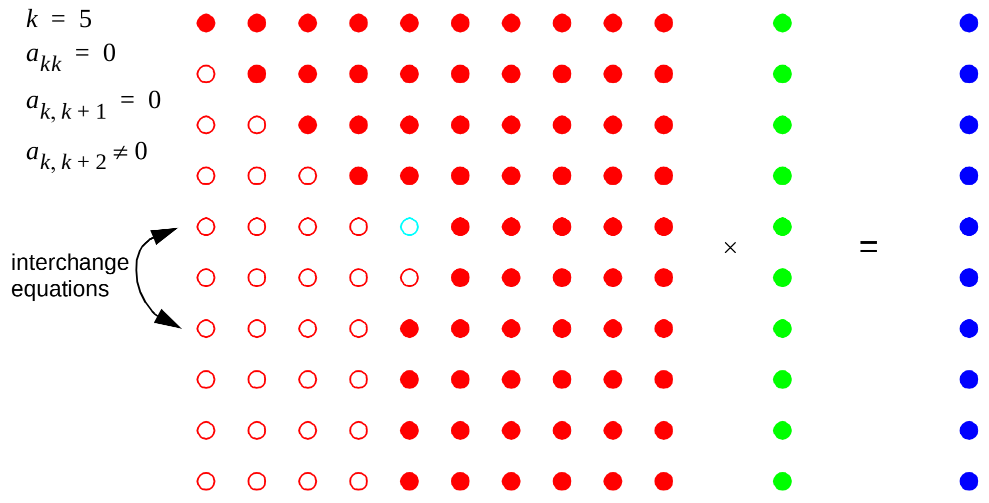
  - Interchanging equations is allowed
  - Find the first equation “below” the kth equation that has coefficient in column k non-zero.

$k = 5$

$a_{kk} = 0$

$a_{k,k+1} = 0$

$a_{k,k+2} \neq 0$



Systems of linear equations: pivoting

- Partial pivoting: shuffle only equations (rows in the matrix)
- Full pivoting possible: bookkeeping in the program becomes complicated

- Strategy: shuffle equations in such a way that the pivoting element is the largest possible
  - Index vector

$$\mathbf{l} = \begin{bmatrix} l_1 & l_2 & \dots & l_N \end{bmatrix}$$

tells the order in which the equations are handled

  - In the beginning calculate scaling factors s_i for all equations

$$s_i = \max_j(|a_{ij}|),\ 1 \le i \le N$$

  - From these form the vector

$$\mathbf{s} = \begin{bmatrix} s_1 & s_2 & \dots & s_N \end{bmatrix}$$

  - First elimination for the equation for which $\frac{|a_{i1}|}{s_i}$ is largest. Let’s call it equation l_1.
  - Equation l_1 is subtracted from all others to zero matrix elements a_{i1}.

Systems of linear equations: pivoting

- A handy way to do the index bookkeeping is the following
  - In the beginning set $\mathbf{l} = [1\ 2\ \dots\ N]$
  - Choose j so that it has the maximum value of $\left\{ \frac{|a_{l_i 1}|}{s_{l_i}} \middle| 1 \le i \le N \right\}$.
  - Exchange l_j and l_1 in index vector $\mathbf{l}$.
  - Now subtract equation 1 multiplied with coefficients $\frac{a_{l_i 1}}{a_{l_1 1}}$ from equations i, $2 \le i \le N$.
  - Generally at step k
    - Choose the index j so that $\left\{ \frac{|a_{l_i k}|}{s_{l_i}} \middle| k \le i \le N \right\}$ has the maximum value.
    - Exchange indices l_j and l_k.
    - Use coefficients $\frac{a_{l_i k}}{a_{l_k k}}$ when subtracting the pivot equation l_k from other equations l_i, $k+1 \le i \le N$.
    - Normally the scaling vector $\mathbf{s}$ is not updated during computation. General belief says that it not worth the trouble.

Systems of linear equations: pivoting

- A numerical example: the original equation

$$\begin{bmatrix} 3 & -13 & 9 & 3 \\ -6 & 4 & 1 & -18 \\ 6 & -2 & 2 & 4 \\ 12 & -8 & 6 & 10 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} -19 \\ -34 \\ 16 \\ 26 \end{bmatrix}$$

- In the beginning $\mathbf{l} = [1\ 2\ 3\ 4]$ and $\mathbf{s} = [13\ 18\ 6\ 12]$.
- Calculate ratios $\left\{ \frac{|a_{l_i 1}|}{s_{l_i}} \middle| i = 1, 2, 3, 4 \right\} = \left\{ \frac{3}{13}, \frac{6}{18}, \frac{6}{6}, \frac{12}{12} \right\}$.
- Choose j as the first maximum value in this vector: $j = 3$.
- After exchange index vector is $\mathbf{l} = [3\ 2\ 1\ 4]$.
- Subtract equation 3 (in the original equation) from others weighted by appropriate coefficients:

$$\begin{bmatrix} 0 & -12 & 8 & 1 \\ 0 & 2 & 3 & -14 \\ 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} -27 \\ -18 \\ 16 \\ -6 \end{bmatrix}.$$

- Next find the maximum from $\left\{ \frac{|a_{l_i 2}|}{s_{l_i}} \middle| i = 2, 3, 4 \right\} = \left\{ \frac{2}{18}, \frac{12}{13}, \frac{4}{12} \right\}$.
- Results is $j = 3$ and the new index vector is $\mathbf{l} = [3\ 1\ 2\ 4]$.

Systems of linear equations: pivoting

- Equation 1 is subtracted from equations 2 and 4 multiplied by $-1/6$ and $1/3$, respectively:

$$\begin{bmatrix} 0 & -12 & 8 & 1 \\ 0 & 0 & \frac{13}{3} & -\frac{83}{6} \\ 6 & -2 & 2 & 4 \\ 0 & 0 & -\frac{2}{3} & \frac{5}{3} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} -27 \\ -\frac{45}{2} \\ 16 \\ 3 \end{bmatrix}$$

- In the last step get maximum from $\left\{ \frac{|a_{l_i 3}|}{s_{l_i}} \middle| i = 3, 4 \right\} = \left\{ \frac{13/3}{18}, \frac{2/3}{12} \right\}$.

- We get $j = 3$ and the index vector remains unchanged: $\mathbf{l} = \begin{bmatrix} 3 & 1 & 2 & 4 \end{bmatrix}$.

- Equation 2 multiplied by $-2/13$ is subtracted from equation 4:

$$\begin{bmatrix} 0 & -12 & 8 & 1 \\ 0 & 0 & \frac{13}{3} & -\frac{83}{6} \\ 6 & -2 & 2 & 4 \\ 0 & 0 & 0 & -\frac{6}{13} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} -27 \\ -\frac{45}{2} \\ 16 \\ -\frac{6}{13} \end{bmatrix}.$$

Systems of linear equations: pivoting

- Index vector is now $\mathbf{l} = \begin{bmatrix} 3 & 1 & 2 & 4 \end{bmatrix}$ and the solution is obtained by going through it starting from the end:

$$x_4 = \frac{-6/13}{-6/13} = 1$$

$$x_2 = \frac{-27 - 8(-2) - 1(1)}{-12} = 1$$

$$x_1 = \frac{16 + 2(1) - 2(-2) - 4(1)}{6} = 3$$

$$x_3 = \frac{(-45/2) + (83/6)(1)}{13/3} = -2 .$$

- Solution is $\mathbf{x} = \begin{bmatrix} 3 \\ 1 \\ -2 \\ 1 \end{bmatrix}$.

Systems of linear equations: Gauss-Jordan elimination

• In Gauss elimination we get the solution corresponding to only one vector $\mathbf{b}$.

• Gauss-Jordan: solutions for many $\mathbf{b}$ at the same time and also the inverse $\mathbf{A}^{-1}$.

- 'Augment' the equation into form

$$\mathbf{A} \cdot [\mathbf{x}_1 \cup \mathbf{x}_2 \cup \mathbf{x}_3 \cup \mathbf{Y}] = [\mathbf{b}_1 \cup \mathbf{b}_2 \cup \mathbf{b}_3 \cup \mathbf{1}]$$

- Operation $\mathbf{A} \cup \mathbf{B}$ denotes the combination of the two matrices:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \cup \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & b_{11} & b_{12} & b_{13} \\ a_{21} & a_{22} & a_{23} & b_{21} & b_{22} & b_{23} \\ a_{31} & a_{32} & a_{33} & b_{31} & b_{32} & b_{33} \end{bmatrix}$$

- It easy to see that the above equation corresponds to four equations:

$$\mathbf{A} \cdot \mathbf{x}_1 = \mathbf{b}_1$$
$$\mathbf{A} \cdot \mathbf{x}_2 = \mathbf{b}_2$$
$$\mathbf{A} \cdot \mathbf{x}_3 = \mathbf{b}_3$$
$$\mathbf{A} \cdot \mathbf{Y} = \mathbf{1}$$

Systems of linear equations: Gauss-Jordan elimination

- The following operations do not change the equations
  1. Exchange of two rows in matrix $\mathbf{A}$ and exchange of the corresponding rows in vectors $\mathbf{b}_i$ and unit matrix 1.
  2. Substitute a row in $\mathbf{A}$ by a linear combination of all rows in $\mathbf{A}$ and the corresponding changes in $\mathbf{b}_i$ and $\mathbf{1}$.
  3. Exchange two columns in $\mathbf{A}$ and exchange corresponding rows in $\mathbf{x}$ and $\mathbf{Y}$.

- Gauss-Jordan (GJ) elimination (with pivoting): use the abovementioned operations to change $\mathbf{A}$ into a unit matrix.
  - Equations now read as:

$$\mathbf{1} \cdot \mathbf{x}_1 = \mathbf{b}'_1 \qquad \mathbf{1} \cdot \mathbf{x}_2 = \mathbf{b}'_2 \qquad \mathbf{1} \cdot \mathbf{x}_3 = \mathbf{b}'_3 \qquad \mathbf{1} \cdot \mathbf{Y} = \mathbf{A}^{-1}$$

  - So we get solutions for many vectors $\mathbf{b}$ and the inverse $\mathbf{A}^{-1}$.

  - GJ elimination (without pivoting) goes like this
    1. The first row is divided by a_{11}.
    2. The first row is subtracted from other rows scaled in such a way that $a_{i1} = 0$.
    3. Now the first column corresponds to unit matrix.
    4. The second row is divided by a_{22}.
    5. The second row is subtracted from other rows scaled in such a way that $a_{i2} = 0$.

    ...

Systems of linear equations: Gauss-Jordan elimination

- Example: a 3×3 matrix

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \rightarrow \begin{bmatrix} 1 & a'_{12} & a'_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \rightarrow \begin{bmatrix} 1 & a'_{12} & a'_{13} \\ 0 & a'_{22} & a'_{23} \\ 0 & a'_{32} & a'_{33} \end{bmatrix} \rightarrow \begin{bmatrix} 1 & a'_{12} & a'_{13} \\ 0 & 1 & a''_{23} \\ 0 & a'_{32} & a'_{33} \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & a''_{13} \\ 0 & 1 & a''_{23} \\ 0 & 0 & a''_{33} \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & a''_{13} \\ 0 & 1 & a''_{23} \\ 0 & 0 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- The corresponding changes are made to the vectors $\mathbf{b}_i$ and to the matrix $\mathbf{Y}$.

- Gauss-Jordan is also prone to rounoff errors if pivoting is not used.
- In pivoting the order in which the elements a_{ii} are handled is changed.
    - The ciriterion is usually to choose the largest element among the candidates.

Systems of linear equations: Gauss-Jordan elimination

- Numerical Recipes routine for GJ elimination with full pivoting:

```
#include <math.h>
#define NRANSI
#include "nrutil.h"
#define SWAP(a,b) {temp=(a);(a)=(b);(b)=temp;}

void gaussj(float **a, int n, float **b, int m)
{
  int *indxc,*indxr,*ipiv;
  int i,icol,irow,j,k,l,ll;
  float big,dum,pivinv,temp;

  indxc=ivector(1,n);
  indxr=ivector(1,n);
  ipiv=ivector(1,n);
  for (j=1;j<=n;j++) ipiv[j]=0;
  for (i=1;i<=n;i++) {
    big=0.0;
    for (j=1;j<=n;j++)
      if (ipiv[j] != 1)
   for (k=1;k<=n;k++) {
     if (ipiv[k] == 0) {
       if (fabs(a[j][k]) >= big) {
         big=fabs(a[j][k]);
         irow=j;
         icol=k;
       }
     } else if (ipiv[k] > 1) nrerror("gaussj: Singular Matrix-1");
   }
    ++(ipiv[icol]);
    if (irow != icol) {
      for (l=1;l<=n;l++) SWAP(a[irow][l],a[icol][l])
   for (l=1;l<=m;l++) SWAP(b[irow][l],b[icol][l])
     }
    indxr[i]=irow;
    indxc[i]=icol;
    for (l=1;l<=n;l++) a[icol][l] *= pivinv;
     for (l=1;l<=m;l++) b[icol][l] *= pivinv;
     for (ll=1;ll<=n;ll++)
       if (ll != icol) {
    dum=a[ll][icol];
   a[ll][icol]=0.0;
    for (l=1;l<=n;l++) a[ll][l] -= a[icol][l]*dum;
    for (l=1;l<=m;l++) b[ll][l] -= b[icol][l]*dum;
       }
  }
  for (l=n;l>=1;l--) {
    if (indxr[l] != indxc[l])
      for (k=1;k<=n;k++)
         SWAP(a[k][indxr[l]],a[k][indxc[l]]);
  }
  free_ivector(ipiv,1,n);
  free_ivector(indxr,1,n);
  free_ivector(indxc,1,n);
}
#undef SWAP
#undef NRANSI
```

Systems of linear equations: Gauss-Jordan elimination

• Scaling behavior of matrix inversion by Gauss(-Jordan)

Elimination consists of $N-1$ steps. $O(N)$
At the kth step ($k = 1, \ldots, N-1$) the following substitutions
are done to equation i ($k+1 \le i \le N$) $O(N)$

$$a_{ij} \leftarrow a_{ij} - \left(\frac{a_{ik}}{a_{kk}}\right)a_{kj},\ b_i \leftarrow b_i - \left(\frac{a_{ik}}{a_{kk}}\right)b_k,\ k \le j \le N \qquad O(N)$$

$\rightarrow$ scales as $O(N^3)$!

• When we have to solve many equations with the same $\mathbf{A}$ there's a remedy: LU factorization

- Factorization scales as $O(N^3)$.
- Results of factorization can be used to solve any equation corresponding to matrix $\mathbf{A}$ in $O(N^2)$.

Systems of linear equations: LU factorization

• LU (**l**ower triangular—**u**pper triangular) factorization (or decomposition):

- Find matrices $\mathbf{L}$ and $\mathbf{U}$ so that
  - $\mathbf{L} \cdot \mathbf{U} = \mathbf{A}$
  - $\mathbf{L}$ is a lower triangular matrix
  - $\mathbf{U}$ is an upper triangular matrix
- E.g. for a 4×4 matrix

$$\underset{\mathbf{L}}{\begin{bmatrix} \alpha_{11} & 0 & 0 & 0 \\ \alpha_{21} & \alpha_{22} & 0 & 0 \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & 0 \\ \alpha_{41} & \alpha_{42} & \alpha_{43} & \alpha_{44} \end{bmatrix}} \cdot \underset{\mathbf{U}}{\begin{bmatrix} \beta_{11} & \beta_{12} & \beta_{13} & \beta_{14} \\ 0 & \beta_{22} & \beta_{23} & \beta_{24} \\ 0 & 0 & \beta_{33} & \beta_{34} \\ 0 & 0 & 0 & \beta_{44} \end{bmatrix}} = \underset{\mathbf{A}}{\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}}$$

- Substituting $\mathbf{L} \cdot \mathbf{U}$ for $\mathbf{A}$ in the linear equation we get

$$\mathbf{A} \cdot \mathbf{x} = (\mathbf{L} \cdot \mathbf{U}) \cdot \mathbf{x} = \mathbf{L} \cdot (\mathbf{U} \cdot \mathbf{x}) = \mathbf{b}$$

- Now the equation can be solved in two steps

$$\mathbf{L} \cdot \mathbf{y} = \mathbf{b} \quad , \quad \mathbf{U} \cdot \mathbf{x} = \mathbf{y}$$

Systems of linear equations: LU factorization

- What do we gain here?

  - When the matrix of the equation is in the triangular form solving it is trivial.
  - Just use forward substitution for $\mathbf{y}$:

$$y_1 = \frac{b_1}{\alpha_{11}},\ y_i = \frac{1}{\alpha_{ii}}\left[b_i - \sum_{j=1}^{i-1} \alpha_{ij} y_j\right],\ i = 2, 3, \ldots, N$$

  - And back substitution for $\mathbf{x}$:

$$x_N = \frac{y_N}{\beta_{NN}},\ x_i = \frac{1}{\beta_{ii}}\left[y_i - \sum_{j=i+1}^{N} \beta_{ij} x_j\right],\ i = N-1, N-2, \ldots, 1$$

  - The reason for doing all this is that factorization is an $O(N^3)$ operation but solving the equation using $\mathbf{L}$ and $\mathbf{U}$ is $O(N^2)$.

Systems of linear equations: LU factorization

- Actually, the naive Gauss elimination does exactly LU factorization on the matrix $\mathbf{A}$.

- Let's take a numerical example

$$\begin{bmatrix} 6 & -2 & 2 & 4 \\ 12 & -8 & 6 & 10 \\ 3 & -13 & 9 & 3 \\ -6 & 4 & 1 & -18 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 16 \\ 26 \\ -19 \\ -34 \end{bmatrix}$$

- Doing the Gauss elimination we get the equation into form

$$\begin{bmatrix} 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \\ 0 & 0 & 2 & -5 \\ 0 & 0 & 0 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 16 \\ -6 \\ -9 \\ -3 \end{bmatrix}$$

- This can be interpreted as transforming the equation $\mathbf{Ax} = \mathbf{b}$ into

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

Systems of linear equations: LU factorization

- Here $\mathbf{M}$ is chosen in such a way that $\mathbf{MA}$ is in the upper triangular form

$$\mathbf{MA} = \begin{bmatrix} 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \\ 0 & 0 & 2 & -5 \\ 0 & 0 & 0 & -3 \end{bmatrix} \equiv \mathbf{U}$$

- Forward elimination consists of a series of steps
  - The first step gives

$$\begin{bmatrix} 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \\ 0 & -12 & 8 & 1 \\ 0 & 2 & 3 & -14 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 16 \\ -6 \\ -27 \\ -18 \end{bmatrix}$$

or $\mathbf{M}_1\mathbf{Ax} = \mathbf{M}_1\mathbf{b}$ where

$$\mathbf{M}_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -2 & 1 & 0 & 0 \\ -\frac{1}{2} & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

Systems of linear equations: LU factorization

- The second step gives

$$\begin{bmatrix} 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \\ 0 & 0 & 2 & -5 \\ 0 & 0 & 4 & -13 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 16 \\ -6 \\ -9 \\ -21 \end{bmatrix}$$

or $\mathbf{M}_2\mathbf{M}_1\mathbf{Ax} = \mathbf{M}_2\mathbf{M}_1\mathbf{b}$ where

$$\mathbf{M}_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -3 & 1 & 0 \\ 0 & \frac{1}{2} & 0 & 1 \end{bmatrix}$$

- Finally the third step gives the upper triangular form

$$\begin{bmatrix} 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \\ 0 & 0 & 2 & -5 \\ 0 & 0 & 0 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 16 \\ -6 \\ -9 \\ -3 \end{bmatrix}$$

Systems of linear equations: LU factorization

- This is equivalent to $\mathbf{M}_3\mathbf{M}_2\mathbf{M}_1\mathbf{A}\mathbf{x} = \mathbf{M}_3\mathbf{M}_2\mathbf{M}_1\mathbf{b}$ where

$$\mathbf{M}_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -2 & 1 \end{bmatrix}$$

- Thus we get the matrix $\mathbf{M}$ as the product of all three *multiplier* matrices

$$\mathbf{M} = \mathbf{M}_3\mathbf{M}_2\mathbf{M}_1$$

- We wrote $\mathbf{MA} = \mathbf{U} \rightarrow \mathbf{A} = \mathbf{M}^{-1}\mathbf{U} = \mathbf{M}_1^{-1}\mathbf{M}_2^{-1}\mathbf{M}_3^{-1}\mathbf{U} = \mathbf{LU}$

- Now $\mathbf{M}_i$ have such a simple structure (unit diagonal, lower triangular, only one column nonzero) that their inverse is obtained simply by inverting the sings of the nondiagonal elements:

$$\mathbf{L} = \mathbf{M}_1^{-1}\mathbf{M}_2^{-1}\mathbf{M}_3^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ \frac{1}{2} & 0 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 3 & 1 & 0 \\ 0 & -\frac{1}{2} & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ \frac{1}{2} & 3 & 1 & 0 \\ -1 & -\frac{1}{2} & 2 & 1 \end{bmatrix}$$

Systems of linear equations: LU factorization

- It is easy to verify that

$$\mathbf{LU} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ \frac{1}{2} & 3 & 1 & 0 \\ -1 & -\frac{1}{2} & 2 & 1 \end{bmatrix} \begin{bmatrix} 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \\ 0 & 0 & 2 & -5 \\ 0 & 0 & 0 & -3 \end{bmatrix} = \begin{bmatrix} 6 & -2 & 2 & 4 \\ 12 & -8 & 6 & 10 \\ 3 & -13 & 9 & 3 \\ -6 & 4 & 1 & -18 \end{bmatrix} = \mathbf{A}$$

- In summary:
  - The lower triangular elements of matrix $\mathbf{L}$ are the multipliers located at the positions of the elements their annihilated from $\mathbf{A}$.
  - $\mathbf{U}$ is the final coefficient matrix obtained after the forward elimination phase.

Systems of linear equations: LU factorization

- Let's look at the factorization for a 4x4 matrix

$$\begin{bmatrix}\alpha_{11} & 0 & 0 & 0\\ \alpha_{21} & \alpha_{22} & 0 & 0\\ \alpha_{31} & \alpha_{32} & \alpha_{33} & 0\\ \alpha_{41} & \alpha_{42} & \alpha_{43} & \alpha_{44}\end{bmatrix} \cdot \begin{bmatrix}\beta_{11} & \beta_{12} & \beta_{13} & \beta_{14}\\ 0 & \beta_{22} & \beta_{23} & \beta_{24}\\ 0 & 0 & \beta_{33} & \beta_{34}\\ 0 & 0 & 0 & \beta_{44}\end{bmatrix} = \begin{bmatrix}a_{11} & a_{12} & a_{13} & a_{14}\\ a_{21} & a_{22} & a_{23} & a_{24}\\ a_{31} & a_{32} & a_{33} & a_{34}\\ a_{41} & a_{42} & a_{43} & a_{44}\end{bmatrix}$$

- On the left hand side we have $N^2 + N$ coefficients; on the right hand side N^2
  - We can set N coefficients as we like.
  - Normal convention $\alpha_{ii} = 1$ (this is understandable since $\mathbf{L}$ is the forward elimination matrix):

$$\begin{bmatrix}1 & 0 & 0 & 0\\ \alpha_{21} & 1 & 0 & 0\\ \alpha_{31} & \alpha_{32} & 1 & 0\\ \alpha_{41} & \alpha_{42} & \alpha_{43} & 1\end{bmatrix} \cdot \begin{bmatrix}\beta_{11} & \beta_{12} & \beta_{13} & \beta_{14}\\ 0 & \beta_{22} & \beta_{23} & \beta_{24}\\ 0 & 0 & \beta_{33} & \beta_{34}\\ 0 & 0 & 0 & \beta_{44}\end{bmatrix} = \begin{bmatrix}a_{11} & a_{12} & a_{13} & a_{14}\\ a_{21} & a_{22} & a_{23} & a_{24}\\ a_{31} & a_{32} & a_{33} & a_{34}\\ a_{41} & a_{42} & a_{43} & a_{44}\end{bmatrix}$$

  - Nice from the practical point of view:
    both $\mathbf{L}$ and $\mathbf{U}$ can be stored in the same array:

$$\begin{bmatrix}\beta_{11} & \beta_{12} & \beta_{13} & \beta_{14}\\ \alpha_{21} & \beta_{22} & \beta_{23} & \beta_{24}\\ \alpha_{31} & \alpha_{32} & \beta_{33} & \beta_{34}\\ \alpha_{41} & \alpha_{42} & \alpha_{43} & \beta_{44}\end{bmatrix}$$

Systems of linear equations: LU factorization

- How to do the LU factorization efficiently?

$$\begin{bmatrix}\alpha_{11} & 0 & 0 & 0\\ \alpha_{21} & \alpha_{22} & 0 & 0\\ \alpha_{31} & \alpha_{32} & \alpha_{33} & 0\\ \alpha_{41} & \alpha_{42} & \alpha_{43} & \alpha_{44}\end{bmatrix} \cdot \begin{bmatrix}\beta_{11} & \beta_{12} & \beta_{13} & \beta_{14}\\ 0 & \beta_{22} & \beta_{23} & \beta_{24}\\ 0 & 0 & \beta_{33} & \beta_{34}\\ 0 & 0 & 0 & \beta_{44}\end{bmatrix} = \begin{bmatrix}a_{11} & a_{12} & a_{13} & a_{14}\\ a_{21} & a_{22} & a_{23} & a_{24}\\ a_{31} & a_{32} & a_{33} & a_{34}\\ a_{41} & a_{42} & a_{43} & a_{44}\end{bmatrix}$$

- ij^{th} equation of this group reads (i.e. for element a_{ij})

$$\alpha_{i1}\beta_{1j} + \alpha_{i2}\beta_{2j} + \ldots = a_{ij}$$

- How many terms are included in the equation depends on the order of i and j:

$$\sum_{k=1}^{i} \alpha_{ik}\beta_{kj} = a_{ij}, \quad i \le j \qquad (1)$$

$$\sum_{k=1}^{j} \alpha_{ik}\beta_{kj} = a_{ij}, \quad i > j \qquad (2)$$

- As noted before we have here N^2 equations but $N^2 + N$ unknowns. By setting $\alpha_{ii} = 1$ this is corrected.

Systems of linear equations: LU factorization

- One efficient way to do the LU factorization is **Crout's algorithm**

  - Set $\alpha_{ii} = 1 \;\; i = 1, \ldots, N$ (3)

  - For every $i = 1, \ldots, N$ do the following:

    1. For all $j = 1, \ldots, i$ solve β_{ij} using the equations (1), (3):

    $$\beta_{ij} = a_{ij} - \sum_{k=1}^{i-1} \alpha_{ik}\beta_{kj}, \text{ (if } i = 1 \text{ no summing)}$$

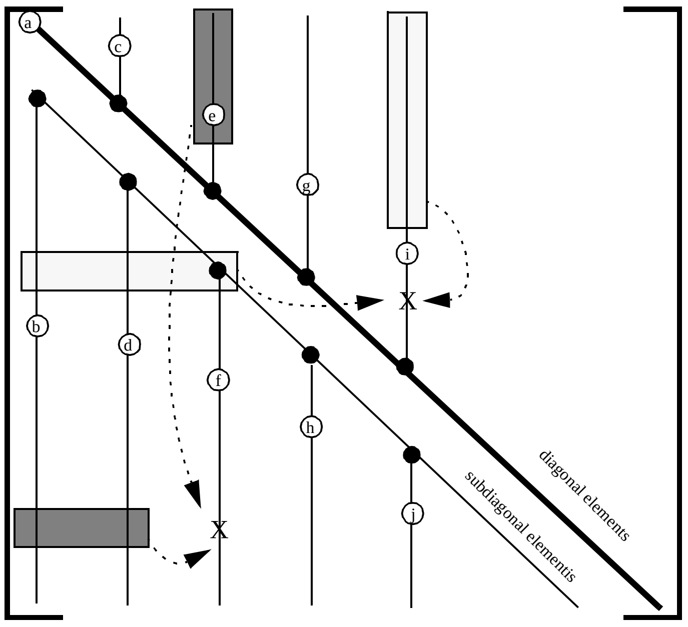
    2. For all $i = j+1, j+2, \ldots, N$ solve α_{ij} using equation (2):

    $$\alpha_{ij} = \frac{1}{\beta_{jj}}\left(a_{ij} - \sum_{k=1}^{j-1} \alpha_{ik}\beta_{kj}\right)$$

  - By going through a couple of iterations it is easy to see that those α's and β's that appear on the RHS are already determined when they are needed.

Systems of linear equations: LU factorization

- Filling of the $\alpha\beta$ matrix:
  columns from left to right
  every column from top to bottom:



Systems of linear equations: LU factorization

- The most simple non-trivial example 3x3 matrix:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} 9 & 2 & 3 \\ 4 & 2 & 4 \\ 1 & 1 & 9 \end{bmatrix}$$

- Calculate the factorization

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ \alpha_{21} & 1 & 0 \\ \alpha_{31} & \alpha_{32} & 1 \end{bmatrix} \cdot \begin{bmatrix} \beta_{11} & \beta_{12} & \beta_{13} \\ 0 & \beta_{22} & \beta_{23} \\ 0 & 0 & \beta_{33} \end{bmatrix}$$

Systems of linear equations: LU factorization

$j = 1$ $i = 1$ $\beta_{11} = a_{11} = 9$

$i = 2$ $\alpha_{21} = \frac{1}{\beta_{11}} a_{21} = \frac{4}{9} = 0.4444$

$i = 3$ $\alpha_{31} = \frac{1}{\beta_{11}} a_{31} = \frac{1}{9} = 0.1111$

$j = 2$ $i = 1$ $\beta_{12} = a_{12} = 2$

$i = 2$ $\beta_{22} = a_{22} - \alpha_{21}\beta_{12} = 2 - \frac{4}{9}2 = 1.111$

$i = 3$ $\alpha_{32} = \frac{1}{\beta_{22}}(a_{32} - \alpha_{31}\beta_{12}) = \frac{1}{1.111}\left(1 - \frac{2}{9}\right) = 0.7000$

$j = 3$ $i = 1$ $\beta_{13} = a_{13} = 3$

$i = 2$ $\beta_{23} = a_{23} - \alpha_{21}\beta_{13} = 4 - \frac{4}{9}3 = 2.6667$

$i = 3$ $\beta_{33} = a_{33} - \alpha_{31}\beta_{13} - \alpha_{32}\beta_{23} = \ldots = 6.8000$

Systems of linear equations: LU factorization

- The final results is:

$$\mathbf{L} = \begin{bmatrix} 1 & 0 & 0 \\ 0.4444 & 1 & 0 \\ 0.1111 & 0.7000 & 1 \end{bmatrix} \qquad \mathbf{U} = \begin{bmatrix} 9 & 2 & 3 \\ 0 & 1.1111 & 2.6667 \\ 0 & 0 & 6.8000 \end{bmatrix} \qquad \mathbf{A} = \begin{bmatrix} 9 & 2 & 3 \\ 4 & 2 & 4 \\ 1 & 1 & 9 \end{bmatrix}$$

- Check by Matlab:

```
>> L=[1 0 0; 0.4444 1 0; 0.1111 0.7 1.0]
L =
    1.0000         0         0
    0.4444    1.0000         0
    0.1111    0.7000    1.0000
>> U=[9 2 3; 0 1.1111 2.6667; 0 0 6.8]
U =
    9.0000    2.0000    3.0000
         0    1.1111    2.6667
         0         0    6.8000
>> L*U
ans =
    9.0000    2.0000    3.0000
    3.9996    1.9999    3.9999
    0.9999    1.0000    9.0000
```

Or using the matlab `lu`-function:

```
>> A=[9 2 3; 4 2 4; 1 1 9]
A =
     9     2     3
     4     2     4
     1     1     9
>> [L,U]=lu(A);
>> L,U
L =
    1.0000         0         0
    0.4444    1.0000         0
    0.1111    0.7000    1.0000
U =
    9.0000    2.0000    3.0000
         0    1.1111    2.6667
         0         0    6.8000
```

Systems of linear equations: LU factorization

- How about pivoting?

  - Normally partial pivoting (row interchange) is enough.

  - Row interchange can be formally written in the form

$$\mathbf{A} = \mathbf{PLU}, \quad \text{where } \mathbf{P} \text{ is the permutation matrix.}$$

   - This is the form in which the LAPACK and Matlab LU routines give their results.

Sometimes the above is written as

$$\mathbf{P'A} = \mathbf{LU}.$$

It easy to see that

$$\mathbf{P'} = \mathbf{P}^{-1} = \mathbf{P}^{\mathrm{T}}.$$

  - Permutation matrix has the form $\mathbf{P} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$, i.e. unit matrix with rows shuffled.

   - This particular example gives a matrix with the order of rows as $[2, 4, 1, 3]$ (instead of $[1, 2, 3, 4]$):

$$\mathbf{PA} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} = \begin{bmatrix} a_{21} & a_{22} & a_{23} & a_{24} \\ a_{41} & a_{42} & a_{43} & a_{44} \\ a_{11} & a_{12} & a_{13} & a_{14} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{bmatrix}$$

   - Later we give examples how to use LAPACK routines to LU factorize matrices and to use the factorization in solving linear equations.

Systems of linear equations: LU factorization

- Matlab `lu`-function:

```
>> X=rand(4)
X =
    0.2679    0.2126    0.2071    0.5751
    0.4399    0.8392    0.6072    0.4514
    0.9334    0.6288    0.6299    0.0439
    0.6833    0.1338    0.3705    0.0272
>> [L,U]=lu(X);
>> L*U
ans =
    0.2679    0.2126    0.2071    0.5751
    0.4399    0.8392    0.6072    0.4514
    0.9334    0.6288    0.6299    0.0439
    0.6833    0.1338    0.3705    0.0272
>> L,U
L =
    0.2871    0.0590    0.0832    1.0000
    0.4713    1.0000         0         0
    1.0000         0         0         0
    0.7321   -0.6015    1.0000         0
U =
    0.9334    0.6288    0.6299    0.0439
         0    0.5429    0.3103    0.4307
         0         0    0.0960    0.2542
         0         0         0    0.5160
>> [L,U,P]=lu(X);
>> P*X
ans =
    0.9334    0.6288    0.6299    0.0439
    0.4399    0.8392    0.6072    0.4514
    0.6833    0.1338    0.3705    0.0272
    0.2679    0.2126    0.2071    0.5751
```

```
 LU     LU factorization.
    [L,U] = LU(X) stores an upper triangular matrix in U and a
    "psychologically lower triangular matrix" (i.e. a product
    of lower triangular and permutation matrices) in L, so
    that X = L*U. X can be rectangular.

    [L,U,P] = LU(X) returns unit lower triangular matrix L, upper
    triangular matrix U, and permutation matrix P so that
    P*X = L*U.
```

```
>> L*U
ans =
    0.9334    0.6288    0.6299    0.0439
    0.4399    0.8392    0.6072    0.4514
    0.6833    0.1338    0.3705    0.0272
    0.2679    0.2126    0.2071    0.5751
>> L,U,P
L =
    1.0000         0         0         0
    0.4713    1.0000         0         0
    0.7321   -0.6015    1.0000         0
    0.2871    0.0590    0.0832    1.0000
U =
    0.9334    0.6288    0.6299    0.0439
         0    0.5429    0.3103    0.4307
         0         0    0.0960    0.2542
         0         0         0    0.5160
P =
     0     0     1     0
     0     1     0     0
     0     0     0     1
     1     0     0     0
```

Systems of linear equations: LU factorization

- Calculating the inverse $\mathbf{A}^{-1}$ using LU decomposition

  - Let's take an example with $N = 3$: $\mathbf{A}^{-1} = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix}$

  - Definition of the inverse $\mathbf{LUA}^{-1} = \mathbf{1}$
  - This corresponds to equations:

$$\mathbf{LU}\begin{bmatrix} b_{11} \\ b_{21} \\ b_{31} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \qquad \mathbf{LU}\begin{bmatrix} b_{12} \\ b_{22} \\ b_{32} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \qquad \mathbf{LU}\begin{bmatrix} b_{13} \\ b_{23} \\ b_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

  - When we have computed the LU factorization we only need to do N forward and backsubstitution steps to get the inverse; in Fortran using LAPACK

```
call dgetrf(...,N,A,...) ! Compute the LU factorization
do j=1,N
      b=0.0
      b(j)=1.0
      call dgerts(...,N,...,A,...,b,...) ! Solve Ax=b using LU
      Ainv(1:N,j)=b
end do
```

Systems of linear equations: error estimation, condition number

• How do inaccuracies (due to finite precision and roundoff errors) in matrix $\mathbf{A}$ show up in the results?

- Assume a perturbation $\delta\mathbf{A}$ in $\mathbf{A}$
  $\rightarrow$ error $\delta\mathbf{x}$ in $\mathbf{x}$:

$$(\mathbf{A}+\delta\mathbf{A})(\mathbf{x}+\delta\mathbf{x}) = \mathbf{b}$$

$$\delta\mathbf{x} = -\mathbf{A}^{-1}\delta\mathbf{A}(\mathbf{x}+\delta\mathbf{x})$$

(to 1st order in perturbation)

$$\|\delta\mathbf{x}\| \le \|\mathbf{A}^{-1}\|\|\delta\mathbf{A}\|\|\mathbf{x}\|$$

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} \le \|\mathbf{A}^{-1}\|\|\delta\mathbf{A}\|$$

- Define *condition number* of matrix $\mathbf{A}$:

$$\kappa(\mathbf{A}) = \|\mathbf{A}^{-1}\|\|\mathbf{A}\|$$

$$\rightarrow \quad \frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} \le \kappa(\mathbf{A})\frac{\|\delta\mathbf{A}\|}{\|\mathbf{A}\|}$$

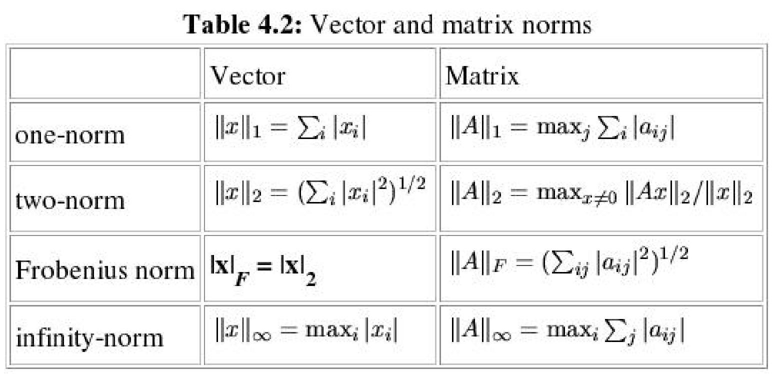


Table 4.2: Vector and matrix norms

	Vector	Matrix
one-norm	$\|x\|_1 = \sum_i \|x_i\|$	$\|A\|_1 = \max_j \sum_i \|a_{ij}\|$
two-norm	$\|x\|_2 = (\sum_i \|x_i\|^2)^{1/2}$	$\|A\|_2 = \max_{x\neq 0} \|Ax\|_2/\|x\|_2$
Frobenius norm	$\|x\|_F = \|x\|_2$	$\|A\|_F = (\sum_{ij} \|a_{ij}\|^2)^{1/2}$
infinity-norm	$\|x\|_\infty = \max_i \|x_i\|$	$\|A\|_\infty = \max_i \sum_j \|a_{ij}\|$

1) $\|\mathbf{A}\| \ge 0$
2) $\|\mathbf{A}\| = 0$, iff $\forall a_{ij} = 0$
3) $\|\alpha\mathbf{A}\| = |\alpha|\|\mathbf{A}\|$, $\alpha \in R$
4) $\|\mathbf{A}+\mathbf{B}\| \le \|\mathbf{A}\| + \|\mathbf{B}\|$
5) $\|\mathbf{AB}\| \le \|\mathbf{A}\|\|\mathbf{B}\|$

Systems of linear equations: error estimation, condition number

- This can be interpreted as:

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} \le \kappa(\mathbf{A})\frac{\|\delta\mathbf{A}\|}{\|\mathbf{A}\|}$$

$$(\text{relative error in results}) \le (\text{maximum amplification factor})(\text{relative error in matrix})$$

- Condition number measures the sensitivity of the results on perturbations in the matrix $\mathbf{A}$.
  - For ill-conditioned matrices the condition number is much larger than one.
  - LAPACK routines **XYYCON** (see below) calculate the reciprocal of the condition number:

```
anorm=dlange(norm,n,n,a,n,work)                    ! Compute ‖A‖
call dgetrf(n,n,a,n,ipiv,info1)                    ! Get LU factorization
call dgecon(norm,n,a,n,anorm,rcond,work,iwork,info2) ! Compute condition number
```

  - Function **RCOND(A)** gives the same in Matlab.
  - For well-conditioned matrices **RCOND(A)** ~ 1 and for ill-conditioned **RCOND(A)** $\sim \varepsilon$ (ε is the machine epsilon).

- For perturbations in RHS vector $\mathbf{b}$: $\mathbf{A}(\mathbf{x}+\delta\mathbf{x}) = \mathbf{b}+\delta\mathbf{b} \rightarrow \delta\mathbf{x} = \mathbf{A}^{-1}\delta\mathbf{b} \rightarrow \|\delta\mathbf{x}\| \le \|\mathbf{A}^{-1}\|\|\mathbf{b}\| \rightarrow$

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} \le \kappa(\mathbf{A})\frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|}.$$

Systems of linear equations: error estimation, condition number

- Examples with Matlab:

```
>> a=rand(5)
a =
    0.6418    0.0582    0.0748    0.9885    0.1288
    0.1785    0.5876    0.3100    0.6916    0.6868
    0.5294    0.4161    0.9441    0.2417    0.2972
    0.2187    0.1864    0.9807    0.8098    0.6472
    0.5481    0.0639    0.5551    0.9345    0.4638
>> [u1,s1,v1]=svd(a);
>> r1=rcond(a);
>> a(:,1)=a(:,2);
>> [u2,s2,v2]=svd(a);
>> r2=rcond(a);
>> a(1,1)=1.001*a(1,1);
>> [u3,s3,v3]=svd(a);
>> r3=rcond(a);
>> r1,r2,r3
r1 =        0.0146964338747443
r2 =        1.8740983020316e-18
r3 =        3.103933272459e-06
```

```
>> s1,s2,s3
s1 =
       2.5858            0            0            0            0
            0      0.93008            0            0            0
            0            0      0.62295            0            0
            0            0            0      0.44952            0
            0            0            0            0     0.087944
s2 =
       2.4969            0            0            0            0
            0      0.93435            0            0            0
            0            0      0.67109            0            0
            0            0            0      0.28254            0
            0            0            0            0    2.492e-17
s3 =
       2.4969            0            0            0            0
            0      0.93434            0            0            0
            0            0      0.67109            0            0
            0            0            0      0.28255            0
            0            0            0            0   1.2098e-05
```

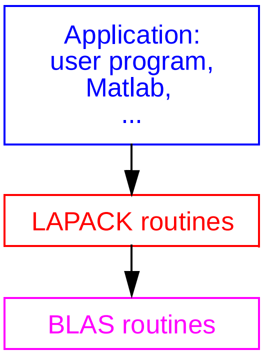
```
>> help rcond
 RCOND  LAPACK reciprocal condition estimator.
    RCOND(X) is an estimate for the reciprocal
     of the condition of X in the 1-norm obtained
    by the LAPACK condition estimator. If X is well
    conditioned, RCOND(X) is near 1.0. If X is badly
    conditioned, RCOND(X) is near EPS.
```

```
>> help svd
 SVD    Singular value decomposition.
    [U,S,V] = SVD(X) produces a diagonal matrix S, of the same
     dimension as X and with nonnegative diagonal elements in
     decreasing order, and unitary matrices U and V so that
     X = U*S*V'.
```

- Here we have used the singular value decomposition of a matrix. Zero elements in the diagonal matrix **S** tell that the matrix is singular.

Systems of linear equations: LAPACK

• A package of subroutines performing the most common linear algebra task.



- BLAS (Basic Linear Algebra Subroutines) performs the low-level matrix operations
  - Often hardware vendors provide highly optimized BLAS routines for their platform.

- Source can be downloaded from e.g. NETLIB: http://www.netlib.org/lapack/

  - Many Linux distributions have precompiled packages.
  - You can also compile it yourself.

- User's quide in HTML in http://www.netlib.org/lapack/lug/

- An easily installable LAPACK package on the course web page:
  http://www.physics.helsinki.fi/courses/s/tl3/progs/lapack/lapackf90.tgz

Systems of linear equations: LAPACK

- Naming scheme of the routines

  - All driver and computational routines have names of the form **XYYZZZ**

    - **X** indicates the data type:

S	real
D	double (precision)
C	complex
Z	double complex

- **YY** indicates the type of matrix

BD	bidiagonal	**PO**	symmetric or Hermitian positive definite
DI	diagonal	**PP**	symmetric or Hermitian positive definite, packed storage
GB	general band	**PT**	symmetric or Hermitian positive definite tridiagonal
GE	general (i.e., unsymmetric, in some cases rectangular)	**SB**	(real) symmetric band
GG	general matrices, generalized problem	**SP**	symmetric, packed storage
GT	general tridiagonal	**ST**	(real) symmetric tridiagonal
HB	(complex) Hermitian band	**SY**	symmetric
HE	(complex) Hermitian	**TB**	triangular band
HG	upper Hessenberg matrix, generalized problem	**TG**	triangular matrices, generalized problem
HP	(complex) Hermitian, packed storage	**TP**	triangular, packed storage
HS	upper Hessenberg	**TR**	triangular (or in some cases quasi-triangular)
OP	(real) orthogonal, packed storage	**TZ**	trapezoidal
OR	(real) orthogonal	**UN**	(complex) unitary
PB	symmetric or Hermitian positive definite band	**UP**	(complex) unitary, packed storage

Systems of linear equations: LAPACK

- **ZZZ** indicates the computation performed; for example:

SV	solve linear equation
SVX	solve linear equation (*expert* version)
TRF	factorize
TRS	use the factorization to solve linear equations
CON	estimate the reciprocal of the condition number
TRI	use the factorization to compute inverse of a matrix
EV	determine eigenvalues

- For example:

DGESV	compute the solution to a double system of linear equations for general matrices
ZGEEV	compute for an N-by-N complex nonsymmetric matrix the eigenvalues
DGETRF	compute an LU factorization of a general M-by-N matrix

Systems of linear equations: LAPACK

- Using LAPACK in Kumpula Linux (**punk, mutteri,** etc.) system:

```
gfortran test.f90 -llapack
gcc test.c -llapack -lm
```

- All routines have individual man pages on **punk** and **mutteri**:

```
# man dgesv

DGESV(l)                                                                    DGESV(l)
NAME
       DGESV - compute the solution to a real system of linear equations A * X = B,
SYNOPSIS
       SUBROUTINE DGESV( N, NRHS, A, LDA, IPIV, B, LDB, INFO )
           INTEGER       INFO, LDA, LDB, N, NRHS
           INTEGER       IPIV( * )
           DOUBLE        PRECISION A( LDA, * ), B( LDB, * )
PURPOSE
       DGESV  computes the solution to a real system of linear equations A * X = B,
      where A is an N-by-N matrix and X and B
. . .
```

Systems of linear equations: LAPACK

- A typical subroutine interface (Fortran90):

```
SUBROUTINE DGETRF( M, N, A, LDA, PIV, INFO )
     INTEGER       INFO, LDA, M, N
     INTEGER       PIV( * )
     DOUBLE        PRECISION A( LDA, * )

PURPOSE
DGETRF computes an LU factorization of a general
     M-by-N matrix A using partial pivoting with
     row interchanges.
ARGUMENTS
M    (input) INTEGER
     The number of rows of the matrix A.  M >= 0.
N    (input) INTEGER
     The number of columns of the matrix A.  N >= 0.
A    (input/output) DOUBLE PRECISION array,
     dimension (LDA,N)
     On entry, the M-by-N matrix to be factored.
     On exit, the factors L
     and U from the factorization A = P*L*U;
     the unit diagonal elements of L are not stored.
LDA  (input) INTEGER
     The leading dimension of the array A.  LDA >= max(1,M).
PIV  (output) INTEGER array, dimension (min(M,N))
     The pivot indices; for 1 <= i <= min(M,N), row i of the matrix was
     interchanged with row IPIV(i).
INFO (output) INTEGER
     = 0:  successful exit
     < 0:  if INFO = -i, the i-th argument had an illegal value
     > 0:  if INFO = i, U(i,i) is exactly zero. The factorization has
     been completed, but the factor U is exactly singular, and division
     by zero will occur if it is used to solve a system of equations.
```

```
A typical subroutine call when
dealing with square matrices:

real :: a(n,n)
integer :: pivot(n),ok

. . .

call dgetrf(n,n,a,n,pivot,ok)
```

Systems of linear equations: LAPACK

- Calling LAPACK routines from C

  - LAPACK written in Fortran 77 :-(

  - *Things to remember when using both Fortran and C in the same program:*

    1. Fortran routine names have usually and underscore appended: in Fortran **`subr`**, in C **`subr_`**

    2. In Fortran all subroutine parameters passed *by reference*. This means that in C they have to be pointers.

Main program in Fortran

```
program fmain
  integer :: i
  i=1
  call csub(i)
  print '(a,i2)','i=',i
end program fmain

- - - - - - - - - - - - - - -

void csub_(int *i) {
  (*i)++;
}

- - - - - - - - - - - - - - -

c_f90> gfortran -c fmain.f90
c_f90> gcc -c csub.c
c_f90> gfortran fmain.o csub.o
c_f90> a.out
i= 2
```

Main program in C

```
#include <stdio.h>
int main() {
  int i;
  i=1;
  fsub_(&i);
  printf("i=%d\n",i);
  return 0;
}
- - - - - - - - - - - - - - -
subroutine fsub(i)
  integer :: i
  i=i+1
  return
end subroutine fsub
- - - - - - - - - - - - - - -
c_f90> gcc -c cmain.c
c_f90> gfortran -c fsub.f90
c_f90> gcc cmain.o fsub.o
c_f90> a.out
i=2
```

Systems of linear equations: LAPACK

- *C-Fortran continued*

  3. The order of multidimensional array elements in memory is different in Fortran and C.
     E.g. a 3×3 matrix:
     `double A[3][3]` or
     `real(kind=8) :: A(3,3).`

     In C the last index changes fastest, in Fortran the first:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

order of elements in memory:

C: $\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{21} & a_{22} & a_{23} & a_{31} & a_{32} & a_{33} \end{bmatrix}$

Fortran: $\begin{bmatrix} a_{11} & a_{21} & a_{31} & a_{12} & a_{22} & a_{32} & a_{13} & a_{23} & a_{33} \end{bmatrix}$

Consequently, you have to transpose the matrix before calling the Fortran routine.
(Or in C use one-dimensional arrays and take care of index arithmetics yourself.)

Systems of linear equations: LAPACK

- Example for calling LAPACK routines from C

Fortran:

```
integer,parameter :: rk=8
real(rk),allocatable :: a(:,:),b(:),x(:)
integer :: n,ok
integer,allocatable :: pivot(:)
. . .
allocate(a(n,n),b(n),pivot(n))
. . .
do i=1,n
   read(5,*) (a(i,j),j=1,n)
enddo
. . .
call dgesv(n, 1, a, n, pivot, b, n, ok)
```

C:

```
int n,i,j,c1,c2,*pivot,ok;
double *A,*b;
. . .
A=(double*)malloc((size_t)n*n*sizeof(double));
b=(double *)malloc((size_t)n*sizeof(double));
pivot=(int *)malloc((size_t)n*sizeof(int));
. . .
for (i=0;i<n;i++)
  for (j=0;j<n;j++)
    scanf("%lg",&A[j*n+i]);
. . .
c1=n;c2=1;
dgesv_(&c1, &c2, A, &c1, pivot, b, &c1, &ok);
```

Note that we use a 1D array instead of a 2D one and store (actually read in) it in the right (Fortran-like) order.

Awkward? Yes!!

- Now a few practical examples[1].

1. See http://www.physics.helsinki.fi/courses/s/tl3/progs/lapack/

Systems of linear equations: GSL

- Gnu Scientific Library (GSL) also includes linear algebra routines.
  - However, they are recommended to be used *only with small systems*.
  - Below is an simple example[1] from the GSL `info` page[2]:

```
#include <stdio.h>
#include <gsl/gsl_linalg.h>

int main (void)
{
  double a_data[] = { 0.18, 0.60, 0.57, 0.96,
                      0.41, 0.24, 0.99, 0.58,
                      0.14, 0.30, 0.97, 0.66,
                      0.51, 0.13, 0.19, 0.85 };

  double b_data[] = { 1.0, 2.0, 3.0, 4.0 };

  gsl_matrix_view m=gsl_matrix_view_array(a_data, 4, 4);
  gsl_vector_view b=gsl_vector_view_array(b_data, 4);
  gsl_vector *x=gsl_vector_alloc (4);
  int s;
  gsl_permutation *p=gsl_permutation_alloc(4);

  gsl_linalg_LU_decomp(&m.matrix, p, &s);

  gsl_linalg_LU_solve(&m.matrix, p, &b.vector, x);

  printf ("x = \n");
  gsl_vector_fprintf(stdout, x, "%g");

  gsl_permutation_free(p);
  return 0;
}
```

```
linearalgebra> gcc lusolve_gsl.c -lgsl -lgslcblas -lm
linearalgebra> a.out
x =
-4.05205
-12.6056
1.66091
8.69377
```

Note that GSL uses its own data types to present vectors and matrices. These are essentially `structs` but should be accesses by using the functions provided by the library.

1. http://www.physics.helsinki.fi/courses/s/tl3/progs/gsl/linearalgebra/
2. To access the `info` help system on **punk** or **mutteri** give the command `info gsl` or use the `info` pages in GNU Emacs or Xemacs.

Systems of linear equations: other factorizations

• LU factorization assumes nothing about the matrix

- If $\mathbf{A}$ has some special properties (e.g. symmetries) there are faster[1] and more stable factorizations

- Symmetric matrices: **$\mathbf{LDL}^T$ factorization**:

  $\mathbf{A} = \mathbf{LDL}^T$, where $\mathbf{L}$ is a lower triangular matrix with unit diagonal and $\mathbf{D}$ is a diagonal matrix

- Positive definite matrices ($\mathbf{x}^T\mathbf{A}\mathbf{x} > 0, \quad \forall\mathbf{x}$, or equivalently all eigenvalues[2] are positive): **Cholesky factorization**:

  $\mathbf{A} = \mathbf{LL}^T$, where $\mathbf{L}$ is a lower triangular matrix

  - LAPACK uses Cholesky factorization for positive definite matrices in routines *X***PO**SV, *X***PO**TRF, ...
  - Matlab has the function `chol`:

```
>> help chol
 CHOL   Cholesky factorization.
    CHOL(X) uses only the diagonal and upper triangle of X.
    The lower triangular is assumed to be the (complex conjugate)
    transpose of the upper.  If X is positive definite, then
    R = CHOL(X) produces an upper triangular R so that R'*R = X.
    If X is not positive definite, an error message is printed.
```

1. Though they still behave as $O(N^3)$.
2. We will talk about eigenvalues and eigenvectors later.

Systems of linear equations: other factorizations

- Permutation matrix

$$\mathbf{P} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$ is a unit matrix with rows interchanged.

- In practice stored as a vector: $\mathbf{p} = [2, 4, 1, 3]$

- Shuffle rows: $\mathbf{A} \leftarrow \mathbf{PA}$
- Shuffle columns: $\mathbf{A} \leftarrow \mathbf{AP}$

- To preserve symmetry use $\mathbf{A} \leftarrow \mathbf{PAP}^T$

Systems of linear equations: iterative methods

• In the case of very large systems of linear equations roundoff errors sometimes prevent using the direct methods described above.

- One has to resort to iterative metods.

• Let $\mathbf{x}$ be the solution of the equation

$$\mathbf{A}\mathbf{x} = \mathbf{b}$$

- We don't know its exact value but an approximation $\mathbf{x} + \delta\mathbf{x}$ where $\delta\mathbf{x}$ is an unknown error.
- Insert it to the equation:

$$\mathbf{A}(\mathbf{x} + \delta\mathbf{x}) = \mathbf{b} + \delta\mathbf{b} \qquad (1)$$

- Subtract the two equations:

$$\mathbf{A}\delta\mathbf{x} = \delta\mathbf{b} \qquad (2)$$

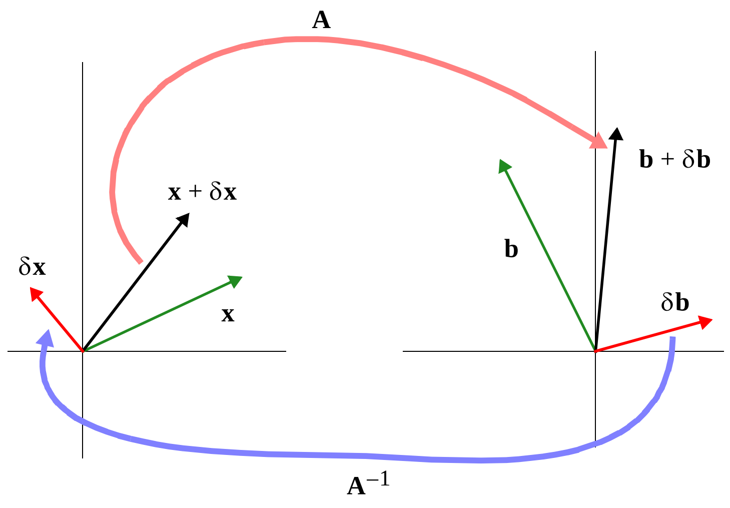
- Solving $\delta\mathbf{b}$ from (1) and substituting it in (2) we get

$$\mathbf{A}\delta\mathbf{x} = \mathbf{A}(\mathbf{x} + \delta\mathbf{x}) - \mathbf{b} \qquad (3)$$

- Because RHS of (3) is known we can solve $\delta\mathbf{x}$ from it
  $\rightarrow$ subtract the obtained $\delta\mathbf{x}$ from the current solution
  $\rightarrow$ a better approximation to the true solution.

Systems of linear equations: iterative methods

- Graphically:



- One way of using these methods is to first find a solution by direct methods and then improve the approximation by a few iterations.

Systems of linear equations: iterative methods

- We assumed solution has error $\delta\mathbf{x}$

- However, also $\mathbf{A}^{-1}$ has errors: let $\mathbf{B}_0$ be its approximation, and residual matrix $\mathbf{R}$

$$\mathbf{R} \equiv \mathbf{1} - \mathbf{B}_0\mathbf{A} \qquad \rightarrow \qquad \mathbf{B}_0\mathbf{A} = \mathbf{1} - \mathbf{R}$$

- Now a few steps of matrix algebra gives:

$$\mathbf{A}^{-1} = \mathbf{A}^{-1}(\mathbf{B}_0^{-1}\mathbf{B}_0) = (\mathbf{A}^{-1}\mathbf{B}_0^{-1})\mathbf{B}_0 = (\mathbf{B}_0\mathbf{A})^{-1}\mathbf{B}_0 = (1-\mathbf{R})^{-1}\mathbf{B}_0 = (1+\mathbf{R}+\mathbf{R}^2+\mathbf{R}^3+\ldots)\mathbf{B}_0$$

- Let's denote the truncated sum of the previous expression as

$$\mathbf{B}_n = (1+\mathbf{R}+\mathbf{R}^2+\mathbf{R}^3+\ldots+\mathbf{R}^n)\mathbf{B}_0 \qquad (4)$$

$$\lim_{n\to\infty} \mathbf{B}_n = \mathbf{A}^{-1}$$

- Further define

$$\mathbf{x}_n \equiv \mathbf{B}_n\mathbf{b}$$

- Based on all this it is easy to show that (4) satisfies the following recurrence relation

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \mathbf{B}_0(\mathbf{b} - \mathbf{A}\mathbf{x}_n)$$

- This is exactly equation (3) when we write $-\delta\mathbf{x} = \mathbf{x}_{n+1} - \mathbf{x}_n$ and $\mathbf{A}^{-1}$ as $\mathbf{B}_0$ ($\rightarrow -\delta\mathbf{x} = \mathbf{A}^{-1}(\mathbf{b} - \mathbf{A}\mathbf{x}_n)$).

- This means that the LU factorization of $\mathbf{A}$ need not be exact but only the residual has to be 'small'; i.e. $\|\mathbf{R}\| < 1$.

Systems of linear equations: iterative methods

- Let's take a simple example: the Gauss-Jacobi method
  - Equation $\mathbf{A}\mathbf{x} = \mathbf{b}$ can be written as

$$x_i = \frac{1}{a_{ii}}\left[b_i - \sum_{j=1,(j\neq i)}^{N} a_{ij}x_j\right], \quad i = 1, 2, \ldots, N.$$

  - Now, assuming $a_{ii} \neq 0 \ \forall i$ we can solve vector $\mathbf{x}$ by iteration

$$x_i^{(m+1)} = \frac{1}{a_{ii}}\left[b_i - \sum_{j=1,(j\neq i)}^{N} a_{ij}x_j^{(m)}\right], \quad i = 1, 2, \ldots, N.$$

  - How about convergence of the iteration? One can show that[1] if

$r_\sigma(\mathbf{M}) < 1$, where $\mathbf{M} = \begin{bmatrix} 0 & a_{12}/a_{11} & \ldots & a_{1N}/a_{11} \\ a_{21}/a_{22} & 0 & \ldots & a_{2N}/a_{22} \\ \ldots & \ldots & \ldots & \ldots \\ a_{N1}/a_{NN} & \ldots & \ldots & 0 \end{bmatrix}$ and $r_\sigma(\mathbf{M}) = \max_{\lambda\in\sigma(\mathbf{M})}|\lambda|$, $\sigma(\mathbf{M})$ is the set of $\mathbf{M}$'s eigen-

values and if $\mathbf{A}$ is **strictly diagonally dominant**: $\sum_{j=1,(j\neq i)}^{N} |a_{ij}| < |a_{ii}|$, $i = 1, 2, \ldots, N$

then $\lim_{m\to\infty} \mathbf{x}^{(m)} = \mathbf{x}$.

1. K.E.Atkinson: *An Introduction to Numerical Analysis*, paragraph 8.6

Systems of linear equations: iterative methods

- Generally in an iterative method to solve $\mathbf{Ax} = \mathbf{b}$ the matrix is split $\mathbf{A} = \mathbf{N} - \mathbf{P}$, so that the equation can be written as

$$\mathbf{Nx} = \mathbf{b} + \mathbf{Px}$$

  - Matrix $\mathbf{N}$ is chosen such that equation $\mathbf{Ny} = \mathbf{c}$ is easy to solve
    - I.e. $\mathbf{N}$ is diagonal, triangular, etc.

  - The iteration to solve the original equation is now

$$\mathbf{Nx}^{(m+1)} = \mathbf{b} + \mathbf{Px}^{(m)},\ m \geq 0,\quad \text{with } \mathbf{x}^{(0)} \text{ given (guessed).}$$

  - One can show that the method converges for any $\mathbf{x}^{(0)}$ if

$$r_\sigma(\mathbf{M}) < 1, \text{ where } \mathbf{M} = \mathbf{N}^{-1}\mathbf{P}.$$

  - A variation of the Gauss-Jacobi method is Gauss-Seidel:

$$x_i^{(m+1)} = \frac{1}{a_{ii}}\left[b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(m+1)} - \sum_{j=i+1}^{N} a_{ij}x_j^{(m)}\right],\ i = 1, 2, \ldots, N$$

  - This can be cast into matrix form as

$$\mathbf{N} = \begin{bmatrix} a_{11} & 0 & 0 & \ldots & 0 \\ a_{21} & a_{22} & 0 & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots & 0 \\ a_{N1} & \ldots & \ldots & \ldots & a_{NN} \end{bmatrix}$$

Systems of linear equations: iterative methods

• Some iterative methods are based on optimization methods.

- Conjugate gradient (CG) method finds a minimum of a scalar function $f(\mathbf{x})$; $\mathbf{x} = [x_1, x_2, \ldots, x_N]^{\mathrm{T}}$
- CG is exact (minimum found in$N$ steps) for quadratic functions
- The method is derived by approximating $f(\mathbf{x})$ by a quadratic form

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{Ax} - \mathbf{b}^{\mathrm{T}}\mathbf{x} + \mathbf{c}\ ,$$

where $\mathbf{A}$ is so called Hessian matrix (positive definite: $\mathbf{x}^{\mathrm{T}}\mathbf{Ax} > 0$, $\forall\mathbf{x}$)

- The minimum is found when the gradient is zero:

$$\nabla f(\mathbf{x}) = 0 \rightarrow \quad \nabla\left(\frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{Ax} - \mathbf{b}^{\mathrm{T}}\mathbf{x} + \mathbf{c}\right) = 0 \rightarrow \mathbf{Ax} - \mathbf{b} = 0$$

- So by minimizing the function $f(\mathbf{x})$ we find the solution to the group of linear equations $\mathbf{Ax} = \mathbf{b}$.

- The principle of the CG methods is to start with some initial value of $\mathbf{x}$ and iterate so that the new ‘direction’ is found so that it minimizes the function in that direction but does not spoil the minimzations of previous iterated directions:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k\mathbf{p}_k \text{ , where } \alpha_k \text{ is chosen so that } f(\mathbf{x}_{k+1}) \text{ is minimized, and } \mathbf{p}_i^{\mathrm{T}}\mathbf{Ap}_j = 0 \text{, when } i \neq j.$$

(We will talk more about CG when dealing with function minimization.)

- There is a wide variety of iterative methods and in many cases the methods are specific to the problem at hand.

Systems of linear equations: sparse matrices

- In many application the matrix $\mathbf{A}$ is large but has only few non-zero elements; it is sparse.
  - In these cases abovementioned direct methods are inefficient
    - Most arithmetic operations are performed on zeroes.
    - Memory is wasted in storing zeroes.
  - However, in some cases LU factorization is straightforward for sparse matrices:
    - LU factorization of a *tridiagonal* system is trivial

$$\begin{bmatrix} b_1 & c_1 & 0 & \dots & & & \\ a_2 & b_2 & c_2 & \dots & & & \\ & & & \dots & & & \\ & & & \dots & a_{N-1} & b_{N-1} & c_{N-1} \\ & & & & 0 & a_N & b_N \end{bmatrix} \cdot \begin{bmatrix} u_1 \\ u_2 \\ \dots \\ u_{N-1} \\ u_N \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ \dots \\ r_{N-1} \\ r_N \end{bmatrix},$$

i.e. if $|i-j| > 1 \rightarrow a_{ij} = 0$.

```
void tridag(double a[], double b[], double c[], double r[], double u[], unsigned long n)
    {
          unsigned long j;
          double bet,*gam;

          gam=malloc(sizeof(double)*n);
          bet=b[0];
          u[0]=r[0]/b[0];
          for (j=1;j<n;j++) {
             gam[j]=c[j-1]/bet;
             bet=b[j]-a[j]*gam[j];
             u[j]=(r[j]-a[j]*u[j-1])/bet;
          }
          for (j=n-2;j>1;j--)
             u[j] -= gam[j+1]*u[j+1];
          free(gam);
    }
```

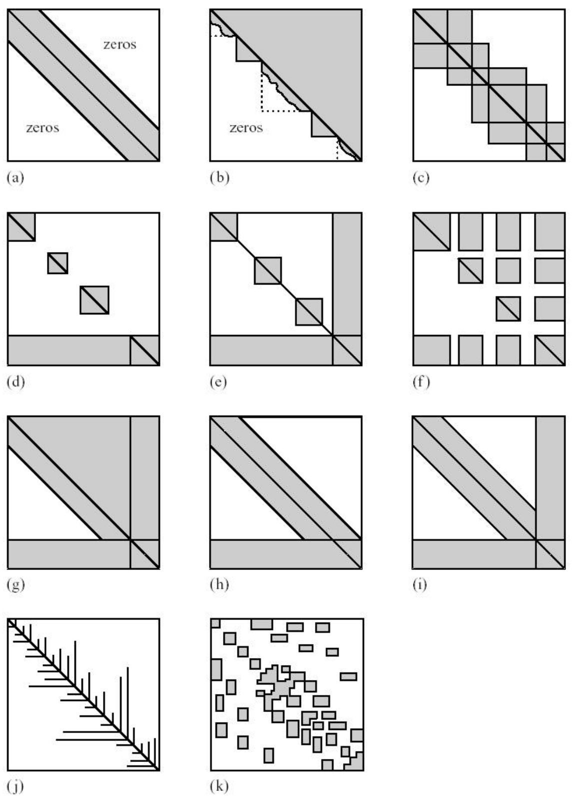
Matrix elements a_i, b_i, c_i stored in arrays **`a[]`**, **`b[]`**, **`c[]`**.

Systems of linear equations: sparse matrices

- Many kinds of sparsity:
  - During the computation there is the possibility that the number of nonzero elements increases: fill-in
  - This may be minimized by clever enumeration of variables.
  - Moreover, it may be wise to enumarate the variables initially in such a way that the resulting matrix has a band structure:

$$a_{ij} > 0 \text{ if } |i-j| < \text{a positive constant}$$

  - For sparse matrix operation there are libraries (NAG, IMSL) and Matlab has a toolbox for sparse matrices.
  - Iterative methods are commonly used with large sparse matrices.
  - Subroutine library for solving large sparse matrices by iterative methods: **ITPACK**
    www.netlib.org/itpack



W.H. Press et al., Numerical Recipes, Fig. 2.7.1