# Lecture 11

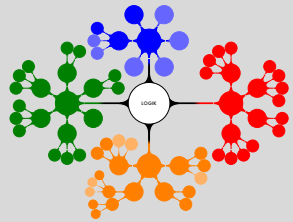
# Numerical Techniques: Solving Sets of Equations - Linear Algebra and Matrix Computing II

## Normal Modes of Coupled Oscillation

IKC-MH.55 *Scientific Computing with Python* at January 05, 2024

Dr. Cem Özdoğan
Engineering Sciences Department
İzmir Kâtip Çelebi University

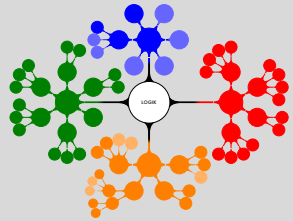




# Contents







# Solving Sets of Equations

- Solving sets of linear equations and eigenvalue problems are the most frequently used numerical procedures when real-world situations are modelled.

## 1 Matrices and Vectors

## 2 Elimination Methods

Continued.

## 3 The Inverse of a Matrix

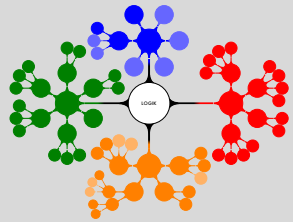
Shows how an important derivative of a matrix, its inverse, can be computed. It shows when a matrix **cannot be inverted** and tells of situations where **no unique solution** exists to a system of equations.

## 4 Iterative Methods

It is described how a linear system can be solved in an entirely different way, by beginning with an initial estimate of the solution.



Dr. Cem Özdoğan



# Gaussian Elimination XIII

Continue with the previous example.

- If we had replaced the zeros below the main diagonal with the ratio of coefficients at each step, the resulting augmented matrix would be

$$\begin{bmatrix} 6 & 1 & -6 & -5 & 6 \\ (0.66667) & -3.6667 & 4 & 4.3333 & -11 \\ (0.33333) & (-0.45454) & 6.8182 & 5.6364 & -9.0001 \\ (0.0) & (-0.54545) & (0.32) & 1.5600 & -3.1199 \end{bmatrix}$$

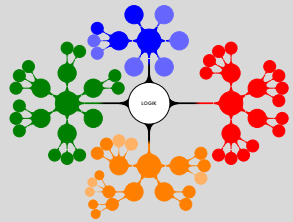
- This gives a $LU$ decomposition as

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.66667 & 1 & 0 & 0 \\ 0.33333 & -0.45454 & 1 & 0 \\ 0.0 & -0.54545 & 0.32 & 1 \end{bmatrix} \begin{bmatrix} 6 & 1 & -6 & -5 \\ 0 & -3.6667 & 4 & 4.3333 \\ 0 & 0 & 6.8182 & 5.6364 \\ 0 & 0 & 0 & 1.5600 \end{bmatrix}$$

- It should be noted that the product of these matrices produces a permutation of the original matrix, call it $A'$, where

$$A' = \begin{bmatrix} 6 & 1 & -6 & -5 \\ 4 & -3 & 0 & 1 \\ 2 & 2 & 3 & 2 \\ 0 & 2 & 0 & 1 \end{bmatrix}$$





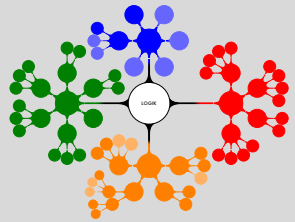
# Gaussian Elimination XIV

- The determinant of the original matrix of coefficients can be easily computed according to the formula

$$det(A) = (-1)^2 * (6) * (-3.6667) * (6.8182) * (1.5600) = -234.0028$$

  which is close to the exact solution: -234.
- The exponent 2 is required, because there were *two row interchanges* in solving this system.
- To summarize
  1. The solution to the four equations
  2. The determinant of the coefficient matrix
  3. A $LU$ decomposition of the matrix, $A'$, which is just the original matrix, $A$, after we have interchanged its rows.
- **"These" are readily obtained after solving the system by Gaussian elimination method.**





# Gaussian Elimination XV

**Example py-files:** LU factorization without pivoting.
myLUshow.py LU factorization with pivoting.
myLUPivShow.py

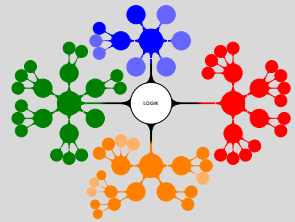
```
Tolerance value in pivoting is 1.110223e-15.
LU decomposition of the system:
[[ 0.  2.  0.  1.]
 [ 2.  2.  3.  2.]
 [ 4. -3.  0.  1.]
 [ 6.  1. -6. -5.]]
Zero Pivot Encountered. Exiting.
```

```
Tolerance value in pivoting is 1.110223e-15.
LU decomposition of the system:
[[ 0.  2.  0.  1.]
 [ 2.  2.  3.  2.]
 [ 4. -3.  0.  1.]
 [ 6.  1. -6. -5.]]
Swap rows 0 and 3;  new pivot = 6.000000
Swap rows 1 and 2;  new pivot = -3.666667
MyLUshow - Lower Triangular :
 [[ 1.          0.          0.          0.        ]
 [ 0.66666667  1.          0.          0.        ]
 [ 0.33333333 -0.45454545  1.          0.        ]
 [ 0.         -0.54545455  0.32        1.        ]]
MyLUshow - Upper Triangular :
 [[ 6.          1.         -6.         -5.        ]
 [ 0.         -3.66666667  4.          4.33333333]
 [ 0.          0.          6.81818182  5.63636364]
 [ 0.          0.          0.          1.56      ]]
SciPy LU-decomposition: PL - Permutation Matrix, Lower
 [[ 0.         -0.54545455  0.32        1.        ]
 [ 0.33333333 -0.45454545  1.          0.        ]
 [ 0.66666667  1.          0.          0.        ]
 [ 1.          0.          0.          0.        ]]
SciPy LU-decomposition: U - Upper Triangular
 [[ 6.          1.         -6.         -5.        ]
 [ 0.         -3.66666667  4.          4.33333333]
 [ 0.          0.          6.81818182  5.63636364]
 [ 0.          0.          0.          1.56      ]]
```

**Figure:** (a) Without Pivoting (b) With Pivoting.



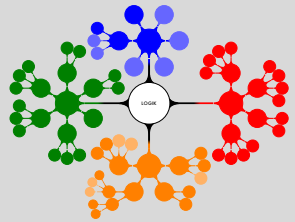
Dr. Cem Özdoğan



# Using the LU Matrix for Multiple Right-Hand Sides I

- Many physical situations are modelled with a large set of linear equations.
- The equations will depend on the geometry and certain external factors that will determine the right-hand sides.
- For example, in electrical circuit problems, the resistors at the circuit ($A$ matrix) are unchanged with the varying applied voltages ($b$ vector). (e.g., Kirchhoff's Rule)
- If we want the solution for **many different values of these right-hand sides**,
  - it is inefficient to solve the system from the start with each one of the right-hand-side values.
  - Using the $LU$ equivalent of the coefficient matrix is preferred.
- Suppose we have solved the system $Ax = b$ by Gaussian elimination.
- We now know the $LU$ equivalent of $A$: $A = L * U$





# Using the LU Matrix for Multiple Right-Hand Sides II

- We can write

$$Ax = b$$

$$LUx = b$$

$$Ly = b$$

- e.g., Solve $Ax = b$, where we already have its $L$ and $U$ matrices:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.66667 & 1 & 0 & 0 \\ 0.33333 & -0.45454 & 1 & 0 \\ 0.0 & -0.54545 & 0.32 & 1 \end{bmatrix} * \begin{bmatrix} 6 & 1 & -6 & -5 \\ 0 & -3.6667 & 4 & 4.3333 \\ 0 & 0 & 6.8182 & 5.6364 \\ 0 & 0 & 0 & 1.5600 \end{bmatrix}$$

- Suppose that the $b$-vector is $[6 \ -7 \ -2 \ 0]^T$.
- We first get $y(= Ux)$ from $Ly = b$ by forward substitution:

$$y = [6 \ -11 \ -9 \ -3.12]^T$$

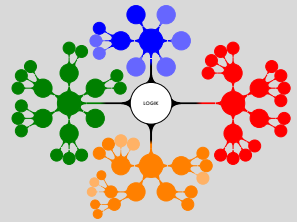
- and use it to compute $x$ from $Ux = y$:

$$x = [-0.5 \ 1 \ 0.3333 \ -2]^T.$$

- Exercise: $b = [1 \ 4 \ -3 \ 1]^T \Longrightarrow$
  $x = [0.0128 \ -0.5897 \ -2.0684 \ 2.1795]^T$



Dr. Cem Özdoğan



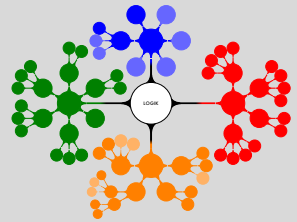
# Using the LU Matrix for Multiple Right-Hand Sides III

## Phyton Code:

```
1  import numpy as np
2  from scipy.linalg import lu
3  A = np.array([[0.0, 2.0, 0.0, 1.0], [2.0, 2.0, 3.0, 2.0], [4.0, -3.0, 0.0, 1.0], [6.0, 1.0,
       -6.0, -5.0]])
4  P, L, U = lu(A)
5  print("SciPy LU-decomposition: P - Permutation Matrix \n", P)
6  print("SciPy LU-decomposition: L - Lower Triangular with unit diagonal elements \n", L)
7  print("SciPy LU-decomposition: U - Upper Triangular \n", U)
8  def forward(L, b):
9      y=np.zeros(np.shape(b),dtype=float)
10     for i in range(len(b)):
11         y[i]=np.copy(b[i])
12         for j in range(i):
13             y[i]=y[i]-(L[i, j]*y[j])
14         y[i] = y[i]/L[i, i]
15     return y
16 b = np.array([[6.0], [-7.0], [-2.0], [0.0]])
17 # b = np.array([[1.0], [4.0], [-3.0], [1.0]])
18 y=forward(L,b)
19 print("y vector from Ly=b by forward substitution :", np.transpose(y))
20 def backward(U, y):
21     x=np.zeros(np.shape(y),dtype=float)
22     ylen=len(y)-1
23     x[ylen] =y[ylen]/U[ylen, ylen] # Print the last stage x value
24     for i in range(ylen-1,-1,-1):
25         x[i]=np.copy(y[i])
26         for j in range(ylen,i,-1):
27             x[i]=x[i]-(U[i, j]*x[j])
28         x[i] = x[i]/U[i, i]
29     return x
30 x=backward(U,y)
31 print("x vector from Ux=y by backward substitution :", np.transpose(x))
```



Dr. Cem Özdoğan



# The Inverse of a Matrix I

- Division by a matrix is not defined but the equivalent is obtained from the ***inverse*** of the matrix.
- If the product of two square matrices, $A * B$, equals to the *identity matrix*, $I$, $B$ is said to be the inverse of $A$ (and also $A$ is the inverse of $B$).
- By multiplying each element with its cofactor to find the inverse of the matrix is not useful since $N^3$ multiplication and division are required for an N-dimensional matrix.
- To find the inverse of matrix $A$, use an elimination method.
- We augment the $A$ matrix with the identity matrix of the same size and solve. **The solution is $A^{-1}$.** Example;

$$A = \begin{bmatrix} 1 & -1 & 2 \\ 3 & 0 & 1 \\ 1 & 0 & 2 \end{bmatrix} \rightarrow \left[\begin{array}{ccc|ccc} 1 & -1 & 2 & 1 & 0 & 0 \\ 3 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 2 & 0 & 0 & 1 \end{array}\right] \begin{array}{l} \\ R_2 - (3/1)R_1 \rightarrow \\ R_3 - (1/1)R_1 \rightarrow \end{array}$$

$$\left[\begin{array}{ccc|ccc} 1 & -1 & 2 & 1 & 0 & 0 \\ 0 & 3 & -5 & -3 & 1 & 0 \\ 0 & 1 & 0 & -1 & 0 & 1 \end{array}\right] \underbrace{\left[\begin{array}{ccc|ccc} 1 & -1 & 2 & 1 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 1 \\ 0 & 3 & -5 & -3 & 1 & 0 \end{array}\right]}_{\text{Row Interchange}} R_3 - (3/1)R_2 \rightarrow$$

# The Inverse of a Matrix II

- Contd.

$$\left[\begin{array}{cccccc} 1 & -1 & 2 & 1 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 1 \\ 0 & 0 & -5 & 0 & 1 & -3 \end{array}\right] \quad \left\| \begin{array}{l} \\ R_3/(-5) \rightarrow \end{array} \right\| \quad \left\| \begin{array}{l} \\ R_1 - (2/1)R_3 \rightarrow \end{array} \right\|$$

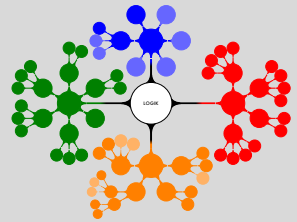
$$\left[\begin{array}{cccccc} 1 & -1 & 0 & 1 & 2/5 & -6/5 \\ 0 & 1 & 0 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 & -1/5 & 3/5 \end{array}\right] \quad \left\| \begin{array}{l} \\ R_2 - (1/-1)R_1 \rightarrow \end{array} \right\|$$

$$\left[\begin{array}{cccccc} 1 & 0 & 0 & 0 & 2/5 & -1/5 \\ 0 & 1 & 0 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 & -1/5 & 3/5 \end{array}\right]$$

- We confirm the fact that we have found the inverse by multiplication:

$$\underbrace{\left[\begin{array}{ccc} 1 & -1 & 2 \\ 3 & 0 & 1 \\ 1 & 0 & 2 \end{array}\right]}_{A} * \underbrace{\left[\begin{array}{ccc} 0 & 2/5 & -1/5 \\ -1 & 0 & 1 \\ 0 & -1/5 & 3/5 \end{array}\right]}_{A^{-1}} = \underbrace{\left[\begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{array}\right]}_{I}$$





# The Inverse of a Matrix III

- It is more *efficient* to use Gaussian elimination. We show only the final triangular matrix; we used pivoting:

$$\begin{bmatrix} 1 & -1 & 2 & 1 & 0 & 0 \\ 3 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 2 & 0 & 0 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 3 & 0 & 1 & 0 & 1 & 0 \\ (0.333) & -1 & 1.667 & 1 & -0.333 & 0 \\ (0.333) & (0) & 1.667 & 0 & -0.333 & 1 \end{bmatrix}$$

- After doing the back-substitutions, we get

$$\begin{bmatrix} 3 & 0 & 1 & 0 & 0.4 & -0.2 \\ (0.333) & -1 & 1.667 & -1 & 0 & 1 \\ (0.333) & (0) & 1.667 & 0 & -0.2 & 0.6 \end{bmatrix}$$

- If we have the inverse of a matrix, we can *use it to solve a set of equations*, $Ax = b$,
- because multiplying by $A^{-1}$ gives the answer ($x$):

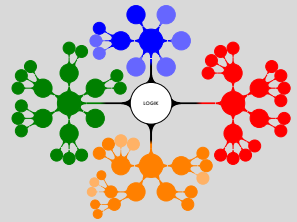
$$A^{-1}Ax = A^{-1}b$$
$$x = A^{-1}b$$

- Phyton Code:

```
1 import numpy as np
2 A = np.array([[1.0,-1.0,2.0],[3.0,0.0,1.0],[1.0,0.0,2.0]])
3 b = np.array([[1.0,0.0,0.0],[0.0,1.0,0.0],[0.0,0.0,1.0]])
4 x = np.linalg.solve(A, b)
5 print("NumPy - Inverse Matrix: \n",x)
6 from scipy import linalg
7 x=linalg.solve(A,b)
8 print("SciPy - Inverse Matrix: \n",x)
```



Dr. Cem Özdoğan



# Eigenvalues and Eigenvectors I

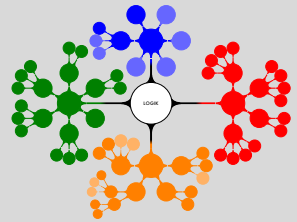
- For a square matrix A,

$$A\vec{u} = \lambda\vec{u}$$

  $\vec{u}$ vectors satisfying this condition are called the *eigenvectors* of the matrix A, and the lambda scalar coefficients are called the *eigenvalues*.
- An $N \times N$ matrix has $N$ different eigenvectors. However, the corresponding lambda eigenvalues for these eigenvectors may not be different.
- **State of a system can be expressed in terms of the eigenvectors of the system of linear equations and in terms of their eigenvalues for the measured quantities.**
- Create an $U$ matrix by arranging eigenvectors side by side:

$$U = \overbrace{\begin{bmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ u_{21} & u_{22} & \dots & u_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ u_{n1} & u_{n2} & \dots & u_{nn} \end{bmatrix}}^{\vec{u_1} \quad \vec{u_2} \quad \dots \quad \vec{u_n}}$$





# Eigenvalues and Eigenvectors II

- When we multiply the matrix $A$ with the matrix $U$ and its inverse matrix $U^{-1}$ from both sides, we get

$$A' = U^{-1}AU = \begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & 0 & \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & \lambda_n \end{bmatrix}$$

- That is, the similarity transformation with the eigenvectors matrix makes $A$ as being diagonalized and the elements on the diagonal become the eigenvalues of $A$.
- In principle, the eigenvalue problem is easy to solve. So-called characteristic equation is to be solved:

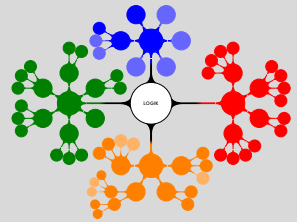
$$det|A - \lambda I| = 0$$

- After finding the roots of this n-degree polynomial equation, the corresponding eigenvectors can be obtained by solving the following system of equations:

$$(A - \lambda I)\vec{v} = 0$$

- **Since this method requires determinant calculation, it is not useful for large dimensional matrices.**





# Normal Modes of Oscillation I

- Consider the coupled oscillations problem of two equal masses $m$ connected by springs of constant $k$ (see Figure).

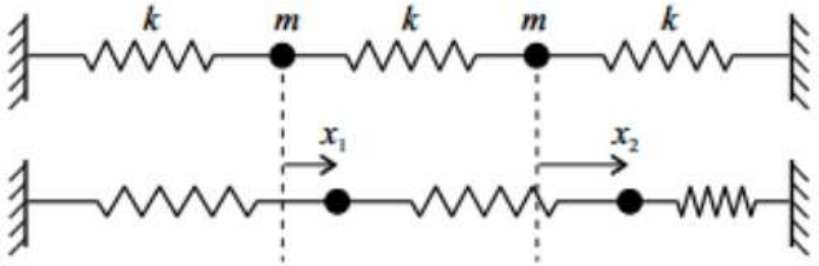


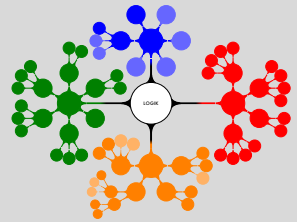

**Figure:** Mass-Spring system.

- The differential equation provided by each mass is written using Newton's law of motion as follows

$$\begin{aligned} k(x_2 - x_1) - kx_1 &= m\frac{d^2x_1}{dt^2} \\ -kx_2 - k(x_2 - x_1) &= m\frac{d^2x_2}{dt^2} \end{aligned}$$

- In this system, the frequencies ($\omega$) that both masses oscillate as in common are called **normal oscillation modes**.





# Normal Modes of Oscillation II

- To find the normal mode frequencies, try a solution for both unknowns as:

$$x_1 = x_{10} cos\omega t \quad \& \quad x_2 = x_{20} cos\omega t$$

- Substitute these solutions into the system of linear equations above and simplify by removing $cos\omega t$,

$$\begin{aligned} \frac{2k}{m} x_{10} - \frac{k}{m} x_{20} &= \omega^2 x_{10} \\ -\frac{k}{m} x_{10} + \frac{2k}{m} x_{20} &= \omega^2 x_{20} \end{aligned}$$

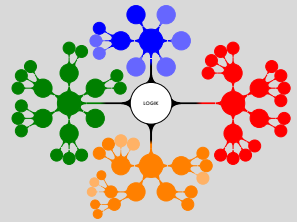
- This system of equations can be written as the product of a matrix and a column vector as follows (replace $\omega^2$ by $\lambda$ ):

$$\begin{bmatrix} 2k/m & -k/m \\ -k/m & 2k/m \end{bmatrix} \begin{bmatrix} x_{10} \\ x_{20} \end{bmatrix} = \lambda \begin{bmatrix} x_{10} \\ x_{20} \end{bmatrix}$$

- This structure can also be written as:

$$\begin{bmatrix} 2k/m - \lambda & -k/m \\ -k/m & 2k/m - \lambda \end{bmatrix} \begin{bmatrix} x_{10} \\ x_{20} \end{bmatrix} = 0$$





# Normal Modes of Oscillation III

- The determinant must be zero for this linear system of equations to have a unique solution:

$$det \begin{vmatrix} 2k/m - \lambda & -k/m \\ -k/m & 2k/m - \lambda \end{vmatrix} = 0$$

$$\longrightarrow (2k/m - \lambda)^2 - k^2/m^2 = 0$$

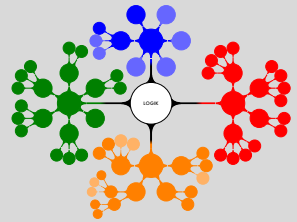
- There are two oscillation frequencies ($\omega$) and their corresponding amplitudes $\vec{x_0} = (x_{10}, x_{20})$:

$$\lambda_1 = k/m \longrightarrow \vec{x_{0,1}} = \begin{pmatrix} 0.71 \\ 0.71 \end{pmatrix}$$

$$\lambda_2 = 3k/m \longrightarrow \vec{x_{0,2}} = \begin{pmatrix} -0.71 \\ 0.71 \end{pmatrix}$$

- The first of these solutions represents the mode in which the two masses oscillate in the **same phase** (–> –>)
- and the second represents the mode in which they oscillate in the **opposite phase** (–> <–).



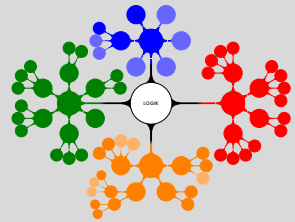


# Normal Modes of Oscillation IV

## Phyton Code:

```
print("****************SymPy Solution for Characteristic Equation: ")
from sympy import Matrix, symbols, pprint, factor
M = Matrix([[2, -1], [-1, 2]])
lamda = symbols('lamda')
poly = M.charpoly(lamda) # Get the characteristic polynomial
print(poly) # Printing polynomial
pprint(factor(poly.as_expr())) # Prints expr in pretty form.
print("****************NumPy Solution for Characteristic Equation: ")
import numpy as np
A = np.array([[2, -1], [-1, 2]])
print(np.poly(A))
print("****************NumPy Solution for Eigenvalues and Eigenvectors: ")
w,v=np.linalg.eig(A)
print('Eigenvalue:', w)
print('Eigenvector1:', v[0])
print('Eigenvector2:', v[1])
print("****************SciPy Solution for Eigenvalues and Eigenvectors: ")
import scipy.linalg as la
w,v = la.eig(A)
print('Eigenvalue:', w)
print('Eigenvector1:', v[0])
print('Eigenvector2:', v[1])
```





# Iterative Methods

- Gaussian elimination and its variants are called ***direct methods***.
- An entirely different way to solve many systems is through ***iteration***.
- In this way, we start with an initial estimate of the solution vector and proceed to refine this estimate.
- An $n \times n$ matrix $A$ is *diagonally dominant* if and only if;

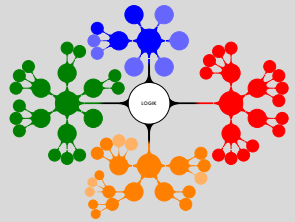
$$|a_{ii}| > \sum_{j=1, j\neq i}^{n} |a_{ij}|, \quad i = 1, 2, \ldots, n$$

- Example. Given matrix & After reordering;

$$\begin{aligned} 6x_1 - 2x_2 + x_3 &= 11 \\ x_1 + 2x_2 - 5x_3 &= -1 \\ -2x_1 + 7x_2 + 2x_3 &= 5 \end{aligned} \quad \& \quad \begin{aligned} 6x_1 - 2x_2 + x_3 &= 11 \\ -2x_1 + 7x_2 + 2x_3 &= 5 \\ x_1 + 2x_2 - 5x_3 &= -1 \end{aligned}$$

- The solution is $x_1 = 2, x_2 = 1, x_3 = 1$ (for both cases?).
- Before we begin our iterative scheme we must first reorder the equations so that the coefficient matrix is diagonally dominant.





# Jacobi Method I

- The iterative methods depend on the rearrangement of the equations in this manner:

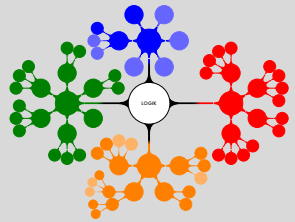
$$\boxed{x_i = \frac{b_i}{a_{ii}} - \sum_{j=1, j\neq i}^{n} \frac{a_{ij}}{a_{ii}} x_j}, i = 1, 2, \ldots, n, \mapsto x_1 = \frac{11}{6} - \left( \frac{-2}{6} x_2 + \frac{1}{6} x_3 \right) \tag{1}$$

- Each equation now solved for the variables in succession:

$$\begin{aligned} x_1 &= 1.8333 + 0.3333 x_2 - 0.1667 x_3 \\ x_2 &= 0.7143 + 0.2857 x_1 - 0.2857 x_3 \\ x_3 &= 0.2000 + 0.2000 x_1 + 0.4000 x_2 \end{aligned}$$

- We begin with some initial approximation to the value of the variables.
- Say initial values are; $x_1 = 0, x_2 = 0, x_3 = 0$. Each component might be taken equal to *zero if no better initial estimates* are at hand.





# Jacobi Method II

- The new values are substituted in the right-hand sides to generate a second approximation,
- and the process is repeated until successive values of each of the variables are sufficiently alike.
- Now, general form

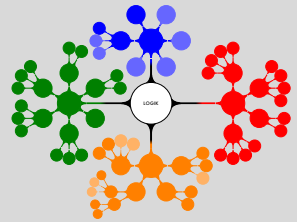
$$\begin{aligned} x_1^{(n+1)} &= 1.8333 + 0.3333x_2^{(n)} - 0.1667x_3^{(n)} \\ x_2^{(n+1)} &= 0.7143 + 0.2857x_1^{(n)} - 0.2857x_3^{(n)} \\ x_3^{(n+1)} &= 0.2000 + 0.2000x_1^{(n)} + 0.4000x_2^{(n)} \end{aligned} \tag{2}$$

- Starting with an initial vector of $x^{(0)} = (0, 0, 0, )$, we obtain Table 1

| | First | Second | Third | Fourth | Fifth | Sixth | ... | Ninth |
|---|---|---|---|---|---|---|---|---|
| $x_1$ | 0 | 1.833 | 2.038 | 2.085 | 2.004 | 1.994 | ... | 2.000 |
| $x_2$ | 0 | 0.714 | 1.181 | 1.053 | 1.001 | 0.990 | ... | 1.000 |
| $x_3$ | 0 | 0.200 | 0.852 | 1.080 | 1.038 | 1.001 | ... | 1.000 |

**Table:** Successive estimates of solution (Jacobi method)





# Jacobi Method III

- Rewrite in matrix notation; let $A = L + D + U$,

$$Ax = b \Longrightarrow \begin{bmatrix} 6 & -2 & 1 \\ -2 & 7 & 2 \\ 1 & 2 & -5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 11 \\ 5 \\ -1 \end{bmatrix}$$

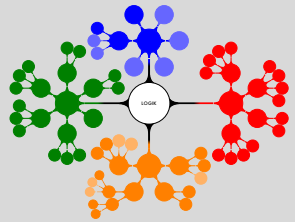
$$L = \begin{bmatrix} 0 & 0 & 0 \\ -2 & 0 & 0 \\ 1 & 2 & 0 \end{bmatrix}, D = \begin{bmatrix} 6 & 0 & 0 \\ 0 & 7 & 0 \\ 0 & 0 & -5 \end{bmatrix}, U = \begin{bmatrix} 0 & -2 & 1 \\ 0 & 0 & 2 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\begin{aligned} Ax &= (L + D + U)x = b \\ Dx &= -(L + U)x + b \\ x &= -D^{-1}(L + U)x + D^{-1}b \end{aligned}$$

- From this we have, identifying $x$ on the left as the new iterate,

$$x^{(n+1)} = -D^{-1}(L + U)x^{(n)} + D^{-1}b$$





# Jacobi Method IV

- In Eqn. 2,

$$b' = D^{-1}b = \begin{bmatrix} 1.8333 \\ 0.7143 \\ 0.2000 \end{bmatrix}$$

$$D^{-1}(L+U) = \begin{bmatrix} 0 & -0.3333 & 0.1667 \\ -0.2857 & 0 & 0.2857 \\ -0.2000 & -0.4000 & 0 \end{bmatrix}$$

- This procedure is known as the Jacobi method, also called "*the method of simultaneous displacements*",
- because each of the equations is simultaneously changed by using the most recent set of $x$-values (see Table 1).
- **Example py-file:** The Jacobi approximation to the solution of AX = B. myJacobi.py