Notes for 2016-01-29

Matrices

From your linear algebra background, you should know a matrix as a representation of a linear map. A matrix can *also* represent a bilinear function mapping two vectors into the real numbers (or complex numbers for complex vector spaces):

$$(v, w) \rightarrow w^* A v.$$

Symmetric matrices also represent *quadratic forms* mapping vectors to real numbers

$$\phi(v) = v^* A v.$$

We say a symmetric matrix is A is *positive definite* if the corresponding quadratic form is positive definite, i.e.

$$v^* A v \geq 0 \text{ with equality iff } v = 0.$$

We will talk more about matrices as representations of linear maps, bilinear forms, and quadratic forms in the next lecture.

Many "rookie mistakes" in linear algebra involve forgetting ways in which matrices differ from scalars:

- Not all matrices are square.
- Not all matrices are invertible (even nonzero matrices can be singular).
- Matrix multiplication is associative, but not commutative.

Don't forget these facts!

In matrix computations, we deal not only with the linear algebraic perspective on a matrix, but also with concrete representations. We usually represent a dense matrix as an array of numbers that are stored sequentially in computer memory. But we may use different representations depending on what we want to do. Often our goal is to evaluate some expression involving a matrix, such as evaluating a linear map or a quadratic form or solving a linear system. In these cases, we might prefer other different representations that take advantage of a particular problem's structure.

Twelve Commandments

When Charlie Van Loan teaches matrix computations, he states "twelve commandments" of matrix manipulations:

1. Matrix $\times$ vector = linear combination of matrix columns.
2. Inner product = sum of products of corresponding elements.
3. Order of operations is important to performance.
4. Matrix $\times$ diagonal = scaling of the matrix columns.
5. Diagonal $\times$ matrix = scaling of the matrix rows.
6. Never form an explicit diagonal matrix.
7. Never form an explicit rank one matrix.
8. Matrix $\times$ matrix = collection of matrix-vector products.
9. Matrix $\times$ matrix = collection of dot products.
10. Matrix $\times$ matrix = sum of rank one matrices.
11. Matrix $\times$ matrix $\implies$ linearly combine rows from the second matrix.
12. Matrix $\times$ matrix $\implies$ linearly combine columns from the first matrix.

I might add more, but twelve is a nicer-sounding number than thirteen or fourteen, and fifteen is clearly too many.

Block matrices

We often partition matrices into submatrices of different sizes. For example, we might write

$$\begin{bmatrix} a_{11} & a_{12} & b_1 \\ a_{21} & a_{22} & b_2 \\ c_1 & c_2 & d \end{bmatrix} = \begin{bmatrix} A & b \\ c^T & d \end{bmatrix}, \text{ where } A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}, b = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}, c = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}.$$

We can manipulate block matrices in much the same way we manipulate ordinary matrices; we just need to remember that matrix multiplication does not commute.

Standard matrices

We will see a handful of standard matrices throughout the course:

- The zero matrix (written 0 – we distinguish from the scalar zero by context). In MATLAB: `zeros(m,n)`.
- The identity matrix I. In MATLAB: `eye(n)`.
- Diagonal matrices, usually written D. In MATLAB: `diag(d)` where `d` is the vector of diagonal entries.
- Permutation matrices, usually written P or Π (but sometimes written Q if P is already used). These are square 0-1 matrices with exactly one 1 in each row and column. They look like the identity matrix with the columns permuted. In MATLAB, I would usually write `P = eye(n); P = P(:,idx)` where `idx` is an index vector such that data at index `idx(k)` in a vector v gets mapped to index `k` in Pv.

Though I have given MATLAB commands to construct these matrices, we usually would not actually create them explicitly except as a step in creating another matrix (see Van Loan's sixth commandment!).

Matrix shapes and structures

In linear algebra, we talk about different matrix structures. For example:

- $A \in \mathbb{R}^{n\times n}$ is *nonsingular* if the inverse exists; otherwise it is *singular*.
- $Q \in \mathbb{R}^{n\times n}$ is *orthogonal* if $Q^TQ = I$.
- $A \in \mathbb{R}^{n\times n}$ is *symmetric* if $A = A^T$.
- $S \in \mathbb{R}^{n\times n}$ is *skew-symmetric* if $S = -S^T$.
- $L \in \mathbb{R}^{n\times m}$ is *low rank* if $L = UV^T$ for $U \in \mathbb{R}^{n\times k}$ and $V \in \mathbb{R}^{m\times k}$ where $k \ll \min(m,n)$.

These are properties of an underlying linear map or quadratic form; if we write a different matrix associated with an (appropriately restricted) change of basis, it will also have the same properties.

In matrix computations, we also talk about the *shape* (nonzero structure) of a matrix. For example:

- D is *diagonal* if $d_{ij} = 0$ for $i \neq j$.
- T is *tridiagonal* if $t_{ij} = 0$ for $i \notin \{j-1, j, j+1\}$.
- U is *upper triangular* if $u_{ij} = 0$ for $i > j$ and *strictly upper triangular* if $u_{ij} = 0$ for $i \geq j$ (lower triangular and strictly lower triangular are similarly defined).
- H is *upper Hessenberg* if $h_{ij} = 0$ for $i > j + 1$.
- B is *banded* if $b_{ij} = 0$ for $|i - j| > \beta$.
- S is *sparse* if most of the entries are zero. The position of the nonzero entries in the matrix is called the *sparsity structure*.

We often represent the shape of a matrix by marking where the nonzero elements are (usually leaving empty space for the zero elements); for example:

$$\text{Diagonal} \begin{bmatrix} \times & & & & \\ & \times & & & \\ & & \times & & \\ & & & \times & \\ & & & & \times \end{bmatrix} \qquad \text{Tridiagonal} \begin{bmatrix} \times & \times & & & \\ \times & \times & \times & & \\ & \times & \times & \times & \\ & & \times & \times & \times \\ & & & \times & \times \end{bmatrix}$$

$$\text{Triangular} \begin{bmatrix} \times & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & & \times & \times & \times \\ & & & \times & \times \\ & & & & \times \end{bmatrix} \qquad \text{Hessenberg} \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & & \times & \times & \times \\ & & & \times & \times \end{bmatrix}$$

We also sometimes talk about the *graph* of a (square) matrix $A \in \mathbb{R}^{n \times n}$: if we assign a node to each index $\{1, \ldots, n\}$, an edge (i, j) in the graph corresponds to $a_{ij} \neq 0$. There is a close connection between certain classes of graph algorithms and algorithms for factoring sparse matrices or working with different matrix shapes. For example, the matrix A can be permuted so that PAP^T is upper triangular iff the associated directed graph is acyclic.

The shape of a matrix (or graph of a matrix) is not intrinsically associated with a more abstract linear algebra concept; apart from permutations, sometimes, almost any change of basis will completely destroy the shape.

Data sparsity and fast matrix-vector products

We say a matrix $A \in \mathbb{R}^{n \times m}$ is *data sparse* if we can represent it with far fewer than nm parameters. For example,

- Sparse matrices are data sparse – we only need to explicitly know the positions and values of the nonzero elements.
- A rank one matrix is data sparse: if we write it as an outer product $A = uv^T$, we need only $n + m$ parameters (we can actually get away with only $n + m - 1$, but usually wouldn't bother). More generally, low-rank matrices are data sparse.
- A Toeplitz matrix (constant diagonal entries) is data sparse.
- The upper or lower triangle of a low rank matrix is data sparse.

Sums and products of a few data sparse matrices will remain data sparse.

Data sparsity is useful for several reasons. If we are interested in the matrix itself, data sparsity lets us save storage. If we are interested in multiplying by the matrix, or solving linear systems involving the matrix, data sparsity lets us write fast algorithms. For example,

- We can multiply a sparse matrix A times a vector in $O(\text{nnz})$ time in general, where nnz is the number of nonzeros in the matrix.
- If $A = uv^T$ is rank one, we can compute $y = Ax$ in $O(n + m)$ time by first computing $\alpha = v^T x$ (a dot product, $O(m)$ time), then $y = \alpha u$ (a scaling of the vector u, $O(n)$ time).
- We can multiply a square Toeplitz matrix by a vector in $O(n \log n)$ time using fast Fourier transforms.
- We can multiply the upper or lower triangle of a square low rank matrix by a vector in $O(n)$ time with a simple loop (left as an exercise).

In much modern research on numerical linear algebra, sparsity or data sparsity is the name of the game. Few large problems are unstructured; and where there is structure, one can usually play games with data sparsity.