Data Structures for sparse matrices

➤ The use of a proper data structures is critical to achieving good performance.

✎ Generate a symmetric sparse matrix A in matlab and time the operations of accessing (only) all entries by columns and then by rows. Observations?

➤ Many data structures; sometimes unnecessary variants.

➤ These variants are more useful in the context of iterative methods

➤ Basic linear algebra kernels (e.g., matrix-vector products) depend on data structures.

Some Common Data Structures (from SPARSKIT)

DNS	Dense	ELL	Ellpack-Itpack
BND	Linpack Banded	DIA	Diagonal
COO	Coordinate	BSR	Block Sparse Row
CSR	Compressed Sparse Row	SSK	Symmetric Skyline
CSC	Compressed Sparse Column	NSK	Nonsymmetric Skyline
MSR	Modified CSR	JAD	Jagged Diagonal

➤ Most common (and important): CSR (/ CSC), COO

The coordinate format (COO)

$$A = \begin{pmatrix} 1. & 0. & 0. & 2. & 0. \\ 3. & 4. & 0. & 5. & 0. \\ 6. & 0. & 7. & 8. & 9. \\ 0. & 0. & 10. & 11. & 0. \\ 0. & 0. & 0. & 0. & 12. \end{pmatrix}$$

AA	JR	JC
12.	5	5
9.	3	5
7.	3	3
5.	2	4
1.	1	1
2.	1	4
11.	4	4
3.	2	1
6.	3	1
4.	2	2
8.	3	4
10.	4	3

➤ Simplest data structure -

➤ Often used as 'entry' format in packages
➤ Variant used in matlab and matrix market
➤ Also known as 'triplet format'
➤ Note: order of entries is arbitrary [in matlab: sorted by columns]

Compressed Sparse Row (CSR) format

$$A = \begin{pmatrix} 12. & 0. & 0. & 11. & 0. \\ 10. & 9. & 0. & 8. & 0. \\ 7. & 0. & 6. & 5. & 4. \\ 0. & 0. & 3. & 2. & 0. \\ 0. & 0. & 0. & 0. & 1. \end{pmatrix}$$

AA	JA		IA
12	1		1
11	4		
10	1		3
9	2		
8	4		6
7	1		
6	3		10
5	4		
4	5		12
3	3		
2	4		13
1	5		

➤ IA(j) points to beginning or row j in arrays AA, JA

➤ Related formats: Compressed Sparse Column format, Modified Sparse Row format (MSR).

➤ Used predominantly in Fortran & portable codes [e.g. Metis] – what about C?

CSR (CSC) format - C-style

* CSR: Collection of pointers of rows & array of row lengths

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

$aa[i][*]$ $==$ entries of i-th row (col.);

$ja[i][*]$ $==$ col. (row) indices,

$nzcount[i]$ $==$ number of nonzero elmts in row (col.) i

Data structure used in Csparse [T. Davis' SuiteSparse code]

```
typedef struct cs_sparse
{/* matrix in compressed-column or triplet form */
 int nzmax ; /* maximum number of entries */
 int m ;       /* number of rows */
 int n ;       /* number of columns */
 int *p ;      /* column pointers (size n+1) or
                  col indices (size nzmax) */
 int *i ;      /* row indices, size nzmax */
 double *x ;   /* numerical values, size nzmax */
 int nz ;      /* # of entries in triplet matrix,
                  -1 for compressed-col */
} cs ;
```

➤ Can be used for CSR, CSC, and COO (triplet) storage

➤ Easy to use from Fortran

The Diagonal (DIA) format

$$A = \begin{pmatrix} 1. & 0. & 2. & 0. & 0. \\ 3. & 4. & 0. & 5. & 0. \\ 0. & 6. & 7. & 0. & 8. \\ 0. & 0. & 9. & 10. & 0. \\ 0. & 0. & 0. & 11. & 12. \end{pmatrix}$$

$$DA = \begin{pmatrix} * & 1. & 2. \\ 3. & 4. & 5. \\ 6. & 7. & 8. \\ 9. & 10. & * \\ 11 & 12. & * \end{pmatrix} \qquad IOFF = \boxed{-1 \ 0 \ 2}$$

The Ellpack-Itpack format

$$A = \begin{pmatrix} 1. & 0. & 2. & 0. & 0. \\ 3. & 4. & 0. & 5. & 0. \\ 0. & 6. & 7. & 0. & 8. \\ 0. & 0. & 9. & 10. & 0. \\ 0. & 0. & 0. & 11. & 12. \end{pmatrix}$$

$$AC = \begin{pmatrix} 1. & 2. & 0. \\ 3. & 4. & 5. \\ 6. & 7. & 8. \\ 9. & 10. & 0. \\ 11 & 12. & 0. \end{pmatrix} \qquad JC = \begin{pmatrix} 1 & 3 & 1 \\ 1 & 2 & 4 \\ 2 & 3 & 5 \\ 3 & 4 & 4 \\ 4 & 5 & 5 \end{pmatrix}$$

Block matrices

$$A = \begin{pmatrix} 1. & 2. & 0. & 0. & 3. & 4. \\ 5. & 6. & 0. & 0. & 7. & 8. \\ 0. & 0. & 9. & 10. & 11. & 12. \\ 0. & 0. & 13. & 14. & 15. & 16. \\ 17. & 18. & 0. & 0. & 20. & 21. \\ 22. & 23. & 0. & 0. & 24. & 25. \end{pmatrix}$$

$$AA = \begin{array}{|ccc|} 1.\ 3. & 9.\ 11. & 17.\ 20. \\ 5.\ 7. & 15.\ 13. & 22.\ 24. \\ 2.\ 4. & 10.\ 12. & 18.\ 21. \\ 6.\ 8. & 14.\ 16. & 23.\ 25. \end{array}$$

JA = $\boxed{1\ 5\ 3\ 5\ 1\ 5}$

IA = $\boxed{1\ 3\ 5\ 7}$

➤ Columns of AA hold 2 x 2 blocks. JA(k) = col. index of (1,1) entries of k-th block. FORTRAN: declare as AA(2,2,6)

➤ Can also store the blocks row-wise in AA.

$$AA = \begin{array}{|cccc|} 1. & 5. & 2. & 6. \\ 3. & 7. & 4. & 8. \\ 9. & 15. & 10. & 14. \\ 11. & 13. & 12. & 16. \\ 17. & 22. & 18. & 23. \\ 20. & 24. & 21. & 25. \end{array}$$

JA = $\boxed{1\ 5\ 3\ 5\ 1\ 5}$

IA = $\boxed{1\ 3\ 5\ 7}$

➤ In other words AA is simply transposed

✎ What are the advantages and disadvantages of each scheme?

➤ Block formats are important in many applications..

➤ Also valuable: block structure with variable block size.

Sparse matrices – data structure in C

➤ 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)

➤ How to perform the operation $y = A * x$ in each case?

Matvec – row version

```
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]];
    }
}
```

➤ Uses sparse dot products (sparse SDOTS)

✎ Operation count

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;
  for (i=0; i<n; i++) {
    kr = mata->ma[i];
    ki = mata->ja[i];
    for (k=0; k<mata->nzcount[i]; k++)
      y[ki[k]] += kr[k] * x[i];
  }
}
```

➤ Uses sparse vector combinations (sparse SAXPY)

✎ Operation count

➤ Using the CS data structure from Suite-Sparse:

```c
int cs_gaxpy (cs *A, double *x, double *y) {
 int p, j, n, *Ap, *Ai;
 n = A->n; Ap = A-> p; Ai = A->i; Ax = A->x;
 for (j=0; j<n; j++) {
   for (p=Ap[j]; p<Ap[j+1];p++)
     y[Ai[p]] += Ax[p]*x[j];
 }
return(1)
}
```

Matvec – row version - FORTRAN

```fortran
      subroutine amux (n, x, y, a, ja, ia)
      real*8  x(*), y(*), a(*), t
      integer n, ja(*), ia(*), i, k
c----- row loop
      do 100 i = 1,n
c----- inner product of row i with vector x
        t = 0.0d0
        do 99 k=ia(i), ia(i+1)-1
          t = t + a(k)*x(ja(k))
 99       continue
        y(i) = t
 100  continue
      return
      end
```

Matvec – column version - FORTRAN

```fortran
      subroutine atmux (n, x, y, a, ja, ia)
      real*8 x(*), y(*), a(*)
      integer n, ia(*), ja(*)
      integer i, k
c----- set y to zero
      do 1 i=1,n
        y(i) = 0.0
 1    continue
c----- column loop
      do 100 i = 1,n
c----- sparse saxpy
        do 99 k=ia(i), ia(i+1)-1
          y(ja(k)) = y(ja(k)) + x(i)*a(k)
 99     continue
 100  continue
c
      return
      end
```

Sparse matrices in matlab

✎ Generate a tridiagonal matrix T

✎ Convert T to sparse format

✎ See how you can generate this sparse matrix directly using `sparse`

✎ See how you can use `spconvert` to achieve the same result

✎ What can you observe about the way the triplets of a sparse matrix are ordered?

✎ Important for performance: `spalloc`. See the difference between

```
A = sparse(m,n)   and   A = spalloc(m,n,nzmax)
```