AMS526: Numerical Analysis I
(Numerical Linear Algebra)
Lecture 1: Course Overview;
Matrix Multiplication

Xiangmin Jiao

Stony Brook University
Outline

1. Course Overview
2. The Language of Matrix Computations
3. Algorithms and Efficiency
Course Description

- What is numerical linear algebra?
  - Solving linear algebra problems using efficient algorithms on computers
- Required textbook (also an excellent reference book)
- Supplementary textbook (a very insightful book)
Prerequisite

- **Prerequisite/Co-requisite:**
  - AMS 510 (linear algebra portion) or equivalent undergraduate-level linear algebra course. Familiarity with following concepts is assumed: Vector spaces, Gaussian elimination, Gram-Schmidt orthogonalization, and eigenvalues/eigenvectors
  - AMS 595 (co-requisite for students without programming experience)
  - This **MUST NOT** be your first course in linear algebra, or you **will** get lost

- To review fundamental concepts of linear algebra, see textbook such as
Why Learn Numerical Linear Algebra?

- Numerical linear algebra is foundation of scientific computations
- Many problems ultimately reduce to linear algebra concepts or algorithms, either analytical or computational
- Examples: Finite-element analysis, data fitting, PageRank (Google)

The $25,000,000,000$ Eigenvector: The Linear Algebra behind Google*

Kurt Bryan†
Tanya Leise†

Abstract. Google’s success derives in large part from its PageRank algorithm, which ranks the importance of web pages according to an eigenvector of a weighted link matrix. Analysis of the PageRank formula provides a wonderful applied topic for a linear algebra course. Instructors may assign this article as a project to more advanced students or spend one or two lectures presenting the material with assigned homework from the exercises. This material also complements the discussion of Markov chains in matrix algebra. Maple and Mathematica files supporting this material can be found at www.rose-hulman.edu/~bryan.

- Focus of this course: Fundamental concepts, efficiency and stability of algorithms, and programming
Course Outline

- Basic linear algebra concepts and algorithms (1.5 weeks)
- Gaussian elimination and its variants (2 weeks)
- Sensitivity and stability (1.5 weeks)
- Linear least squares problem (1.5 weeks)
- Eigenvalues, eigenvectors, and SVD (4 weeks)
- Iterative methods for linear systems (2 weeks)

Course webpage:
http://www.ams.sunysb.edu/~jiao/teaching/ams526_fall14

Note: Course schedule online is tentative and is subject to change.
Course Policy

- **Assignments (written or programming)**
  - Assignments are due in class one to two weeks after assigned
  - You can discuss course materials and homework problems with others, but you must write your answers completely independently
  - Do NOT copy solutions from any source. Do NOT share your solutions with others

- **Exams and tests**
  - All exams are closed-book
  - However, one-page cheat sheet is allowed

- **Grading**
  - Assignments: 30%
  - Two midterm exams: 40%
  - Final exam: 30%
Outline

1 Course Overview

2 The Language of Matrix Computations

3 Algorithms and Efficiency
Matrices and Vectors

- Denote vector space of all $m$-by-$n$ real matrices by $\mathbb{R}^{m \times n}$.

$$A \in \mathbb{R}^{m \times n} \iff A = (a_{ij}) = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{bmatrix}$$

- Denote vector space of all real $n$-vectors by $\mathbb{R}^n$, or $\mathbb{R}^{n \times 1}$

$$x \in \mathbb{R}^n \iff x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

- Transposition ($\mathbb{R}^{m \times n} \rightarrow \mathbb{R}^{n \times m}$): $C = A^T \Rightarrow c_{ij} = a_{ji}$

- Row vectors are transpose of column vectors and are in $\mathbb{R}^{1 \times n}$
Matrix Operations

- **Addition and subtraction** \((\mathbb{R}^{m \times n} \times \mathbb{R}^{m \times n} \rightarrow \mathbb{R}^{m \times n})\):
  \[ C = A \pm B \Rightarrow c_{ij} = a_{ij} \pm b_{ij} \]

- **Scalar-matrix multiplication or scaling** \((\mathbb{R} \times \mathbb{R}^{m \times n} \rightarrow \mathbb{R}^{m \times n})\):
  \[ C = \alpha A \Rightarrow c_{ij} = \alpha a_{ij} \]

- **Matrix-matrix multiplication/product** \((\mathbb{R}^{m \times p} \times \mathbb{R}^{p \times n} \rightarrow \mathbb{R}^{m \times n})\):
  \[ C = A \cdot B \Rightarrow c_{ij} = \sum_{k=1}^{p} a_{ik} b_{kj} \text{ (denoted by } A \ast B \text{ in MATLAB)} \]

  Each operation also applies to vectors. In particular,

  - **Inner product** is row vector times column vector, i.e., \( c = x^T y \)
    (it is called **dot product** in vector calculus and denoted as \( x \cdot y \))
  - **Outer product** is column vector times row vector, i.e., \( C = x y^T \)
    (it is a special case of Kronecker product and denoted as \( x \otimes y \))

- **Elementwise multiplication and division** \((\mathbb{R}^{m \times n} \times \mathbb{R}^{m \times n} \rightarrow \mathbb{R}^{m \times n})\)
  
  - \( C = A \ast B \Rightarrow c_{ij} = a_{ij} b_{ij} \)
  - \( C = A./B \Rightarrow c_{ij} = a_{ij} / b_{ij} \), where \( b_{ij} \neq 0 \)

- **Matrix inversion** \((A^{-1})\) and **division** \((A/B \text{ and } A \backslash B)\) to be defined later
Notation of Matrices and Vectors

- **Matrix notation**
  - Capital letters (e.g., $A$, $B$, $\Delta$, etc.) for matrices
  - Corresponding lower case with subscript $ij$ (e.g., $a_{ij}$, $b_{ij}$, $\delta_{ij}$) for $(i, j)$ entry; sometimes with notation $[A]_{ij}$ or $A(i, j)$

- **Vector notation**
  - Lowercase letters (e.g., $x$, $y$, etc.) for vectors
  - Corresponding lower case with subscript $i$ for $i$th entry (e.g., $x_i$, $y_i$)

- Lower-case letters for scalars (e.g., $c$, $s$, $\alpha$, $\beta$, etc.)
- Some suggest using boldface lowercase (e.g., $\mathbf{x}$) for vectors, regular lowercase (e.g. $c$) for scalars, and boldface uppercase for matrices

- A matrix is a collection of column vectors or row vectors

\[
A \in \mathbb{R}^{m \times n} \iff A = [c_1 | c_2 | \ldots | c_n], \quad c_k \in \mathbb{R}^m
\]

\[
A \in \mathbb{R}^{m \times n} \iff A = \begin{bmatrix} r_1^T \\ \vdots \\ r_m^T \end{bmatrix}, \quad r_k \in \mathbb{R}^n
\]
Complex Matrices

- Occasionally, complex matrices are involved
- Vector space of $m$-by-$n$ complex matrices is designated by $\mathbb{C}^{m \times n}$
  - Scaling, addition, multiplication of complex matrices correspond exactly to real case
  - If $A = B + iC \in \mathbb{C}^{m \times n}$, then $\text{Re}(A) = B$, $\text{Im}(A) = C$, and conjugate of $A$ is $\overline{A} = (\overline{a}_{ij})$
  - Conjugate transpose is defined as $A^H = B^T - iC^T$, or
    \[
    G = A^H \Rightarrow g_{ij} = \overline{a}_{ji}
    \]
    (also called adjoint, and denoted by $A^*$; $(AB)^* = B^* A^*$)
- Vector space of complex $n$-vectors is designated by $\mathbb{C}^n$
  - Inner product of complex $n$-vectors $x$ and $y$ is $s = x^H y$
  - Outer product of complex $n$-vectors $x$ and $y$ is $S = xy^H$
- We will primarily focus on real matrices
Matrix-Vector Product

- Matrix-vector product $b = Ax$ is special case of matrix-matrix product

$$b_i = \sum_{j=1}^{n} a_{ij}x_j$$

- For $A \in \mathbb{R}^{m \times n}$, $Ax$ is a mapping $x \mapsto Ax$ from $\mathbb{R}^n$ to $\mathbb{R}^m$

- This map is *linear*, which means that for any $x, y \in \mathbb{R}^n$ and any $\alpha \in \mathbb{R}$

$$A(x + y) = Ax + Ay$$
$$A(\alpha x) = \alpha Ax$$
Linear Combination

- Let \( a_j \) denote \( j \)th column of matrix \( A \)
  - Alternative notation is colon notation: \( A(:,j) \) or \( a_{:,j} \)
  - Use \( A(i,:) \) or \( a_{i,:} \) to denote \( i \)th row of \( A \)
- \( b \) is a **linear combination** of column vectors of \( A \), i.e.,

\[
b = Ax = \sum_{j=1}^{n} x_j a_j = x_1 \begin{bmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{m1} \end{bmatrix} + x_2 \begin{bmatrix} a_{12} \\ a_{22} \\ \vdots \\ a_{m2} \end{bmatrix} + \cdots + x_n \begin{bmatrix} a_{1m} \\ a_{2n} \\ \vdots \\ a_{mn} \end{bmatrix}
\]

- In summary, two different views of matrix-vector products:
  1. **Scalar operations**: \( b_i = \sum_{j=1}^{n} a_{ij} x_j \): \( A \) acts on \( x \) to produce \( b \)
  2. **Vector operations**: \( b = \sum_{j=1}^{n} x_j a_j \): \( x \) acts on \( A \) to produce \( b \)
Matrix-Matrix Multiplication

- Computes $C = AB$, where $A \in \mathbb{R}^{m \times r}$, $B \in \mathbb{R}^{r \times n}$, and $C \in \mathbb{R}^{m \times n}$.
- Element-wise, each entry of $C$ is
  \[ c_{ij} = \sum_{k=1}^{r} a_{ik} b_{kj} \]
- Column-wise, each column of $C$ is
  \[ c_j = Ab_j = \sum_{k=1}^{r} b_{kj} a_k; \]
  in other words, $j$th column of $C$ is $A$ times $j$th column of $B$.
Outline

1. Course Overview

2. The Language of Matrix Computations

3. Algorithms and Efficiency
Algorithms for Matrix-Vector Multiplication

- Suppose $A \in \mathbb{R}^{m \times n}$ and $x \in \mathbb{R}^n$
- MATLAB-style code for $b = Ax$:

  ```
  \begin{align*}
  \text{Row oriented} & \\
  \text{for } i = 1 : m & \\
  & \quad b(i) = 0 \\
  \text{for } j = 1 : n & \\
  & \quad b(i) = b(i) + A(i, j) \times x(j) \\
  \text{end} & \\
  \end{align*}
  
  \begin{align*}
  \text{Column oriented} & \\
  b(:) &= 0 \\
  \text{for } j = 1 : n & \\
  & \quad \text{for } i = 1 : m \\
  & \quad \quad b(i) = b(i) + A(i, j) \times x(j) \\
  \text{end} & \\
  \end{align*}
  ```

- Number of operations is $O(mn)$, but big-Oh notation is insufficient
- Number of operations is $2mn$: coefficient of leading-order term is important for comparison
Flop Count

- It is important to assess *efficiency* of algorithms. But how?
  - We could implement different algorithms and do direct comparison, but implementation details can affect true performance
  - We could estimate cost of all operations, but it is very tedious
  - Relatively simple and effective approach is to estimate amount of floating-point operations, or “flops”, and focus on asymptotic analysis as sizes of matrices approach infinity

- Idealization
  - Count each operation $+,-,\times,\div$, and $\sqrt{}$ as one flop
  - This estimation is crude, as it omits data movement in memory, which is non-negligible on modern computer architectures (e.g., different loop orders can affect cache performance)

- Matrix-vector product requires about $2mn$ flops
- Suppose $m = n$, it takes quadratic time in $n$, or $O(n^2)$
Algorithms for Saxpy: Scalar $a \times x$ plus $y$

- Saxpy computes $ax + y$ and updates $y$

  $$y = ax + y \Rightarrow y_i = ax_i + y_i$$

- Suppose $x, y \in \mathbb{R}^n$ and $a \in \mathbb{R}$

  ```matlab
  for i = 1 : n
    y(i) = y(i) + a * x(i)
  end
  ```

  ```pseudo-code
  for i = 1 : n
    y_i ← y_i + a * x_i
  end
  ```

- Number of flops is $2n$

- Pseudo-code cannot run on any computer, but are human readable and straightforward to convert into real codes in any programming language (e.g., C, FORTRAN, MATLAB, etc.)

- We use pseudo-code on slides for conciseness
Gaxpy: Generalized saxpy

- Computes $y = y + Ax$, where $A \in \mathbb{R}^{m \times n}$ and $x \in \mathbb{R}^n$

<table>
<thead>
<tr>
<th>Row oriented</th>
<th>Column oriented</th>
</tr>
</thead>
</table>
| for $i = 1 : m$  
  for $j = 1 : n$  
  $y_i = y_i + a_{ij}x_j$ | for $j = 1 : n$  
  for $i = 1 : m$  
  $y_i = y_i + a_{ij}x_j$ |

- Inner loop of column-oriented algorithm can be converted to
  $y = y + x_j a_{i,j}$

- Number of flops is $2mn$
Matrix Multiplication Update

- Computes $C = C + AB$, where $A \in \mathbb{R}^{m \times r}$, $B \in \mathbb{R}^{r \times n}$, and $C \in \mathbb{R}^{m \times n}$

  \[ c_{ij} = c_{ij} + \sum_{k=1}^{r} a_{ik} b_{kj} \]

  for $i = 1 : m$
  for $j = 1 : n$
  for $k = 1 : r$

  \[ c_{ij} = c_{ij} + a_{ik} b_{kj} \]

- Number of flops is $2mnr$

- In BLAS (Basic Linear Algebra Subroutines), functions are grouped into level-1, 2, and 3, depending on whether complexity is linear, quadratic, or cubic
Six Variants of Algorithms

- There are six variants depending on permutation of $i, j, \text{ and } k$:
  
  $ijk, jik, ikj, jki, kij, kji$

- Inner product: $c_{ij} = c_{ij} + a_{i,:}b_j$

```plaintext
for $i = 1 : m$
  for $j = 1 : n$
    $c_{ij} = c_{ij} + a_{i,:}b_j$
```

- Saxpy: computes as $c_j = c_j + Ab_j$

```plaintext
for $j = 1 : n$
  $c_j = c_j + Ab_j$
```

- Outer product: computes as $C = C + \sum_{k=1}^{r} a_k b_{:,k}$

```plaintext
for $k = 1 : r$
  $C = C + a_k b_{:,k}$
```