

# Tutorial 8: Sparse linear algebra in MATLAB

COMP5930M Scientific Computation

# Today

MATLAB sparse matrix handling

Creating sparse matrices

Experiments with real-world matrices

Direct solvers: fill-in and reordering

Indirect solvers: Jacobi and Gauss-Seidel

## Sparse matrices in MATLAB

- ▶ MATLAB uses sparse column-major format
- ▶ We do not need to access it in that form
- ▶ Most MATLAB operations on matrices, eg. multiplication or  $\backslash$ , also work automatically on sparse matrices
- ▶ Main difference in eigenvalue computations; for dense matrices use eig, for sparse matrices use eigs

## Creating and accessing sparse matrices

- ▶ Creation:  $A = \text{spalloc}(n,n,\text{nnz})$   
A is an  $n \times n$  sparse matrix with space for  $\text{nnz}$  non-zeros
- ▶ Access:  $A(i,j)=c$  or  $c=A(i,j)$   
Direct, simple access through normal matrix notation  
(no need to use the sparse column data structure)
- ▶ Visualisation:  $\text{spy}(A)$   
Plots a graph of the sparse structure

## Finding challenging linear problems to solve

- ▶ Matrix market: <https://math.nist.gov/MatrixMarket/>
- ▶ Download matrices arising from real-world problems:
  - ▶ flow dynamics
  - ▶ structural engineering
  - ▶ electrical circuit simulation
  - ▶ economics
  - ▶ quantum physics
  - ▶ etc.
- ▶ Useful resource for testing linear algebra algorithms

## Direct solution by LU-factorisation

```
function x = directLU(A,b)
    [L,U,P] = lu(A);
    z = forwardSubstitution(L, P*b, size(b,1));
    x = backSubstitution(U, z, size(z,1));
end
```

**Note:** The decomposition  $PA = LU$  implies solving two systems:

1.  $Lz = Pb$
2.  $Ux = z$

The first problem is solved using **forward-substitution** and the second problem **back-substitution**. Both have  $O(n^2)$  complexity.

## Back-substitution

```
function x = backSubstitution(U,b,n)

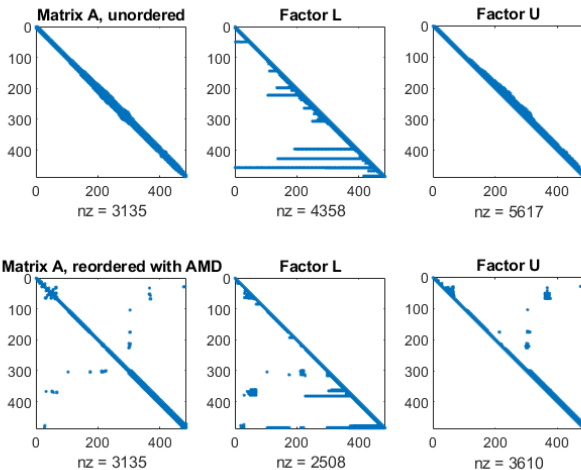
x = zeros(n,1);
for j = n : -1 : 1
    if (U(j,j)==0) error('Matrix is singular!'); end;
    x(j) = b(j) / U(j,j);
    b(1:j-1) = b(1:j-1) - U(1:j-1,j) * x(j);
end
```

## Forward-substitution

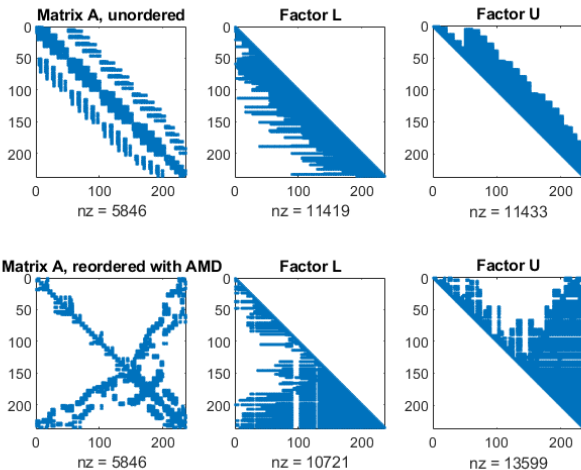
```
function x = forwardSubstitution(L,b,n)

x = zeros(n,1);
for j = 1 : n
    if (L(j,j)==0) error('Matrix is singular!'); end;
    x(j)=b(j)/L(j,j);
    b(j+1:n)=b(j+1:n)-L(j+1:n,j)*x(j);
end
```





For very sparse matrices, **AMD reduces fill-in of  $L, U$  factors**



For more complex matrices there can be little reduction in fill-in

## Jacobi and Gauss-Seidel iteration

```
% Split the matrix  $A = D + E$ 
D = diag(diag(A)); % Jacobi
D = triu(A,0);      % Gauss-Seidel
E = A - D;

Bfun = @(X)(-D\(E*X)); % Function to evaluate  $B * x$ 
z = D \ b;

while ( norm(r)/norm(x0) > tol && k < maxIter )
    x = Bfun(x0) + z;
    x0 = x;
    r = b - A*x0;
    k = k + 1;
end
```

## Example: Comparison of basic iterative methods

The matrix

$$A = \begin{bmatrix} 14 & 0 & 5 & 2 & 1 \\ 0 & 8 & -3 & 1 & -2 \\ 5 & -3 & 12 & 2 & 5 \\ 2 & 1 & 2 & 6 & 1 \\ 1 & -2 & 5 & 1 & 8 \end{bmatrix}$$

is symmetric, positive definite but not strictly diagonally dominant.

Jacobi:  $\rho(B) \approx 0.916$  method converges in 21 iterations

Gauss-Seidel:  $\rho(B) \approx 0.262$  method converges in 7 iterations