San José State University

Math 250: Mathematical Data Visualization

Matrix Computing in MATLAB

Dr. Guangliang Chen

Outline of the lecture:

- Ordinary vector and matrix operations in MATLAB
- Coding techniques for dealing with large matrices
- In-class demonstrations
- Additional learning

Hw2 (see Canvas)

What is MATLAB (MATrix LABoratory)?

MATLAB is commercial software developed by Mathworks.

It is a popular language in applied math and engineering:

- Matrix computing
- Numerical optimization
- Signal and image processing
- Data plotting/visualization

Why MATLAB?

- Simple, flexible and easy to use
- Efficient and robust for linear algebra operations
- High quality data plotting
- Very thorough documentation with lots of examples
- The statistics and machine learning toolbox has all that we need
- SJSU now has a campus wide license (free for students)[1]

[1] https://www.mathworks.com/academia/tah-portal/san-jose-state-university-31511582.html

My strategies for teaching MATLAB as a computing tool

- Focus on what is essential for this course (i.e., matrix operations, and data plotting)
- Example-based
- Emphasize on good practices in MATLAB programming
  - simplicity
  - efficiency
  - readability

Creating vectors in Matlab

- Row vector: $a = [1\ 2\ 3\ 4\ 5\ 6]$; or $a = [1, 2, 3, 4, 5, 6]$; or $a = 1 : 6$;
- Column vector: $a = [1; 2; 3; 4; 5; 6]$; or $a = (1 : 6)'$;
- Zero/one/random vectors:
  $a = zeros(1, 6);\ b = ones(6, 1);\ r = rand(1, 10);$
- Linear: $a = linspace(0, 1, 11)$; or $a = 0 : 0.1 : 1$;
- Periodic: $a = repmat(1 : 3, 1, 5)$; $b = repelem(1 : 3, 5)$;
  $c = repmat((1 : 3)', 5, 1)$;

Basic vector functions in Matlab

- $length(a)$, $numel(a)$
- $sum(a), mean(a), median(a), min(a), max(a), prod(a)$
- $cumsum(a), cumprod(a)$
- $norm(a), norm(a, 1), norm(a, \textit{Inf})$
- $sort(a), sort(a, \textit{'descend'}), find(a > 0)$
- $a.$ˆ2; $1./a$; $sqrt(a)$, where a is a positive vector

Creating matrices in Matlab

- Direct definition: $A = [1\ 2\ 3;\ 4\ 5\ 6;\ 7\ 8\ 9];$
- By rearranging a vector: $A = reshape(1:9, 3, 3);$
- By replicating a vector: $A = repmat(1:3, 5, 1);$
  $B = repmat((1:3)', 1, 5)$
- Special matrices: $O = zeros(5,6);\ J = ones(6,6);\ I = eye(6);$
  $R = rand(10,10);\ D = diag(1:5);$

Basic matrix functions in Matlab

- $size(A)$ and $numel(A)$. The latter is same as $prod(size(A))$
- $sum(A, 1), sum(A, 2), min(A, [], 1), max(A, [], 2)$
- $trace(A)$, same as $sum(diag(A))$
- $eig(A), eig(A, B), det(A), rank(A), inv(A)$ (slow and unreliable for large matrices)
- $eigs(A, K)$ (largest K eigenvalues of A)
- $eigs(A, B, K)$ (largest K generalized eigenvalues of (A, B))

Manipulating a single matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$

- $A.\hat{}2$ (entrywise square), $A\hat{}2$ (square), A' (transpose)
- Row sums: $sum(A, 2)$, but do not use $A * ones(n, 1)$
- Column sums: $sum(A, 1)$ or $sum(A)$ but not $ones(1, m) * A$
- Overall sum: $sum(A, \text{'all'}), sum(sum(A)), sum(A(:))$, but not $ones(1, m) * A * ones(n, 1)$
- ℓ_1 row normalization: $A./repmat(sum(A, 2), 1, n)$, or $A./sum(A, 2)$
- ℓ_2 row normalization: $A./sqrt(sum(A.\hat{}2, 2))$

Computational complexity

In coding, there is often more than one way to implement an operation or algorithm. It matters tremendously HOW you implement it.

For the purpose of efficient coding, we need to know how to analyze the complexity of an operation/algorithm.

There are two kinds of complexity that need to be analyzed:

- **Space/memory complexity**
- **Time/speed complexity**

For general matrices, we suppose each element takes the same amount of space. Thus, the memory required by a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ is $\mathcal{O}(mn)$.

For a sparse matrix, the memory requirement would be just the number of nonzeros in it.

For most time, we will need to determine the time complexity carefully, which is defined as the total number of arithmetic operations $(+, -, \times, /)$ required by the operation.[2]

We will try to identify the order of the time/space complexity, rather than finding the exact amount (which may be too hard/slow to do).

[2] In fact, each of the $+, -, \times$ operations takes 1 unit of time but division takes about 8. We ignore the difference for simplicity.

Time complexity of common linear algebra operations

Let $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ and $\mathbf{A} \in \mathbb{R}^{m\times n}, \mathbf{B} \in \mathbb{R}^{n\times p}, \mathbf{S} \in \mathbb{R}^{n\times n}$. Then

- Summing up the entries of $\mathbf{x}$: $\mathcal{O}(n)$ ($n-1$ additions);
- Dot product $\mathbf{x}^T\mathbf{y}$: $\mathcal{O}(n)$ ($2n-1$ operations in total: n multiplications and $n-1$ additions). This implies that calculating the norm of $\mathbf{x}$, $\|\mathbf{x}\|^2 = \mathbf{x}^T\mathbf{x}$, also has $\mathcal{O}(n)$ complexity.
- $\mathbf{Ax}$: $\mathcal{O}(mn)$ (m dot product operations)
- $\mathbf{AB}$: $\mathcal{O}(mnp)$. In particular, $\mathbf{S}^2$ takes $\mathcal{O}(n^3)$ time.
- $det(\mathbf{S}), eig(\mathbf{S}),$ and $inv(\mathbf{S})$: $\mathcal{O}(n^3)$

Order of multiplication can matter a lot

When multiplying several matrices and a vector, always perform matrix-vector multiplication (and avoid matrix multiplication).

For example, for any $\mathbf{A} \in \mathbb{R}^{m\times n}, \mathbf{B} \in \mathbb{R}^{n\times p}$, and $\mathbf{x} \in \mathbb{R}^{p}$, we have

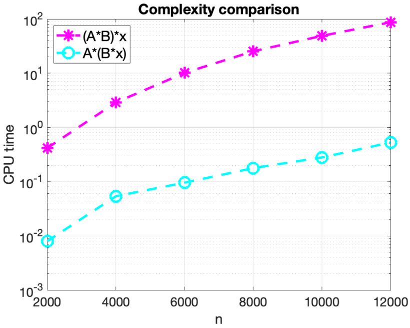
$$(\mathbf{AB})\mathbf{x} = \mathbf{A}(\mathbf{Bx})$$

Although mathematically equivalent, the right-hand side consists of two matrix-vector multiplications and is much faster!

- $(\mathbf{AB})\mathbf{x}$: $\mathcal{O}(mnp + mp)$ complexity
- $\mathbf{A}(\mathbf{Bx})$: $\mathcal{O}(mn + np)$ complexity

Simulation study

We generate matrices $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n\times n}$ and vector $\mathbf{x} \in \mathbb{R}^n$ by sampling entries uniformly at random from $(0,1)$, for each $n = 2000, 4000, \ldots, 12000$, to compare the CPU times needed by the two operations, $(\mathbf{AB})\mathbf{x}$ and $\mathbf{A}(\mathbf{Bx})$.



The plot shows that $\mathbf{A}(\mathbf{Bx})$ is much faster than $(\mathbf{AB})\mathbf{x}$ for all n.

Diagonal matrices are essentially vectors

Let $\mathbf{A} = \mathrm{diag}(\mathbf{a}) \in \mathbb{R}^{n\times n}$ and $\mathbf{B} \in \mathbb{R}^{n\times p}$. Then

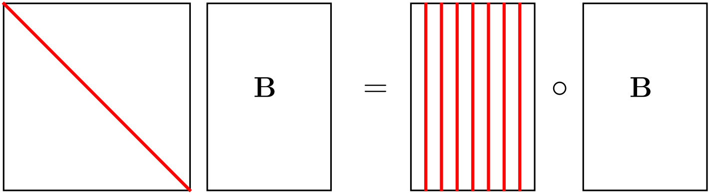
$$\underbrace{\mathbf{A}}_{\text{diagonal}}\mathbf{B} = \begin{pmatrix} a_1 & & \\ & \ddots & \\ & & a_n \end{pmatrix} \begin{pmatrix} B_1 \\ \vdots \\ B_n \end{pmatrix} = \begin{pmatrix} a_1 B_1 \\ \vdots \\ a_n B_n \end{pmatrix}$$

We may implement the matrix product via the Hadamard product:

$$\underbrace{\mathbf{A}}_{n\times n}\underbrace{\mathbf{B}}_{n\times p} = \underbrace{[\mathbf{a}\ldots\mathbf{a}]}_{p\text{ copies}} \circ \mathbf{B}.$$

Matrix Computing in MATLAB

The former takes $\mathcal{O}(n^2 p)$ operations, while the latter takes only $\mathcal{O}(np)$ operations, which is one magnitude faster.



For example,

$$\begin{pmatrix} -1 & & \\ & 0 & \\ & & 1 \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 & 10 \\ 4 & 5 & 6 & 10 \\ 7 & 8 & 9 & 10 \end{pmatrix} = \begin{pmatrix} -1 & -1 & -1 & -1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix} \circ \begin{pmatrix} 1 & 2 & 3 & 10 \\ 4 & 5 & 6 & 10 \\ 7 & 8 & 9 & 10 \end{pmatrix}$$

Sparse matrices

Sparse matrices are very efficient in computing, and all the previously introduced matrix functions apply to them readily.

$A = zeros(5,5);$
$A(1,1) = 2; A(1,2) = -1; A(2,1) = 1;$
$S = sparse(A);$ % $A = full(S)$

$density = nnz(S)/numel(S);$

In-class demonstrations

See sample scripts from instructor

Some (early) coding advice

- Initialize your variables, e.g., $A = zeros(100, 10)$
- Set constant variables to increase readability: $maxIterations = 30$
- Avoid for loops unless necessary (use matrix operations instead)
- Add brief documentation to remind your reader and also yourself
- Use 3D arrays in clever ways
- Smart indexing is important
- Careful (and creative) design is the key

Summary and beyond

Summary: MATLAB is a powerful, convenient computing tool for this course.

Further learning: See course webapge[3]

Next time: Data plotting and visualization in 3D

[3] https://www.sjsu.edu/faculty/guangliang.chen/Math250.html