**Week 1: Friday, Aug 30**

# Logistics

1. Let me know if you're not on CMS (e.g. if you just joined).
2. Get started with the reading (and the HW)! Today is partly out of sequence – see GVL 11.1 for a discussion of general sparse matrices.

# Nonzero structure

One important type of structure in matrices involves where there can be nonzeros. We started talking about this last time – I called this the "graph" structure of a matrix, in contrast to types of structure like orthogonality or symmetry.

For example, a *lower triangular* $L$ matrix satisfies $l_{ij} = 0$ for $j > i$. If we put crosses where the can be nonzeros, we have

$$L = \begin{bmatrix} \times & & & & \\ \times & \times & & & \\ \times & \times & \times & & \\ \vdots & \vdots & \vdots & \ddots & \\ \times & \times & \times & \dots & \times \end{bmatrix}.$$

Similarly, an upper triangular matrix $U$ satisfies $u_{ij} = 0$ for $j < i$. A *banded* matrix has zeros outside some distance of the diagonal; that is, $B$ is banded with lower bandwidth $p$ and upper bandwidth $q$ if $b_{ij} = 0$ for $j < i - p$ and $j > i + q$. For example, a matrix with lower bandwidth 1 and upper bandwidth 2 has this nonzero structure:

$$B = \begin{bmatrix} \times & \times & \times & & & & & \\ \times & \times & \times & \times & & & & \\ & \times & \times & \times & \times & & & \\ & & \times & \times & \times & \times & & \\ & & & \ddots & \ddots & \ddots & \ddots & \\ & & & & \times & \times & \times & \times \\ & & & & & \times & \times & \times \\ & & & & & & \times & \times \end{bmatrix}.$$

A banded matrix with $b+q \ll n$ is a special case of a *sparse* matrix in which most of the elements are zero.

Why do we care about these matrix structures? One reason is that we can use these structures to improve the performance of matrix multiplication. If nnz is the number of nonzeros in a matrix, then matrix-vector multiplication can be written to take $O(\text{nnz})$ time. We can also represent the matrix using $O(\text{nnz})$ storage. Another reason is that some structures are easy to compute with. For example, if we want to solve a linear system with a triangular matrix, we can do so easily using back-substitution; and if we want the eigenvalues of a triangular matrix, we can just read the diagonal.

# General sparse matrices

What if we have relatively few nonzeros in a matrix, but they are not in a narrow band about the origin or some other similarly regular structure? In this case, we would usually represent the matrix by a general sparse format. The format MATLAB uses internally is *compressed sparse columns*. In compressed sparse column format, we keep a list of the nonzero entries and their corresponding rows, stored one column after the other; and a list of pointers saying where the data for each column starts. For example, consider the matrix

$$A = \begin{bmatrix} a_{11} & 0 & 0 & a_{14} \\ 0 & a_{22} & 0 & 0 \\ a_{31} & 0 & a_{33} & 0 \\ 0 & a_{42} & 0 & a_{44} \end{bmatrix}.$$

In compressed sparse column form, we would have

| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| entries | = | $a_{11}$ | $a_{31}$ | $a_{22}$ | $a_{42}$ | $a_{33}$ | $a_{14}$ | $a_{44}$ |
| rows | = | 1 | 3 | 2 | 4 | 3 | 1 | 4 |
| | | | | | | | | |
| column pointers | = | 1 | 3 | 6 | 8 | | | |

The last entry in the column pointer array tells where the *end* of the last column is. If you use MATLAB, the details of this data structure are hidden. To get a sparse representation of a matrix $A$ is as simple as writing

```
As = sparse(A);
```

# Beyond nonzero structure

Consider the matrices whose elements are as follows.

1. $a_{ij}^{(1)} = x_i y_j$ for vectors $x, y \in \mathbb{R}^n$.
2. $a_{ij}^{(2)} = x_i + y_j$.
3. $a_{ij}^{(3)} = 1$ if $i + j$ even, 0 otherwise.
4. $a_{ij}^{(4)} = \delta_{ij} + x_i y_j$.
5. $a_{ij}^{(5)} = \mu^{|i-j|}$.

The questions:

1. How can we write a fast ($O(n)$) algorithm to compute $v = Au$ for each of these matrices?
2. Given general nonsingular $B$ and $C$, can we write a fast algorithm to multiply by $\hat{A} = BAC$ in $O(n)$ time (assuming some precomputation is allowed)?
3. Given a general nonsingular $B$, can we write a fast multiplication algorithm for $\tilde{A} = B^{-1}AB$?

The first three matrices are all low-rank. The first matrix can be written as an outer product $A^{(1)} = xy^T$; the second matrix is $A^{(2)} = xe^T - ey^T$, where $e$ is the vector of all ones; and the third matrix is $A^{(3)} = e_{\text{odd}} e_{\text{odd}}^T + e_{\text{even}} e_{\text{even}}^T$, where $e_{\text{odd}}$ is the vector with ones in all odd-index entries and zeros elsewhere, and $e_{\text{even}}$ is the vector with ones in all even-index entries. We can write efficient MATLAB functions to multiply by each of these matrices:

```
% Compute v = A1*u = (x*y')*u
function v = multA1(u,x,y)
v = x*(y'*u);

% Compute v = A2*u
function v = multA2(u,x,y)
v = x*sum(u)+y'*u;
```

```
% Compute v = A3*u
function v = multA3(u);
v = zeros(length(u),1);
v(1:2:end) = sum(u(1:2:end));
v(2:2:end) = sum(u(2:2:end));
```

Note that all we are *really* using in these routines is the fact that the underlying matrices are low rank. The rank is a property of the underlying linear transformation, independent of basis; that is, $\mathrm{rank}(A) = \mathrm{rank}(BAC)$ for any nonsingular $B$ and $C$. So we can still get a fast matrix multiply for $\hat{A}^{(1)} = BA^{(1)}C$, for example, by precomputing $\hat{x} = Bx$ and $\hat{y} = C^T y$ and then writing $\hat{A}^{(1)} = \hat{x}\hat{y}^T$.

The fourth matrix is an identity plus a low-rank matrix: $A^{(4)} = I + xy^T$. This structure is destroyed if we change bases independently for the domain and range space (i.e., $BA^{(4)}C$ has no useful structure), but it is preserved when we make the same change of basis for both the domain and range (i.e., $B^{-1}A^{(4)}B = I + \hat{x}\hat{y}^T$, where $\hat{x} = B^{-1}x$ and $\hat{y} = B^T y$.

The fifth matrix is much more interesting. Though the matrix does not have lots of zeros and is not related in an obvious way to something with low rank, there is nonetheless enough structure for us to do a fast multiply. Writing each entry of $v = A^{(5)}u$ in component form, we have

$$v_j = \sum_{i=1}^{n} \mu^{|i-j|} u_i = \left( \sum_{i=1}^{j-1} \mu^{j-i} u_i \right) + \left( \sum_{i=j}^{n} \mu^{i-j} u_i \right) = r_j + l_j.$$

where $r_j$ and $l_j$ refer to the parts of the dot product to the right and left of the main diagonal, respectively. Now notice that

$$\begin{aligned} r_1 &= 0 \\ l_n &= u_n \\ r_{j+1} &= \mu(r_j + u_j) \\ l_j &= \mu l_{j+1} + u_j. \end{aligned}$$

The following MATLAB code runs to compute the matrix-vector product with $A^{(6)}$ in $O(n)$ time:

```
% Compute v=A5*u
```

```
function v = multA5(u,mu);
n = length(u);

% Run the recurrence for r forward
r = zeros(n,1);
for j = 1:n-1
  r = (r+u(j)) * mu;
end

% Run the recurrence for l backward
l = zeros(n,1);
l(n) = u(n);
for j = n-1:-1:1
  l(j) = l(j+1)*mu + u(j);
end

v = l+r;
```

There is no fast multiply for $B^{-1}A^{(5)}B$, let alone for $BA^{(5)}C$.

# More questions

1. Ordinary multiplication of two square matrices usually involves $2n^3$ floating-point operations (additions and multiplications) – see discussion in the book. What is the complexity of multiplying two upper triangular matrices?

2. Suppose $A$ and $B$ are upper triangular. Show that $AB$ is triangular.

3. Suppose `i`, `j`, and `aij` are the row index, column index, and element value of each nonzero in a sparse matrix $A$. Describe how you would multiply $A$ by a vector.

4. Describe an $O(n)$ matrix-vector multiply routine for a square matrix whose entries are
$$a_{ij} = \begin{cases} \alpha_i \beta_j, & i \leq j \\ 0, & \text{otherwise} \end{cases}$$