Matrix-Vector products and Triangular systems

- Matrix-vector products
- Background on linear systems
- Triangular systems
- Sparse Right-hand side.

**Sparse matrices – data structure in C**

Recall:

```c
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)
- Let us first recall how to perform the operation $y = A \times x$ (matvecs) – seen earlier

**Matvec – row version**

```c
void matvec( csptr mata, double *x, double *y )
{
    int i, k, *ki;
    double *kr;
    for (i=0; i<mata->n; i++) {
        y[i] = 0.0;
        kr = mata->ma[i];
        ki = mata->ja[i];
        for (k=0; k<mata->nzcount[i]; k++)
            y[i] += kr[k] * x[ki[k]];
    }
}
```

**Matvec – Column version**

```c
void matvecC( csptr mata, double *x, double *y )
{
    int n = mata->n, i, k, *ki;
    double *kr;
    for (i=0; i<n; i++) {
        y[i] = 0.0;
        kr = mata->ma[i];
        ki = mata->ja[i];
        for (k=0; k<mata->nzcount[i]; k++)
            y[ki[k]] += kr[k] * x[i];
    }
}
```
**Background: Linear systems**

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.

Example:

\[
\begin{cases}
2x_1 + 4x_2 + 4x_3 = 6 \\
x_1 + 5x_2 + 6x_3 = 4 \\
x_1 + 3x_2 + x_3 = 8
\end{cases}
\quad \text{or} \quad
\begin{bmatrix}
2 & 4 & 4 \\
1 & 5 & 6 \\
1 & 3 & 1
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3
\end{bmatrix}
= \begin{bmatrix} 6 \\ 4 \\ 8 \end{bmatrix}
\]

**Standard mathematical solution by Cramer’s rule:**

\[x_i = \frac{\det(A_i)}{\det(A)}\]

- \(A_i\) = matrix obtained by replacing \(i\)-th column by \(b\).
- Note: This formula is useless in practice beyond \(n = 3\) or \(n = 4\).

**Three situations:**

1. The matrix \(A\) is nonsingular. There is a unique solution given by \(x = A^{-1}b\).
2. The matrix \(A\) is singular and \(b \in \text{Ran}(A)\). There are infinitely many solutions.
3. The matrix \(A\) is singular and \(b \notin \text{Ran}(A)\). There are no solutions.

**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

\[\text{For } i = n : -1 : 1 \text{ do:}\]
\[t := b_i\]
\[\text{For } j = i + 1 : n \text{ do}\]
\[t := t - a_{ij}x_j\]
\[\text{End}\]
\[x_i = t/a_{ii}\]
\[\text{End}\]

Illustration for sparse case (Sparse \(A\), dense \(b\))
Assumes diagonal entry stored first in inverted form

```c
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

```c
void UsolC(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] *= ma[0];
        // Note: diag. entry avoided
        for (j=1; j<U->nzcount[i]; j++)
            x[ja[j]] -= ma[j] * x[i];
    }
}
```

Operation count?

Illustration for sparse case

(Sparse A, dense b)

Assumes diagonal entry stored first in inverted form

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

Operation count?
Sparse A and sparse b

Illustration: Consider solving \( Lx = b \) in the situation:

\[
L = \begin{bmatrix}
* & * & * & * \\
* & * & * & * \\
* & * & * & * \\
* & * & * & * \\
\end{bmatrix} \quad b = \begin{bmatrix}
*
\end{bmatrix}
\]

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

Show how this pattern can be determined with Topological sorting. Generalize to any sparse \( b \).

Example: Consider triangular system in previous example.

Graph of matrix shown in next figure

Sets dependencies between tasks

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

Post-order traversal: 6, 4, 2, 5, 3, 1

Reverse: 1, 3, 5, 2, 4, 6

In many cases, this leads to a short traversal

Example: remove link 1 \( \rightarrow \) 2 and redo

Consider a triangular system with the following graph where \( b \) has nonzero entries in positions 3 and 7

Same question if \( b \) has a nonzero entry in position 1.

Explore sparsity of solution in each case.

LU factorization from sparse triangular solves

LU factorization built one column at a time. At step \( k \):

We want:

\[
L_k \in \mathbb{R}^{n \times n \times k} \quad U_k \in \mathbb{R}^{n \times k} \quad (\equiv A(1 : n, 1 : k))
\]

\[
\begin{bmatrix}
1 & x & x & x \\
* & 1 & x & x \\
* & * & 1 & x \\
* & * & * & ? \\
* & * & * & ? \\
* & * & * & ? \\
\end{bmatrix}
\begin{bmatrix}
x \\
x \\
x \\
1 \\
? \\
? \\
\end{bmatrix}
= A_k
\]

In blue: has been determined. In red: to be determined
Step 0: Set the terms $\tilde{L}_k$ 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 $u = w_1 w_2 \ldots w_k 0 \ldots 0$
$z = 1$
$w_{k+1}$
$w_{k+2}$
$w_n$

Then $L_k U_k = A_k$ with

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


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