

# CS3430 S24: Scientific Computing

## Lecture 01: Course Overview & Linear Systems: Part 01

Vladimir Kulyukin

Department of Computer Science

Utah State University

# About Me

Ph.D., Computer Science, U. of Chicago, 1998

Software Engineer, BitMobile Technologies, LLC 1998–2002

CS Professor, USU, 2002 – present; graduated 5 Ph.D. students and 27 M.S. students, published over 75 papers and articles, edited 1 open access book, and acquired 1 patent

Professional research interests: precision apiculture/pollination (especially, application/design of AI and computer vision methods to solve apiculture/pollination problems) and computability theory

Personal interests: classical guitar and violin, beekeeping, rational philosophy (especially, foundations of mathematics and epistemology)

# What's Scientific Computing?

Scientific Computing (SciComp) is the collection of algorithms, methods, and tools required to solve numerical problems in various areas of science, engineering, mathematics, economics, etc.

Most of these algorithms, tools, and methods originated in mathematics (calculus, linear algebra, number theory, statistics, etc.) long before the advent of digital computers, which is why these algorithms, tools, and methods used to be called (and are still called by some today) *Numerical Analysis*. However, SciComp is a more common term today, especially in CS, after digital computers have become mainstream and many methods of Numerical Analysis have been computerized.

Python is a convenient tool for SciComp due to a large and ever increasing number of SciComp libraries available in Python. We will use Python 3 in this class.

# Diversity of Students' Interests and Backgrounds

Given the diversity of interests and backgrounds of the students in this class (in the previous years of teaching this class I had students who majored in computer science, engineering, physics, economics, statistics, psychology, aircraft maintenance, food processing, and management information systems), you should expect some problems and applications to come from areas outside of your area of study/interest. Do your best to have an open mind.

Since many SciComp methods/tools are rooted in various branches of mathematics, we will see a fair amount of mathematical notation in this course. I will do my best to make it as digestable for you as I can. In the end, it is my hope that you will gain a clear understanding of the role of SciComp in modern science and will become better scientists for it.

# SciComp S24 Course Themes

Many themes of this course will involve numerical approximation. For example, how do we approximate the value of the numbers  $\pi$  and  $e$ ? The number  $\pi$  introduced and investigated by Archimedes (287 BC – 212 BC) as the ratio of the circumference of a circle to the circle's diameter is used not only in geometry and related fields (e.g., landscaping, computer graphics, computer vision) to approximate the areas of circular objects.

The number  $\pi$  features prominently in radian measurements without which modern navigation systems are impossible. It is also used in Heisenberg's [Uncertainty Principle](#) in quantum mechanics, Einstein's gravitational constant  $k$  in his [field equations](#) of general relativity, and in approximation of functions with sinusoids.

The number  $e$  features prominently in the analysis of growth and decay, e.g., bacteria population growth, spread of epidemics, decay of radioactive materials, compound interest in economics, spread of rumors in sociology.

# SciComp S24 Course Themes

Another theme that we will discuss in the course is randomness. We will take a look at a couple of random number generation algorithms and also discuss how to assess the randomness of various distributions?

Why is randomness important? Because estimated lack of randomness can indicate there may be an underlying pattern that we can discover to explain some natural phenomenon.

For example, if a population of bugs is not randomly distributed in identical bug traps placed in identical (same area, same weather, same plant) agricultural areas, there may be an underlying cause/pattern that warrants further investigation.

# SciComp S24 Course Themes

We will, of course, pay tribute to such standard SciComp themes as integration, differentiation, and irrational number approximation (e.g.,  $\sqrt{2}$ ). Fear not though! We will not be *deriving* or *proving* integration/differentiation formulas, as one does in Calculus classes, but *implementing* and/or *approximating* them in computer programs, e.g., approximating the value of the integral of some function over a closed interval.

We will also talk about congruence systems of degree 1 (used in cryptography) and data compression methods (since modern science deals with large volumes of data, compression is becoming more and more important).

I also budgeted a lecture on entropy, which is used in some machine learning approaches as a measurement of chaos. If we get there, great, but if not, well, life happens!

# Linear Systems

In this lecture, we'll start our journey into SciComp with linear systems. What's a linear system? Here's an example.

$$2x + y = 4$$

$$x - 2y = -3$$

We will call any such collection of simultaneous linear equations a *linear system*. The above two equations are lines in 2D space, but we do not have to be bound by 2 dimensions, we can have 3, 4, 5, or more dimensions.

We'll be concerned with solving a linear systems of  $m$  equations in  $n$  unknowns/variables. The above system has 2 equations in 2 unknowns (i.e.,  $x$  and  $y$ ). We will also use variable names such as  $x_1$ ,  $x_2$ ,  $x_3$ , etc. Using the latter method guarantees that we'll never run out of variable names. Thus, the above linear system can be written as

$$2x_1 + x_2 = 4$$

$$x_1 - 2x_2 = -3$$



# Linear Systems

Why should we care about linear systems? One practical reason is that they allow us to solve linear optimization problems in various domains.

In a typical linear optimization problem, we need to optimize the use of a resource (or several resources) to meet some optimization criteria. For example, we may want to optimize the use of peanuts and raisins in an energy bar to meet specific constraints for caloric and carbohydrate intake and, at the same time, minimize the production cost or we may want to optimize the use of plastic and steel in manufacturing toy cars and trucks to maximize a profit function.

Linear Programming (LP) is the area of scientific computing that uses linear systems to solve such problems so long as all constraints and functions can be expressed with linear equations.

# Matrices

In mathematics (most notably in linear algebra) and in scientific computing, linear systems are solved with matrices. As we'll see shortly, a matrix is a convenient notation for working with a linear system.

The great practical importance of using matrices to solve linear systems is that they enable us to solve linear systems by algebraic methods (i.e., by mechanical manipulation of symbols without using any geometry or computer graphics).

Linear systems are fundamental in linear programming, which we'll study later in this class, partial differential equations, computer vision, etc.

**Historical Note:** The term *matrix* was introduced into the mathematics literature in a 1850 paper by James Joseph Sylvester (1814–1897). Sylvester defined a matrix as an “oblong arrangement of terms.”

# A Few Study Recommendations

In the slides that follow, I will introduce you to the standard mathematical notation for matrices and to the basics of matrix manipulation in numpy.

Some of these techniques we worked out on the board in Lecture 01, others will be covered in Lecture 02. Read these slides once and use them as a point of reference to come back to if and when you need to review some mathematical notation. If you've never seen this notation before, I don't expect you to grasp it all right away.

After you install numpy ([www.numpy.org](http://www.numpy.org)), take a detailed look at and then run `np_elem_row_ops.py`. This file shows you how to do 3 elementary row operations: 1) swapping two rows (aka row interchange), 2) multiplying a row by a real number (aka a scaler), and 3) adding two rows.

I also included two Python files, `mat_add.py` and `mat_mul.py`, that show you how to add and multiply matrices with numpy.

You should definitely run all three files to make sure that numpy works on your system.

# Definition and Notation

A **matrix** is a rectangular array of numbers. Matrices are typically denoted by uppercase boldface type letters (e.g., **A**, **B**, **E**). An  $m \times n$  matrix may be written as

$$\mathbf{A} = [a_{i,j}] = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix}$$

The **order/shape** of the matrix refers to the number of rows and columns of the matrix. The symbol  $a_{i,j}$  refers to the entry in row  $i$  and column  $j$ .

# Definition and Notation

We can use 0-based counting to refer to the rows and columns of a matrix. If we use 0-based notation, then a  $m \times n$  matrix  $\mathbf{A}$  can be defined as follows. Note that the very first entry in the top left corner is now  $a_{0,0}$ , not  $a_{1,1}$ .

$$\mathbf{A} = [a_{i,j}] = \begin{bmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,n-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,n-1} \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ a_{m-1,0} & a_{m-1,1} & \cdots & a_{m-1,n-1} \end{bmatrix}$$

# Definition and Notation

Sometimes the commas are omitted in the subscripts of individual elements.

$$\mathbf{A} = [a_{ij}] = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \cdot & \cdot & \dots & \cdot \\ \cdot & \cdot & \dots & \cdot \\ \cdot & \cdot & \dots & \cdot \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

# Examples

**A** is a  $2 \times 2$  matrix; **B** is a  $3 \times 4$  matrix; **C** is a  $5 \times 4$  matrix.

$$\mathbf{A} = \begin{bmatrix} 3 & 4 \\ 101 & 542 \end{bmatrix}, \text{ where } a_{1,1} = 3, a_{1,2} = 4, a_{2,1} = 101, a_{2,2} = 542.$$

$$\mathbf{B} = \begin{bmatrix} 4 & 5 & 8 & 10 \\ 11 & 52 & 67 & 12 \\ 90 & 7 & 41 & 78 \end{bmatrix}, \text{ where } b_{1,1} = 4, b_{1,2} = 5, \dots, b_{3,4} = 78.$$

$$\mathbf{C} = \begin{bmatrix} 2 & 3 & 4 & 10 \\ 120 & 54 & 78 & 34 \\ 123 & 65 & 81 & 594 \\ 9 & 13 & 74 & 15 \\ 11 & 33 & 54 & 103 \end{bmatrix}, \text{ where } c_{1,1} = 2, c_{1,2} = 3, \dots, c_{5,4} = 103.$$

# Matrix as Shorthand Notation for Linear System

Let's consider this linear system:

$$3x + 2y + z = 39$$

$$2x + 3y + z = 34$$

$$x + 2y + 3z = 26$$

This linear system can be represented in two matrices.

$$\begin{bmatrix} 3 & 2 & 1 \\ 2 & 3 & 1 \\ 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 39 \\ 34 \\ 26 \end{bmatrix}$$

The left matrix contains the variable coefficients; the right matrix (that's a *column matrix* or *column vector*, by the way) contains the right hand side values of each equation. These two matrices preserve all essential information about the linear system, except the variable names (but those are not important). Let's see how we put these 2 matrices into one matrix with augmented matrices.



# Augmented Matrix

An augmented matrix is a matrix in which rows or columns of another matrix of the appropriate order are appended to the original matrix (typically, to the right of the original matrix). If  $\mathbf{A}$  is augmented on the right with  $\mathbf{B}$ , the resultant matrix is denoted as  $(\mathbf{A}|\mathbf{B})$  or  $[\mathbf{A}|\mathbf{B}]$ . Let

$$\mathbf{A} = \begin{bmatrix} 1 & 4 \\ 5 & 6 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 3 \\ 1 \end{bmatrix} \quad \mathbf{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

$$\text{Then } (\mathbf{A}|\mathbf{B}) = \left[ \begin{array}{cc|c} 1 & 4 & 3 \\ 5 & 6 & 1 \end{array} \right] \text{ and } (\mathbf{A}|\mathbf{I}) = \left[ \begin{array}{cc|cc} 1 & 4 & 1 & 0 \\ 5 & 6 & 0 & 1 \end{array} \right].$$

# Matrix as Shorthand Notation for Linear System

Let's consider the same linear system we just looked at:

$$3x + 2y + z = 39$$

$$2x + 3y + z = 34$$

$$x + 2y + 3z = 26$$

We saw that this linear system can be represented by two matrices.

$$\begin{bmatrix} 3 & 2 & 1 \\ 2 & 3 & 1 \\ 1 & 2 & 3 \end{bmatrix} \quad \begin{bmatrix} 39 \\ 34 \\ 26 \end{bmatrix}$$

These two can be combined into one augmented matrix.

$$\left[ \begin{array}{ccc|c} 3 & 2 & 1 & 39 \\ 2 & 3 & 1 & 34 \\ 1 & 2 & 3 & 26 \end{array} \right] \quad \text{or} \quad \left[ \begin{array}{cccc} 3 & 2 & 1 & 39 \\ 2 & 3 & 1 & 34 \\ 1 & 2 & 3 & 26 \end{array} \right]$$

# Matrices in Numpy

Let's define three matrices and construct them with numpy.

**A** is a  $2 \times 2$  matrix; **B** is a  $3 \times 4$  matrix; **C** is a  $5 \times 4$  matrix.

$$\mathbf{A} = \begin{bmatrix} 3 & 4 \\ 101 & 542 \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} 4 & 5 & 8 & 10 \\ 11 & 52 & 67 & 12 \\ 90 & 7 & 41 & 78 \end{bmatrix}$$

$$\mathbf{C} = \begin{bmatrix} 2 & 3 & 4 & 10 \\ 120 & 54 & 78 & 34 \\ 123 & 65 & 81 & 594 \\ 9 & 13 & 74 & 15 \\ 11 & 33 & 54 & 103 \end{bmatrix}$$

# Matrices in Numpy: Construction and Access

Consider this Py code:

```
import numpy as np
A = np.array([[7, -1, -2],
              [3, 3, 0]])
```

Here is a test:

```
>>> A
array([[ 7, -1, -2],
       [ 3,  3,  0]])
>>> A[1,1] == A[1][1] == 3
True
>>> A[0,1] == A[0][1] == -1
True
>>> A[1,2] == A[1][2] == 0
True
```

# Matrix Addition in Numpy

Consider this Py code:

```
A = np.array([[7, -1, -2], [3, 3, 0]])  
B = np.array([[2, -3, 4], [1, 5, 9]])  
print('A+B=')  
print(np.add(A, B))
```

Here is a test:

```
A+B=  
[[ 9 -4  2]  
 [ 4  8  9]]
```

# Matrix Addition in Numpy

Consider this Py code:

```
A = np.array([[7, -1, -2], [3, 3, 0]])  
B = np.array([[2, -3, 4], [1, 5, 9]])  
print('B+A=')  
print(np.add(B, A))
```

Here is a test:

```
B+A=  
[[ 9 -4  2]  
 [ 4  8  9]]
```

# Matrix Multiplication

Two matrices **A** and **B** may be multiplied if they are **conformable** (i.e., if the *number of columns* of **A** is the same as the *number of rows* in **B**). If **A** is an  $m \times n$  matrix and **B** is an  $n \times q$  matrix, then **AB** = **C** is an  $m \times q$  matrix. Each element of **C** is given by

$$c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}, \text{ where}$$

$n$  is the number of columns in **A** (or rows in **B**),  
 $i = 1, \dots, m$  ( $m$  is the number of rows in **A**),  
 $j = 1, \dots, q$  ( $q$  is the number of columns in **B**).

## Example

Let  $\mathbf{A}$  be a  $3 \times 4$  matrix and  $\mathbf{B}$  be a  $4 \times 5$  matrix. Let  $\mathbf{A} \times \mathbf{B} = \mathbf{C}$ , where  $\mathbf{C}$  is a  $3 \times 5$  matrix. Let's compute a few entries in  $\mathbf{C}$ .

1.  $c_{11} = \sum_{k=1}^4 a_{1k}b_{k1} = a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} + a_{14}b_{41};$

2.  $c_{12} = \sum_{k=1}^4 a_{1k}b_{k2} = a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} + a_{14}b_{42};$

3.  $c_{13} = \sum_{k=1}^4 a_{1k}b_{k3} = a_{11}b_{13} + a_{12}b_{23} + a_{13}b_{33} + a_{14}b_{43};$

4.  $c_{14} = \sum_{k=1}^4 a_{1k}b_{k4} = a_{11}b_{14} + a_{12}b_{24} + a_{13}b_{34} + a_{14}b_{44};$

5.  $c_{15} = \sum_{k=1}^4 a_{1k}b_{k5} = a_{11}b_{15} + a_{12}b_{25} + a_{13}b_{35} + a_{14}b_{45};$

6.  $c_{34} = \sum_{k=1}^4 a_{3k}b_{k4} = a_{31}b_{14} + a_{32}b_{24} + a_{33}b_{34} + a_{34}b_{44};$

7.  $c_{23} = \sum_{k=1}^4 a_{2k}b_{k3} = a_{21}b_{13} + a_{22}b_{23} + a_{23}b_{33} + a_{24}b_{43};$



## Example

If **A** is a  $3 \times 4$  matrix and **B** is a  $4 \times$  matrix then their product is **C** a  $3 \times 5$  matrix.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} & b_{15} \\ b_{21} & b_{22} & b_{23} & b_{24} & b_{25} \\ b_{31} & b_{32} & b_{33} & b_{34} & b_{35} \\ b_{41} & b_{42} & b_{43} & b_{44} & b_{45} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} & c_{15} \\ c_{21} & c_{22} & c_{23} & c_{24} & c_{25} \\ c_{31} & c_{32} & c_{33} & c_{34} & c_{35} \end{bmatrix}$$

Each entry  $c_{ij}$  in **C** is  $c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}$ . See the previous slide on how to compute specific  $c_{ij}$  values.

# Matrix Multiplication Examples

Let

$$\mathbf{A} = \begin{bmatrix} 7 & 1 \\ 4 & -3 \\ 2 & 0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 2 & 1 & 7 \\ 0 & -1 & 4 \end{bmatrix} \quad \mathbf{C} = \begin{bmatrix} 1 & -1 & 3 \\ 2 & 2 & 3 \\ -1 & 4 & 7 \end{bmatrix}.$$

$$\text{Then } \mathbf{AB} = \begin{bmatrix} 7 & 1 \\ 4 & -3 \\ 2 & 0 \end{bmatrix} \begin{bmatrix} 2 & 1 & 7 \\ 0 & -1 & 4 \end{bmatrix} = \begin{bmatrix} 14 & 6 & 53 \\ 8 & 7 & 16 \\ 4 & 2 & 14 \end{bmatrix}.$$

$\mathbf{AB} = \mathbf{C} = [c_{ij}] = \sum_{k=1}^2 a_{ik} b_{kj}$ ,  $i \in [1, 3], j \in [1, 2]$ . Let us compute the first two columns of  $\mathbf{C}$ .

$$c_{11} = \sum_{k=1}^2 a_{1k} b_{k1} = a_{11} b_{11} + a_{12} b_{21} = 7 \cdot 2 + 1 \cdot 0 = 14.$$

$$c_{21} = \sum_{k=1}^2 a_{2k} b_{k1} = a_{21} b_{11} + a_{22} b_{21} = 4 \cdot 2 + (-3) \cdot 0 = 8.$$

$$c_{31} = \sum_{k=1}^2 a_{3k} b_{k1} = a_{31} b_{11} + a_{32} b_{21} = 2 \cdot 2 + 0 \cdot 0 = 4.$$

$$c_{12} = \sum_{k=1}^2 a_{1k} b_{k2} = a_{11} b_{12} + a_{12} b_{22} = 7 \cdot 1 + 1 \cdot (-1) = 6.$$

$$c_{22} = \sum_{k=1}^2 a_{2k} b_{k2} = a_{21} b_{12} + a_{22} b_{22} = 4 \cdot 1 + (-3) \cdot (-1) = 7.$$

$$c_{32} = \sum_{k=1}^2 a_{3k} b_{k2} = a_{31} b_{12} + a_{32} b_{22} = 2 \cdot 1 + 0 \cdot (-1) = 2.$$

# Matrix Multiplication in Numpy

Consider this Py code:

```
import numpy as np

A = np.array([[7, 1],
              [4, -3],
              [2, 0]])
B = np.array([[2, 1, 7],
              [0, -1, 4]])
print('A x B = ')
print(np.dot(A, B))
```

Here is the output:

```
A x B =
[[14  6 53]
 [ 8  7 16]
 [ 4  2 14]]
```

# Column Multiplication

What happens if we multiply an  $m \times n$  matrix  $\mathbf{A}$  by a  $n \times 1$  column matrix  $\mathbf{X}$ ? We end up with an  $m \times 1$  column matrix.

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \cdot & \cdot & \dots & \cdot \\ \cdot & \cdot & \dots & \cdot \\ \cdot & \cdot & \dots & \cdot \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \times \begin{bmatrix} x_{11} \\ x_{21} \\ \cdot \\ \cdot \\ \cdot \\ x_{n1} \end{bmatrix} = \begin{bmatrix} \sum_{k=1}^n a_{1k} x_{k1} \\ \sum_{k=1}^n a_{2k} x_{k1} \\ \cdot \\ \cdot \\ \cdot \\ \sum_{k=1}^n a_{mk} x_{k1} \end{bmatrix}$$

## Example

Let's multiply an  $3 \times 3$  matrix **A** by a  $3 \times 1$  column matrix **X**.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \times \begin{bmatrix} x_{11} \\ x_{21} \\ x_{31} \end{bmatrix} = \begin{bmatrix} \sum_{k=1}^n a_{1k}x_{k1} = a_{11}x_{11} + a_{12}x_{21} + a_{13}x_{31} \\ \sum_{k=1}^n a_{2k}x_{k1} = a_{21}x_{11} + a_{22}x_{21} + a_{23}x_{31} \\ \sum_{k=1}^n a_{3k}x_{k1} = a_{31}x_{11} + a_{32}x_{21} + a_{33}x_{31} \end{bmatrix}$$

Let's simplify and let  $b_1 = \sum_{k=1}^n a_{1k}x_{k1}$ ,  $b_2 = \sum_{k=1}^n a_{2k}x_{k1}$ ,  $b_3 = \sum_{k=1}^n a_{3k}x_{k1}$ . Then the above column matrix multiplication equation can be expressed as follows.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \times \begin{bmatrix} x_{11} \\ x_{21} \\ x_{31} \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

## Example's Insight

Here's the column matrix multiplication example from the previous example.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \times \begin{bmatrix} x_{11} \\ x_{21} \\ x_{31} \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

The insight, which is important for the next slide, is that the above equation corresponds to the following linear system of 3 equations in 3 unknowns.

$$a_{11}x_{11} + a_{12}x_{21} + a_{13}x_{31} = b_1$$

$$a_{21}x_{11} + a_{22}x_{21} + a_{23}x_{31} = b_2$$

$$a_{31}x_{11} + a_{32}x_{21} + a_{33}x_{31} = b_3$$

# Generic Linear System

A generic linear systems with  $m$  equations in  $n$  unknowns is written as follows:

$$a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1$$

$$a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2$$

$$\cdot$$
$$\cdot$$
$$\cdot$$

$$a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n = b_m$$

Since the system is determined by its  $m \times n$  coefficient matrix  $\mathbf{A} = [a_{ij}]$  and its column vector  $\mathbf{b}$  of the corresponding right-hand side values, it can be written as  $\mathbf{Ax} = \mathbf{b}$  where  $\mathbf{x}$  is a column vector  $(x_1, x_2, \dots, x_n)$ .

# Generic Linear System as Augmented Matrix

A generic linear systems with  $m$  equations in  $n$  can be expressed with the following augmented matrix:

$$\left[ \begin{array}{cccc|c} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ a_{21} & a_{22} & \dots & a_{2n} & b_2 \\ \cdot & \cdot & \dots & \cdot & \cdot \\ \cdot & \cdot & \dots & \cdot & \cdot \\ a_{m1} & a_{m2} & \dots & a_{mn} & b_m \end{array} \right]$$

The above matrix is typically shorthanded as  $[\mathbf{A}|\mathbf{b}]$ .



# Elementary Row Operations on Augmented Matrix

A commonly used method of solving linear systems, which we'll study in this course, is known as *Gauss-Jordan Elimination* or *Gauss-Jordan Reduction*. It's based on the so-called 3 *elementary row operations* **R1**, **R2**, and **R3** defined below.

- ▶ **R1** (Row interchange): Interchange any two rows in a matrix.
- ▶ **R2** (Row scaling): Multiply any row in the matrix by a nonzero scalar.
- ▶ **R3** (Row addition): Replace any row in the matrix with the sum of that row and another row in the matrix.

# Solving Linear Systems

- ▶ Solving systems of linear equations is a fundamental problem of linear algebra.
- ▶ The solution set of any system of linear equations is the intersection of the solution sets of the individual equations.
- ▶ Any solution of a system must be a solution of each equation in the system.
- ▶ Any solution of every equation in the system is a solution of the system.

# Row Equivalence

- ▶ If a matrix **B** can be obtained from a matrix **A** by a sequence of elementary row operations, then **B** is **row equivalent** to **A**.
- ▶ Since each elementary row operation can be undone (reversed), if **B** is row equivalent to **A**, denoted as  $\mathbf{B} \sim \mathbf{A}$ , then **A** is row equivalent to **B**, i.e.,  $\mathbf{A} \sim \mathbf{B}$ .
- ▶ The elementary row operations do not change the solution set of an augmented matrix.

# A Fundamental Theorem of Linear Algebra

If  $[\mathbf{A}|\mathbf{b}] \sim [\mathbf{H}|\mathbf{c}]$ , then the corresponding linear systems  $\mathbf{Ax} = \mathbf{b}$  and  $\mathbf{Hx} = \mathbf{c}$  have the same solution set.

# Row Echelon Form and Pivot

A matrix is in **row echelon form** if:

- ▶ All rows containing only zeros appear below the rows containing nonzero entries.
- ▶ The first nonzero entry in any row appears in a column to the right of the column of the first nonzero entry in any preceding row.

The first nonzero entry in a row of a row echelon form matrix is called the **pivot**.

## Example

Let's consider the following matrices.

$$A = \begin{bmatrix} 1 & 3 & 2 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad B = \begin{bmatrix} 2 & 4 & 0 \\ 1 & 3 & 2 \\ 0 & 0 & 0 \end{bmatrix} \quad C = \begin{bmatrix} 0 & -1 & 2 \\ 0 & 0 & 3 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$
$$D = \begin{bmatrix} 1 & 3 & 2 & 5 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

**A**, **B** are not in row echelon form; **C**, **D** are in row echelon form.

# References

1. [www.numpy.org](http://www.numpy.org).
2. Wiki article on Heisenberg's Uncertainty Principle  
[en.wikipedia.org/wiki/Uncertainty\\_principle](http://en.wikipedia.org/wiki/Uncertainty_principle)
3. Wiki article on Einstein's Field Equations  
[en.wikipedia.org/wiki/Einstein\\_field\\_equations](http://en.wikipedia.org/wiki/Einstein_field_equations)
4. J. Fraleigh, R. Beauregard. *Linear Algebra*, Ch. 01.