Sparse Triangular Systems

- *Triangular systems*
- *Sparse triangular systems with dense right-hand sides*
- *Sparse triangular systems with sparse right-hand sides*
- *A sparse factorization based on sparse triangular solves*

Sparse Triangular linear systems: the problem

The Problem: A is an $n \times n$ matrix, and b a vector of $\mathbb{R}^n$. Find x such that:

$$Ax = b$$

➤ x is the unknown vector, b the right-hand side, and A is the coefficient matrix

➤ We consider the case when A is upper (or lower)triangular.

Two cases:

1. A sparse, b dense vector [solve once or many times]
2. A sparse, b sparse vector [solve once or many times]

Triangular linear systems

Example:

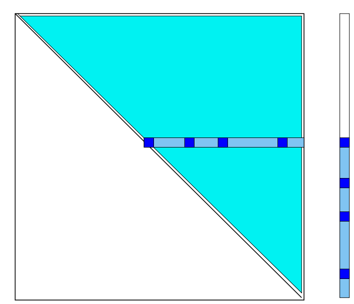
$$\begin{bmatrix} 2 & 4 & 4 \\ 0 & 5 & -2 \\ 0 & 0 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 4 \end{bmatrix}$$

Back-Substitution
Row version

✍1 Operation count?

```
For i = n : -1 : 1 do:
    t := b_i
    For j = i + 1 : n do
        t := t - a_ij x_j
    End
    x_i = t/a_ii
End
```


Illustration for sparse case (Sparse A, dense b)



➤ This will use the CSR data structure

➤ Inner product of a sparse row with a dense column

➤ Sparse BLAS: Sparse 'sdot'

➤ Recall:

```
typedef struct SpaFmt {
/*---------------------------------------------
| C-style CSR format - used internally
| for all matrices in CSR format
|---------------------------------------------*/
  int n;
  int *nzcount;  /* length of each row */
  int **ja;      /*  to store column indices  */
  double **ma;   /*  to store nonzero entries */
} CsMat, *csptr;
```

➤ Can store rows of a matrix (CSR) or its columns (CSC)

➤ Assume that diagonal entry is stored in first location in inverted form.

➤ Result:


```
void Usol(csptr mata, double *b, double *x)
{
  int i, k, *ki;
  double *ma;
  for (i=mata->n-1; i>=0; i--) {
    ma = mata->ma[i];
    ki = mata->ja[i];
    x[i] = b[i] ;
// Note: diag. entry avoided
    for (k=1; k<mata->nzcount[i]; k++)
      x[i] -= ma[k] * x[ki[k]];
    x[i] *= ma[0];
  }
}
```

➤ Operation count?

Column version

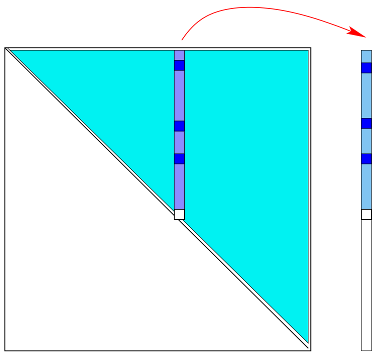
➤ Column version of back-subsitution:

Back-Substitution
Column version

For $j = n : -1 : 1$ do:
  $x_j = b_j / a_{jj}$
  For $i = 1 : j - 1$ do
    $b_i := b_i - x_j * a_{ij}$
  End
End

✍2 Justify the above algorithm [Show that it does indeed give the solution]

Illustration for sparse case (Sparse A, dense b)



➤ Uses the CSC format – (CsMat struct for columns of A)

➤ Sparse BLAS : sparse 'saxpy'

➤ Assumes diagonal entry stored first in inverted form

```
void UsolC(csptr mata, double *b, double *x)
{
  int i, k, *ki;
  double *ma;
  for (i=mata->n-1; i>=0; i--) {
        ja = U->ja[i];
        ma = U->ma[i];
        x[i] *= ma[0];
// Note: diag. entry avoided
        for( j = 1; j < U->nzcount[i]; j++ )
            x[ja[j]] -= ma[j] * x[i];
  }
}
```

✍3 Operation count ?

Sparse A and sparse b

Illustration: Consider solving $Lx = b$ in the situation:

$$L = \begin{bmatrix} * & & & & & \\ * & * & & & & \\ * & & * & & & \\ & * & & * & & \\ * & & * & & * & \\ & * & & & & * \end{bmatrix} \qquad b = \begin{bmatrix} * \\ \\ \\ \\ \\ \\ \end{bmatrix}$$

✍4 Show progress of the pattern of $x = L^{-1}b$ by performing symbolically a column solve for system $Lx = b$.

✍5 Show how this pattern can be determined with Topological sorting. Generalize to any sparse b.

Sparse A and sparse b: Example

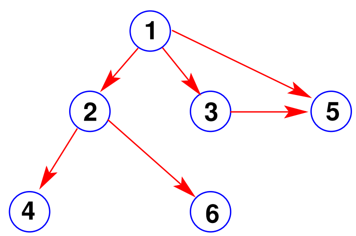
➤ Triangular system of previous example

➤ DAG shown in next figure

➤ Sets dependencies between tasks:

➤ Edge $i \to j$ means $a(j, i) = 1$ (j requires i)

➤ Root: node 1 (see right-hand side b)

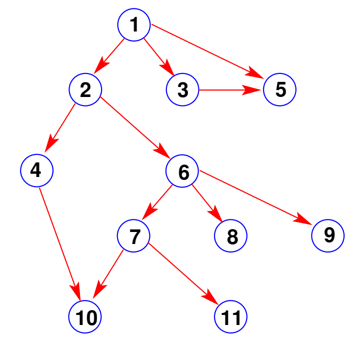


➤ Topological sort: 1, 3, 5, 2, 6, 4 [as produced by a DFS from 1]

➤ In many cases, this leads to a short traversal

✍6 Example: remove link $1 \to 2$ and redo

✍7 Consider a triangular system with the following graph where b has nonzero entries in positions 3 and 7. (1) Progress of solution based on Topolog. sort; (2) Pattern of solution. (3) Verify pattern with matlab.



✍8 Same questions if b has (only) a nonzero entry in position 1.

LU factorization from sparse triangular solves

- LU factorization built one column at a time. At step k:

We want: $\underbrace{L_k}_{n\times n}\underbrace{U_k}_{n\times k} = \underbrace{A_k}_{n\times k} \quad (\equiv A(1:n, 1:k))$

$$\left[\begin{array}{ccc|c|ccc} 1 & & & & & & \\ * & 1 & & & & & \\ * & * & 1 & & & & \\ * & * & * & 1 & & & \\ * & * & * & ? & 1 & & \\ * & * & * & ? & & 1 & \\ * & * & * & ? & & & 1 \end{array}\right] \left[\begin{array}{ccc|c} x & x & x & ? \\ & x & x & ? \\ & & x & . \\ & & & ? \\ & & & 0 \\ & & & 0 \\ & & & 0 \end{array}\right] = A_k$$

- In blue: has been determined. In red: to be determined

- Step 0: Set the terms $?$ in L_k to zero. Result $\equiv \tilde{L}_k$
- Step 1 : Solve $\tilde{L}_k w = a_k$ [Sparse $\tilde{L}_k$, sparse RHS]
- Step 2: set

$$u = \left|\begin{array}{c} w_1 \\ w_2 \\ \vdots \\ w_k \\ \hline 0 \\ \vdots \\ 0 \\ 0 \end{array}\right| \qquad z = \frac{1}{w_k}\left|\begin{array}{c} 0 \\ \vdots \\ 0 \\ 0 \\ \hline w_{k+1} \\ w_{k+2} \\ \vdots \\ w_n \end{array}\right|$$

- Then $L_k U_k = A_k$ with

$$\underbrace{\left[\begin{array}{ccc|c|ccc} 1 & & & & & & \\ * & 1 & & & & & \\ * & * & 1 & & & & \\ * & * & * & 1 & & & \\ * & * & * & z_{k+1} & 1 & & \\ * & * & * & \vdots & & 1 & \\ * & * & * & z_n & & & 1 \end{array}\right]}_{L_k}; \quad \underbrace{\left[\begin{array}{ccc|c} x & x & x & u_1 \\ & x & x & u_2 \\ & & x & \vdots \\ & & & u_k \\ & & & 0 \\ & & & 0 \\ & & & 0 \end{array}\right]}_{U_k}$$

- Verification: Note $L_k = \tilde{L}_k + z e_k^T$; Also $\tilde{L}_k z = z$
- Must verify only $L_k U_k(:,k) = a_k$, i.e., $L_k u = a_k$

$$L_k u = (\tilde{L}_k + z e_k^T) u = \tilde{L}_k (I + z e_k^T) u = \tilde{L}_k (u + w_k z) = \tilde{L}_k w = a_k$$

- Key step: solve triangular system
- In sparse case: sparse triangular system with sparse right-hand side
- Use topological sorting at each step
- Scheme derived from this known as 'left-looking' sparse LU –
- Also known as 'Gilbert and Peierls' approach
- Reference: J. R. Gilbert and T. Peierls, Sparse partial pivoting in time proportional to arithmetic operations, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 862-874

✍D9 Benefit of this approach: Partial pivoting is easy. Show how you would do it.