Sparse Matrix Methods
Chapter 6 lecture notes
LU factorization

Tim Davis

2011

Upper bound on fill-in

Theorem (George and Ng, Gilbert and Ng)

If the matrix A *is strong Hall,* R *is an upper bound on the nonzero pattern of* U*. More precisely,* u_{ij} *can be nonzero if and only if* $r_{ij} \neq 0$*.*

Theorem (Gilbert and Ng)

If the matrix A *is strong Hall, and assuming* $a_{kk}^{[k-1]} \neq 0$ *for all* k*, the Householder matrix* V *is an upper bound on the nonzero pattern of* L *obtained with partial pivoting. More precisely,* l_{ij} *can be nonzero if and only if* $v_{ij} \neq 0$*.*

Left-looking LU

$$\begin{bmatrix} L_{11} & & \\ l_{21} & 1 & \\ L_{31} & l_{32} & L_{33} \end{bmatrix} \begin{bmatrix} U_{11} & u_{12} & U_{13} \\ & u_{22} & u_{23} \\ & & U_{33} \end{bmatrix} = \begin{bmatrix} A_{11} & a_{12} & A_{13} \\ a_{21} & a_{22} & a_{23} \\ A_{31} & a_{32} & A_{33} \end{bmatrix},$$

- $L_{11} u_{12} = a_{12}$: a triangular system, solve for u_{12}
- $l_{21} u_{12} + u_{22} = a_{22}$, solve for pivot entry u_{22}
- $L_{31} u_{12} + l_{32} u_{22} = a_{32}$, solve for l_{32}

Left-looking LU

$$\begin{bmatrix} L_{11} & & \\ l_{21} & 1 & \\ L_{31} & 0 & I \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} a_{12} \\ a_{22} \\ a_{32} \end{bmatrix}.$$

- $u_{12} = x_1$
- $u_{22} = x_2$
- $l_{32} = x_3 / u_{22}$

Left-looking LU

```
function [L,U,P] = lu_left (A)
n = size (A,1) ;
P = eye (n) ;
L = zeros (n) ;
U = zeros (n) ;
for k = 1:n
    x = [ L(:,1:k-1) [ zeros(k-1,n-k+1) ; eye(n-k+1) ]] \ (P * A (:,k))
    U (1:k-1,k) = x (1:k-1) ;              % the column of U
    [a i] = max (abs (x (k:n))) ;          % find the pivot row i
    i = i + k - 1 ;
    L ([i k],:) = L ([k i], :) ;           % swap rows i and k of L, P, a
    P ([i k],:) = P ([k i], :) ;
    x ([i k]) = x ([k i]) ;
    U (k,k) = x (k) ;
    L (k,k) = 1 ;
    L (k+1:n,k) = x (k+1:n) / x (k) ;      % divide the pivot column by U(
end
```

```
csn *cs_lu (const cs *A, const css *S, double tol)
{
    cs *L, *U ;
    csn *N ;
    double pivot, *Lx, *Ux, *x,  a, t ;
    int *Lp, *Li, *Up, *Ui, *pinv, *xi, *q, n, ipiv, k, top, p, i, col,
    if (!CS_CSC (A) || !S) return (NULL) ;          /* check inputs */
    n = A->n ;
    q = S->q ; lnz = S->lnz ; unz = S->unz ;
    x = cs_malloc (n, sizeof (double)) ;            /* get double works
    xi = cs_malloc (2*n, sizeof (int)) ;            /* get int workspac
    N = cs_calloc (1, sizeof (csn)) ;               /* allocate result
    if (!x || !xi || !N) return (cs_ndone (N, NULL, xi, x, 0)) ;
    N->L = L = cs_spalloc (n, n, lnz, 1, 0) ;       /* allocate result
    N->U = U = cs_spalloc (n, n, unz, 1, 0) ;       /* allocate result
    N->pinv = pinv = cs_malloc (n, sizeof (int)) ;  /* allocate result
    if (!L || !U || !pinv) return (cs_ndone (N, NULL, xi, x, 0)) ;
    Lp = L->p ; Up = U->p ;
    for (i = 0 ; i < n ; i++) x [i] = 0 ;           /* clear workspace
for (i = 0 ; i < n ; i++) pinv [i] = -1 ;       /* no rows pivotal yet
for (k = 0 ; k <= n ; k++) Lp [k] = 0 ;         /* no cols of L yet */
lnz = unz = 0 ;
for (k = 0 ; k < n ; k++)       /* compute L(:,k) and U(:,k) */
{
    /* --- Triangular solve -------------------------------------------
    Lp [k] = lnz ;              /* L(:,k) starts here */
    Up [k] = unz ;              /* U(:,k) starts here */
    if ((lnz + n > L->nzmax && !cs_sprealloc (L, 2*L->nzmax + n)) ||
        (unz + n > U->nzmax && !cs_sprealloc (U, 2*U->nzmax + n)))
    {
        return (cs_ndone (N, NULL, xi, x, 0)) ;
    }
    Li = L->i ; Lx = L->x ; Ui = U->i ; Ux = U->x ;
    col = q ? (q [k]) : k ;
    top = cs_spsolve (L, A, col, xi, x, pinv, 1) ;  /* x = L\A(:,col) *
```

```
/* --- Find pivot --------------------------------------------------- *
ipiv = -1 ;
a = -1 ;
for (p = top ; p < n ; p++)
{
    i = xi [p] ;                /* x(i) is nonzero */
    if (pinv [i] < 0)           /* row i is not yet pivotal */
    {
        if ((t = fabs (x [i])) > a)
        {
            a = t ;             /* largest pivot candidate so far */
            ipiv = i ;
        }
    }
    else                        /* x(i) is the entry U(pinv[i],k) */
    {
        Ui [unz] = pinv [i] ;
        Ux [unz++] = x [i] ;
    }
}
if (ipiv == -1 || a <= 0) return (cs_ndone (N, NULL, xi, x, 0)) ;
if (pinv [col] < 0 && fabs (x [col]) >= a*tol) ipiv = col ;
```

```
/* --- Divide by pivot ---------------------------------------------- *
pivot = x [ipiv] ;          /* the chosen pivot */
Ui [unz] = k ;              /* last entry in U(:,k) is U(k,k) */
Ux [unz++] = pivot ;
pinv [ipiv] = k ;           /* ipiv is the kth pivot row */
Li [lnz] = ipiv ;           /* first entry in L(:,k) is L(k,k) = 1 */
Lx [lnz++] = 1 ;
for (p = top ; p < n ; p++) /* L(k+1:n,k) = x / pivot */
{
    i = xi [p] ;
    if (pinv [i] < 0)       /* x(i) is an entry in L(:,k) */
    {
        Li [lnz] = i ;      /* save unpermuted row in L */
        Lx [lnz++] = x [i] / pivot ;    /* scale pivot column */
    }
    x [i] = 0 ;             /* x [0..n-1] = 0 for next k */
}
```

```
 }
 /* --- Finalize L and U ----------------------------------------------
    Lp [n] = lnz ;
    Up [n] = unz ;
    Li = L->i ;                     /* fix row indices of L for final p
    for (p = 0 ; p < lnz ; p++) Li [p] = pinv [Li [p]] ;
    cs_sprealloc (L, 0) ;           /* remove extra space from L and U
    cs_sprealloc (U, 0) ;
    return (cs_ndone (N, NULL, xi, x, 1)) ;     /* success */
}
```