GIAN Course on Solving Linear Systems and Computing Generalized Inverses Using Recurrent Neural Networks
June 09-19, 2025, IIT Indore,
(Tutorial)

June 11, 2025

Electrical Circuit Leading to Linear System

In **Example 1.2.6**, an electrical circuit results in the system:

$$Ax = b,$$

where:

$$A = \begin{bmatrix} 10 & -1 & 2 & 0 \\ -1 & 11 & -1 & 3 \\ 2 & -1 & 10 & -1 \\ 0 & 3 & -1 & 8 \end{bmatrix}, \quad b = \begin{bmatrix} 6 \\ 25 \\ -11 \\ 15 \end{bmatrix}$$

Goal: Find voltages at the nodes, i.e., entries of x.

Accurate MATLAB Solution

We can compute the exact solution in MATLAB as:

```
A = [10 -1 2 0; -1 11 -1 3; 2 -1 10 -1; 0 3 -1 8];
b = [6; 25; -11; 15];
x = A \ b
```

Output:

$$x = \begin{bmatrix} 1 \\ 2 \\ -1 \\ 1 \end{bmatrix}$$

This solution is extremely accurate when using MATLAB or any reliable solver.

Theorem 2.3.9 – Sensitivity of Linear Systems

Suppose A is nonsingular. Then small changes in A and b cause:

$$\frac{\|\delta x\|}{\|x\|} \leq \kappa(A) \left(\frac{\|\delta A\|}{\|A\|} + \frac{\|\delta b\|}{\|b\|} \right)$$

Interpretation:

- $\kappa(A) = \|A\| \cdot \|A^{-1}\|$: Condition number
- If $\kappa(A)$ is large, solution is sensitive to perturbations
- Errors in resistors/battery $\rightarrow$ errors in A, b

Applying Theorem 2.3.9

Assume:

- Relative errors: $\|\delta A\|/\|A\|, \|\delta b\|/\|b\| < 10^{-4}$
- Condition number $\kappa(A) \approx 2500$

Then:

$$\frac{\|\delta x\|}{\|x\|} < 2500 \cdot (10^{-4} + 10^{-4}) = 0.5\%$$

Conclusion: The computed solution deviates from the true value by at most 0.5

MATLAB: Simulating Perturbations

```matlab
% Original system
A = [10 -1 2 0; -1 11 -1 3; 2 -1 10 -1; 0 3 -1 8];
b = [6; 25; -11; 15];
x = A \ b;

% Perturbation
dA = 1e-4 * randn(4);
db = 1e-4 * randn(4,1);

% Perturbed solution
x_perturbed = (A + dA) \ (b + db);

% Relative error
rel_error = norm(x - x_perturbed) / norm(x)
```

Expected ‘rel_error 0.003‘ (or less than 0.5% as predicted).

Conclusion

- Even small errors in system parameters can affect solution
- MATLAB shows actual error is often *much less* than the bound

Function File: solve_LU.m

```
function x = solve_LU(A, b)
    % Solve Ax = b using LU decomposition (no pivoting)

    [L, U] = lu(A);

    n = length(b);
    z = zeros(n, 1);
    for i = 1:n
        z(i) = b(i) - L(i,1:i-1)*z(1:i-1);
    end

    x = zeros(n, 1);
    for i = n:-1:1
        x(i) = (z(i) - U(i,i+1:n)*x(i+1:n)) / U(i,i);
    end
end
```

Calling the Function

```
% Define input
A = [2, -1, 1; 3, 3, 9; 3, 3, 5];
b = [2; -1; 4];

% Call the function
x = solve_LU(A, b);

% Display result
disp('Solution x:');
disp(x);
```

GLOBAL INITIATIVE OF ACADEMIC NETWORKS

Function File: solve_LU_pivot.m

```
function x = solve_LU_pivot(A, b)
    [L, U, P] = lu(A);  % A = P'*L*U
    Pb = P * b;

    n = length(b);
    z = zeros(n, 1);
    for i = 1:n
        z(i) = Pb(i) - L(i,1:i-1)*z(1:i-1);
    end

    x = zeros(n, 1);
    for i = n:-1:1
        x(i) = (z(i) - U(i,i+1:n)*x(i+1:n))/ U(i,i);
    end
end
```

Calling the Function

```
% Example usage
A = [2, -1, 1; 3, 3, 9; 3, 3, 5];
b = [2; -1; 4];

x = solve_LU_pivot(A, b);

disp('Solution x:');
disp(x);
```

Example: LU Solver with Flop Count

```
function [x, flop_count] = solve_LU_flops(A, b)
    % LU decomposition with partial pivoting + flop count

    flop_count = 0;
    [L, U, P] = lu(A); % Pivoting cost is negligible
    Pb = P * b;

    n = length(b);
    z = zeros(n, 1);
    for i = 1:n
        temp = 0;
        for j = 1:i-1
            temp = temp + L(i,j) * z(j);
            flop_count = flop_count + 2; % mul + add
        end

        z(i) = Pb(i) - temp;
        flop_count = flop_count + 1; % subtraction
```

Using the Flop Counter Function

```
% Define system
A = [2, -1, 1; 3, 3, 9; 3, 3, 5];
b = [2; -1; 4];

% Call LU solver with flop counting
[x, flops] = solve_LU_flops(A, b);

disp('Solution:');
disp(x);
fprintf('Total flops: %d\n', flops);
```

Solving $H_n x = b$ with Hilbert Matrix

Let $z = \text{ones}(n, 1)$, and $b = H_n z$, where H_n is the $n \times n$ Hilbert matrix. In theory, solving $H_n x = b$ should return $x = z$. Let's test this numerically:

```
for n = [4 8 12 16]
    H = hilb(n);
    z = ones(n,1);
    b = H * z;
    x = H \ b;
    kappa = cond(H);
    error = norm(x - z);
    residual = norm(b - H*x);

    fprintf('n␣=␣%d\n', n);
    fprintf('Condition_number:␣%.2e\n', kappa);
    fprintf('||x␣-␣z||_2:␣%.2e\n', error);
    fprintf('||b␣-␣Hx||_2:␣%.2e\n\n', residual);
end
```

Example 2.7.15: LU Decomposition of a Hilbert Matrix

Objective: Test the accuracy and stability of LU decomposition on the 12×12 Hilbert matrix.

MATLAB Code:

```
A = hilb(12);
[L, U] = lu(A);
E = L * U - A;

normE = norm(E);
normA = norm(A);
rel_error = normE / normA;
```

Output:

$$\|LU - A\|/\|A\| \approx 2.7 \times 10^{-17}$$

Conclusion: The LU decomposition is extremely accurate despite the ill-conditioning.

(a) Set up the Overdetermined System

We are fitting a straight line

$$p(t) = x_1 + x_2 t$$

to five data points (t_i, b_i), with basis functions:

$$\phi_1(t) = 1, \quad \phi_2(t) = t$$

This leads to the overdetermined system:

$$Ax = b, \quad \text{where } A = \begin{bmatrix} 1 & t_1 \\ 1 & t_2 \\ \vdots & \vdots \\ 1 & t_5 \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

(b) MATLAB Code for Least Squares Solution

MATLAB Code

```
t = (1:0.5:3)';          % t values
b = [2.2; 2.8; 3.6; 4.5; 5.1]; % example data
A = [ones(size(t)), t];

x = A \ b;  % Least squares solution
```

(b) MATLAB Code for Least Squares Solution

MATLAB Code

```
t = (1:0.5:3)';            % t values
b = [2.2; 2.8; 3.6; 4.5; 5.1]; % example data
A = [ones(size(t)), t];

x = A \ b;  % Least squares solution
```

- $A \in \mathbb{R}^{5\times 2}$, **overdetermined**
- **MATLAB's backslash operator (‘) automatically computes least squares solution when A is not square**

(c) Plotting the Data and Least Squares Fit

MATLAB Plotting Code:

```
plot(t, b, 'ro', 'MarkerSize', 8); hold on;
t_fit = linspace(min(t), max(t), 100);
p_fit = x(1) + x(2) * t_fit;
plot(t_fit, p_fit, 'b-', 'LineWidth', 2);
xlabel('t'); ylabel('b');
legend('Data Points', 'Least Squares Line');
title('Least Squares Fit of Straight Line');
grid on;
```

(c) Plotting the Data and Least Squares Fit

MATLAB Plotting Code:

```
plot(t, b, 'ro', 'MarkerSize', 8); hold on;
t_fit = linspace(min(t), max(t), 100);
p_fit = x(1) + x(2) * t_fit;
plot(t_fit, p_fit, 'b-', 'LineWidth', 2);
xlabel('t'); ylabel('b');
legend('Data Points', 'Least Squares Line');
title('Least Squares Fit of Straight Line');
grid on;
```

- Red circles: observed data
- Blue line: best fit line $p(t) = x_1 + x_2 t$

(d) Compute the Residual Norm

Residual vector: $r = b - Ax$

MATLAB Code:

```
r = b - A*x;
residual_norm = norm(r, 2);
disp(['Residual norm: ', num2str(residual_norm)]);
```

(d) Compute the Residual Norm

Residual vector: $r = b - Ax$

MATLAB Code:

```
r = b - A*x;
residual_norm = norm(r, 2);
disp(['Residual norm: ', num2str(residual_norm)]);
```

Interpretation:

- $\|r\|_2$ measures how well the line fits the data
- A small residual norm implies a good fit

Summary

- Constructed an overdetermined system using basis $\phi_1(t) = 1$, $\phi_2(t) = t$
- Solved it in MATLAB using ‘A b‘ to obtain least squares solution
- Visualized the fit using the ‘plot‘ command
- Computed the 2-norm of the residual to assess accuracy

(a) Gram-Schmidt Process

Let

$$v_1 = \begin{bmatrix} 3 \\ -3 \\ 3 \\ -3 \end{bmatrix}, \quad v_2 = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

We apply the **Gram-Schmidt process** to obtain an orthonormal basis $\{q_1, q_2\}$:

$$q_1 = \frac{v_1}{\|v_1\|} = \frac{1}{6} \begin{bmatrix} 3 \\ -3 \\ 3 \\ -3 \end{bmatrix}$$

Project v_2 onto q_1:

$$r_{12} = q_1^T v_2 = \frac{1}{6}(3 - 6 + 9 - 12) = -1$$

$$u_2 = v_2 - r_{12} q_1 = v_2 + q_1 = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} + \frac{1}{6} \begin{bmatrix} 3 \\ -3 \\ 3 \\ -3 \end{bmatrix} = \begin{bmatrix} \frac{9}{6} \\ \frac{9}{6} \\ \frac{21}{6} \\ \frac{21}{6} \end{bmatrix}$$

$$u_2 = v_2 - r_{12}q_1 = v_2 + q_1 = \begin{bmatrix}1\\2\\3\\4\end{bmatrix} + \frac{1}{6}\begin{bmatrix}3\\-3\\3\\-3\end{bmatrix} = \begin{bmatrix}\frac{9}{6}\\\frac{9}{6}\\\frac{21}{6}\\\frac{21}{6}\end{bmatrix}$$

$$q_2 = \frac{u_2}{\|u_2\|} = \frac{1}{\sqrt{18}}\begin{bmatrix}\frac{3}{2}\\\frac{3}{2}\\\frac{7}{2}\\\frac{7}{2}\end{bmatrix}$$

$$u_2 = v_2 - r_{12}q_1 = v_2 + q_1 = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} + \frac{1}{6}\begin{bmatrix} 3 \\ -3 \\ 3 \\ -3 \end{bmatrix} = \begin{bmatrix} \frac{9}{6} \\ \frac{9}{6} \\ \frac{21}{6} \\ \frac{21}{6} \end{bmatrix}$$

$$q_2 = \frac{u_2}{\|u_2\|} = \frac{1}{\sqrt{18}}\begin{bmatrix} \frac{3}{2} \\ \frac{3}{2} \\ \frac{7}{2} \\ \frac{7}{2} \end{bmatrix}$$

Save coefficients: $r_{11} = \|v_1\| = 6,\ r_{12} = -1,\ r_{22} = \|u_2\| = \sqrt{18}$

(b) Constructing Q and R

From part (a):

$$Q = \begin{bmatrix} \frac{1}{2} & \frac{1}{\sqrt{18}} \cdot \frac{3}{2} \\ -\frac{1}{2} & \frac{1}{\sqrt{18}} \cdot \frac{3}{2} \\ \frac{1}{2} & \frac{1}{\sqrt{18}} \cdot \frac{7}{2} \\ -\frac{1}{2} & \frac{1}{\sqrt{18}} \cdot \frac{7}{2} \end{bmatrix}, \quad R = \begin{bmatrix} 6 & -1 \\ 0 & \sqrt{18} \end{bmatrix}$$

(b) Constructing Q and R

From part (a):

$$Q = \begin{bmatrix} \frac{1}{2} & \frac{1}{\sqrt{18}} \cdot \frac{3}{2} \\ -\frac{1}{2} & \frac{1}{\sqrt{18}} \cdot \frac{3}{2} \\ \frac{1}{2} & \frac{1}{\sqrt{18}} \cdot \frac{7}{2} \\ -\frac{1}{2} & \frac{1}{\sqrt{18}} \cdot \frac{7}{2} \end{bmatrix}, \quad R = \begin{bmatrix} 6 & -1 \\ 0 & \sqrt{18} \end{bmatrix}$$

$$V = \begin{bmatrix} v_1 & v_2 \end{bmatrix} = QR$$

(b) Constructing Q and R

From part (a):

$$Q = \begin{bmatrix} \frac{1}{2} & \frac{1}{\sqrt{18}} \cdot \frac{3}{2} \\ -\frac{1}{2} & \frac{1}{\sqrt{18}} \cdot \frac{3}{2} \\ \frac{1}{2} & \frac{1}{\sqrt{18}} \cdot \frac{7}{2} \\ -\frac{1}{2} & \frac{1}{\sqrt{18}} \cdot \frac{7}{2} \end{bmatrix}, \quad R = \begin{bmatrix} 6 & -1 \\ 0 & \sqrt{18} \end{bmatrix}$$

$$V = \begin{bmatrix} v_1 & v_2 \end{bmatrix} = QR$$

Conclusion: We expressed $V \in \mathbb{R}^{4\times 2}$ as a product of:

- $Q \in \mathbb{R}^{4\times 2}$: orthonormal columns
- $R \in \mathbb{R}^{2\times 2}$: upper triangular

(a) Gram-Schmidt Process

Let

$$v_1 = \begin{bmatrix} 3 \\ -3 \\ 3 \\ -3 \end{bmatrix}, \quad v_2 = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

We apply the **Gram-Schmidt process** to get an orthonormal basis $\{q_1, q_2\}$.

(a) Gram-Schmidt Process

Let

$$v_1 = \begin{bmatrix} 3 \\ -3 \\ 3 \\ -3 \end{bmatrix}, \quad v_2 = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

We apply the **Gram-Schmidt process** to get an orthonormal basis $\{q_1, q_2\}$.

Normalize v_1:

$$\|v_1\| = \sqrt{3^2 + (-3)^2 + 3^2 + (-3)^2} = \sqrt{36} = 6 \quad \Rightarrow \quad q_1 = \frac{v_1}{6} = \begin{bmatrix} 0.5 \\ -0.5 \\ 0.5 \\ -0.5 \end{bmatrix}$$

Project v_2 onto q_1:

$$r_{12} = q_1^T v_2 = 0.5(1) + (-0.5)(2) + 0.5(3) + (-0.5)(4) = -1$$

$$u_2 = v_2 - r_{12} q_1 = v_2 + q_1 = \begin{bmatrix} 1.5 \\ 1.5 \\ 3.5 \\ 3.5 \end{bmatrix}$$

$$u_2 = v_2 - r_{12} q_1 = v_2 + q_1 = \begin{bmatrix} 1.5 \\ 1.5 \\ 3.5 \\ 3.5 \end{bmatrix}$$

Normalize u_2:

$$\|u_2\| = \sqrt{1.5^2 + 1.5^2 + 3.5^2 + 3.5^2} = \sqrt{36.5} \quad \Rightarrow \quad q_2 = \frac{u_2}{\sqrt{36.5}}$$

$$u_2 = v_2 - r_{12}q_1 = v_2 + q_1 = \begin{bmatrix} 1.5 \\ 1.5 \\ 3.5 \\ 3.5 \end{bmatrix}$$

Normalize u_2:

$$\|u_2\| = \sqrt{1.5^2 + 1.5^2 + 3.5^2 + 3.5^2} = \sqrt{36.5} \quad \Rightarrow \quad q_2 = \frac{u_2}{\sqrt{36.5}}$$

Save coefficients:

$$r_{11} = 6, \quad r_{12} = -1, r_{22} = \sqrt{36.5}$$

(b) Constructing Q and R

From part (a):

$$Q = \begin{bmatrix} 0.5 & \frac{1.5}{\sqrt{36.5}} \\ -0.5 & \frac{1.5}{\sqrt{36.5}} \\ 0.5 & \frac{3.5}{\sqrt{36.5}} \\ -0.5 & \frac{3.5}{\sqrt{36.5}} \end{bmatrix}, \quad R = \begin{bmatrix} 6 & -1 \\ 0 & \sqrt{36.5} \end{bmatrix}$$

(b) Constructing Q and R

From part (a):

$$Q = \begin{bmatrix} 0.5 & \frac{1.5}{\sqrt{36.5}} \\ -0.5 & \frac{1.5}{\sqrt{36.5}} \\ 0.5 & \frac{3.5}{\sqrt{36.5}} \\ -0.5 & \frac{3.5}{\sqrt{36.5}} \end{bmatrix}, \quad R = \begin{bmatrix} 6 & -1 \\ 0 & \sqrt{36.5} \end{bmatrix}$$

Then:

$$V = \begin{bmatrix} v_1 & v_2 \end{bmatrix} = QR$$

(b) Constructing Q and R

From part (a):

$$Q = \begin{bmatrix} 0.5 & \frac{1.5}{\sqrt{36.5}} \\ -0.5 & \frac{1.5}{\sqrt{36.5}} \\ 0.5 & \frac{3.5}{\sqrt{36.5}} \\ -0.5 & \frac{3.5}{\sqrt{36.5}} \end{bmatrix}, \quad R = \begin{bmatrix} 6 & -1 \\ 0 & \sqrt{36.5} \end{bmatrix}$$

Then:

$$V = \begin{bmatrix} v_1 & v_2 \end{bmatrix} = QR$$

Conclusion:

- $Q \in \mathbb{R}^{4\times 2}$ has orthonormal columns
- $R \in \mathbb{R}^{2\times 2}$ is upper triangular

Schur Decomposition

Theorem (Schur Decomposition): Let $A \in \mathbb{C}^{n \times n}$ be a square matrix. Then there exists a unitary matrix $Q \in \mathbb{C}^{n \times n}$ and an upper triangular matrix $T \in \mathbb{C}^{n \times n}$ such that:

$$\boxed{A = QTQ^*}$$

where Q^* is the conjugate transpose of Q.

Schur Decomposition

Theorem (Schur Decomposition): Let $A \in \mathbb{C}^{n \times n}$ be a square matrix. Then there exists a unitary matrix $Q \in \mathbb{C}^{n \times n}$ and an upper triangular matrix $T \in \mathbb{C}^{n \times n}$ such that:

$$\boxed{A = QTQ^*}$$

where Q^* is the conjugate transpose of Q.

Properties:

- The diagonal entries of T are the eigenvalues of A.
- If $A \in \mathbb{R}^{n \times n}$, then a real Schur decomposition exists:

$$A = QTQ^\top$$

  with Q orthogonal and T quasi-upper triangular (i.e., 1×1 and 2×2 blocks on the diagonal).
- Always exists for any square matrix.

MATLAB Example: Schur Decomposition

Given a matrix A, compute its Schur decomposition using MATLAB:

MATLAB Code

```
A = randn(4);             % Random 4x4 matrix
[Q, T] = schur(A);        % Schur decomposition
norm(A - Q*T*Q')          % Should be close to zero
eig(A), diag(T)           % Compare eigenvalues
```

MATLAB Example: Schur Decomposition

Given a matrix A, compute its Schur decomposition using MATLAB:

MATLAB Code

```
A = randn(4);            % Random 4x4 matrix
[Q, T] = schur(A);       % Schur decomposition
norm(A - Q*T*Q')         % Should be close to zero
eig(A), diag(T)          % Compare eigenvalues
```

Observation:

- Q is unitary (or orthogonal if A is real).
- T has eigenvalues of A on its diagonal.

LU and QR Decomposition

LU Decomposition

- `[L, U] = lu(A)` $(A = LU)$
- `[L, U, P] = lu(A)` $(A = PLU)$

QR Decomposition

- `[Q, R] = qr(A)` $(A = QR)$
- `[Q, R] = qr(A, 0)` (Economy QR)
- `[Q, R, P] = qr(A)` (Column pivoting: $AP = QR$)

Cholesky and Eigen Decomposition

Cholesky Decomposition

- `R = chol(A)` $(A = R^T R)$
- `R = chol(A, 'lower')` $(A = LL^T)$

Eigenvalue Decomposition

- `[V, D] = eig(A)` $(A = VDV^{-1})$

SVD and Schur Decomposition

Singular Value Decomposition (SVD)

- `[U, S, V] = svd(A)` $(A = USV^T)$
- `[U, S, V] = svd(A, 'econ')` (Economy SVD)

Schur Decomposition

- `[Q, T] = schur(A)` $(A = QTQ^T)$
- `[Q, T] = schur(A, 'real')` (Real Schur form)

Other Matrix Decompositions

Hessenberg Decomposition

- `[Q, H] = hess(A)` $(A = QHQ^T)$

Jordan Canonical Form

- `[J, P] = jordan(A)` $(A = PJP^{-1})$
- *(Use cautiously due to numerical instability)*

LDL Decomposition (symmetric)

- `[L, D, P] = ldl(A)` $(A = P^T LDL^T P)$

Polar Decomposition and Utilities

Polar Decomposition (via SVD)

- `[U, S, V] = svd(A)`
- `P = V*S*V'` (positive semidefinite part)
- `U_polar = U*V'` (orthogonal part)

Rank-Revealing QR

- `[Q, R, E] = qr(A, 'vector')` $(A(:, E) = QR)$

Useful Matrix Commands

- `rank(A)`, `det(A)`, `norm(A)`
- `inv(A)` (Use with caution)
- `cond(A)` (Condition number)