• CSCI 2033 • Spring 2016 •

ELEMENTARY COMPUTATIONAL LINEAR ALGEBRA

Class time : MW 4:00-5:15pm

Room : Vincent Hall 16

Instructor : Yousef Saad

URL : www-users.cselabs.umn.edu/classes/Spring-2016/csci2033_afternoon

February 7, 2016

About this class

- Me: Yousef Saad
- TAs:
  1. Dimitrios Kottas
  2. James Cannalte
  3. Alex Dahl

What you will learn and why

- Course is about
"Basics of Numerical Linear Algebra", a.k.a. "matrix computations"
- Topic becoming increasingly important in Computer Science.
- Many courses require some linear algebra
- Course introduced in 2011 to fill a gap.
- In the era of 'big-data' you need 1) statistics and 2) linear algebra

➤ CSCI courses where csci2033 plays an essential role:

- CSCI 5302 – Analysis Num Algs *
- CSCI 5304 – Matrix Theory *
- CSCI 5607 – Computer Graphics I *
- CSCI 5512 – Artif Intelligence II
- CSCI 5521 – Intro to Machine Learning *
- CSCI 5551 – Robotics *
- CSCI 5525 – Machine Learning
- CSCI 5451 – Intro Parall Comput

* = csci2033 prerequisite for this course

➤ Courses for which csci2033 can be helpful

- CSCI 5221 – Foundations of Adv Networking
- CSCI 5552 – Sensing/Estimation in Robotics
- CSCI 5561 – Computer Vision
- CSCI 5608 – Computer Graphics II
- CSCI 5619 – VR and 3D Interaction
- CSCI 5231 – Wireless and Sensor Networks
- CSCI 5481 – Computational Techs. Genomics

Objectives of this course

Set 1 Fundamentals of linear algebra

- Vector spaces, matrices, – [theoretical]
- Understanding bases, ranks, linear independence -
- Improve mathematical reasoning skills [proofs]

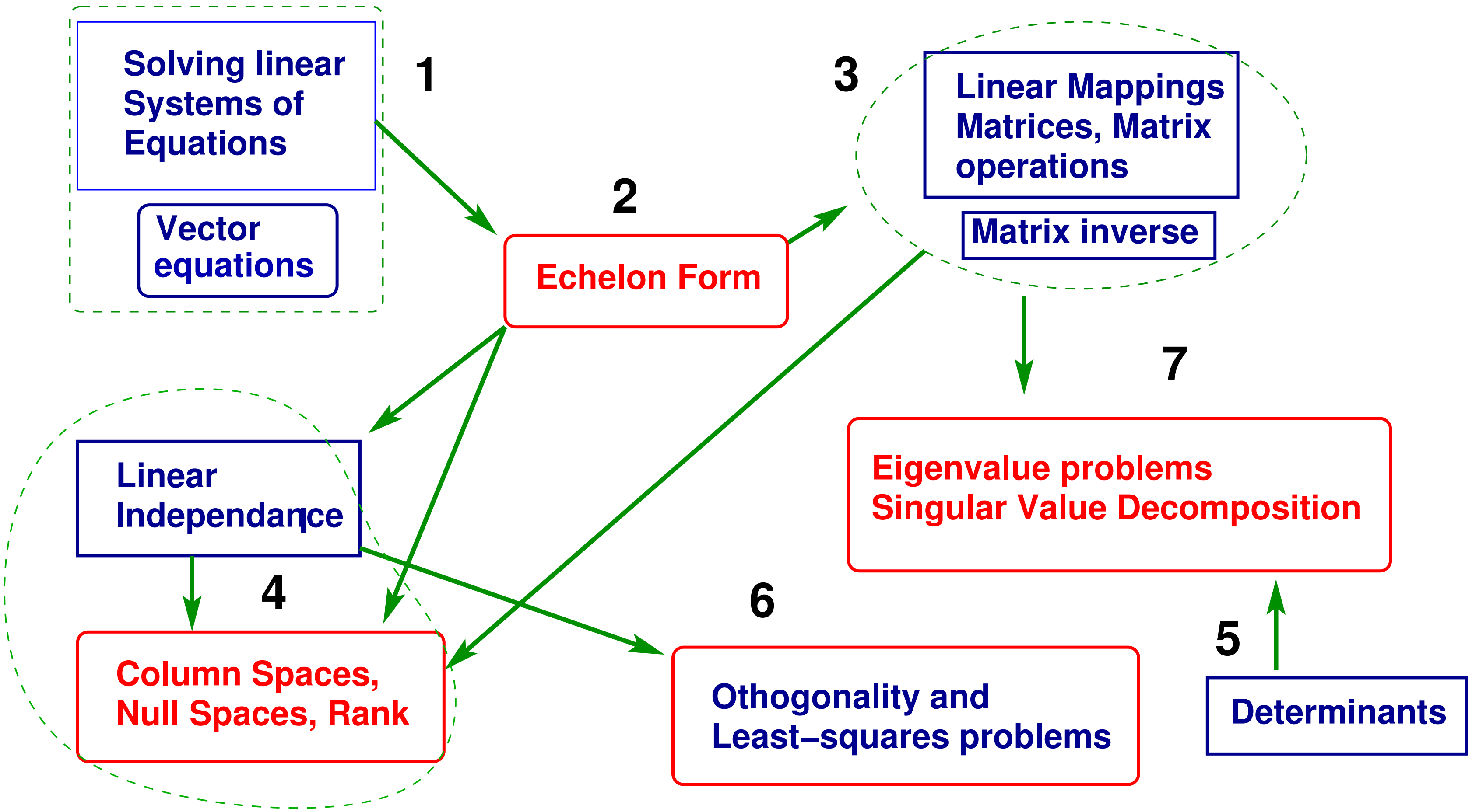
set 2 Computational linear algebra

- Understanding common computational problems
- Solving linear systems
- Get a working knowledge of matlab
- Understanding computational complexity

Set 3 Linear algebra in applications

- See how numerical linear algebra arises in a few computer science -related applications.

The road ahead: Plan in a nutshell



Math classes

- Students who already have had Math 2243 or 2373 (Linear Algebra and Differential Equations) or a similar version of a linear algebra course :

There is a good overlap with this course.

You can substitute 2033 for something else

- See UG adviser if you are in this situation.

Logistics:

- We will use Moodle only to post grades
- Main class web-site is :

`www-users.cselabs.umn.edu/classes/Spring-2016/csci2033_afternoon/`

- There you will find :
  - Lecture notes
  - Homeworks [and solutions]
  - Additional exercises [do before indicated class]
  - .. and more

Three Recitation Sections:

sec 011 – which we will call *Sec. 1* - 2:30 – 3:20 pm

sec 012 – which we will call *Sec. 2* - 3:35 – 4:25pm

sec 013 – which we will call *Sec. 3* - 4:40 – 5:30pm

- All in Keller Hall 2-260
- All lead by Dimitrios Kottas

About lecture notes:

- Lecture notes will be posted on the class web-site – usually before the lecture. [if I am late do not hesitate to send me e-mail]

- Review them and try to get some understanding if possible before class.

- Read the relevant section (s) in the text

- Lecture note sets are grouped by topics (sections in the textbook) rather than by lecture.

- In the notes the symbol ✍ indicates suggested easy exercises or questions – often [not always] done in class.

In-class Practice Exercises

- Posted in advance – see HWs web-page
- You should do them before class (!Important). No need to turn in anything. But...
- ... prepare for an occasional follow-up Quiz
- I will usually start the class with these practice exercises
- On occasion a quiz will follow
- There may also be quizzes at other times

Matlab

- You will need to use matlab for testing algorithms.
- Limited lecture notes on matlab +
- Other documents will be posted in the matlab web-site.
- Most important:
- .. I post the matlab diaries used for the demos (if any).
- First few recitations will cover tutorials on matlab

> • If you do not know matlab at all and have difficulties with it see me or one of the TAs at office hours. This ought to help get you started.

One final point on lecture notes

➤ These notes are ‘evolving’. You can help make them better – report errors and provide feedback.

➤ There will be much more going on in the classroom - so the notes are not enough for studying! Sometimes they are used as a summary.

➤ There are a few topics that are not covered well in the text (e.g., complexity). Rely on lectures and the notes (when available) for these.

Introduction. Math Background

- We will often need proofs in this class.
- A proof is a logical argument to show that a given statement in true
- One of the stated goals of csci2033 is to improve mathematical reasoning skills
- You should be able to prove simple statements
- Here are the most common types of proofs

Proof by contradiction:

Idea: prove that the contrary of the statement implies an impossible ('absurd') conclusion

Example:

✍ Show that $\sqrt{2}$ is not a rational number [famous proof dating back to Pythagoras]

Proof: Assume the contrary is true. Then $\sqrt{2} = p/q$. If p and q can be divided by the same integer divide them both by this integer. Now p and q cannot be both even. The equality $\sqrt{2} = p/q$ implies $p^2 = 2q^2$. This means p^2 is even. However p is also even because the square of an odd number is odd. We now write $p = 2k$. Then $4k^2 = 2q^2$. Hence $q^2 = 2k^2$ and so q is also even. Contradiction. ■

Proof by induction

Problem: to prove that a certain property P_n is true for all n.

Method:

(a) Base: Show that P_{init} is true

(b) Induction Hypothesis: Assume that P_n is true for some n ($n \geq init$). With this assumption prove that P_{n+1} is true..

➤ Important point: A big part of the proof is to clearly state P_n

Example: Show that $\boxed{1 + 2 + 3 + \cdots + n = n(n+1)/2}$

✍ [Challenge] Show:

$$\boxed{1^2 + 2^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6}}$$

By counter-example [to [prove a statement is not true]

Example: All students in MN are above average.

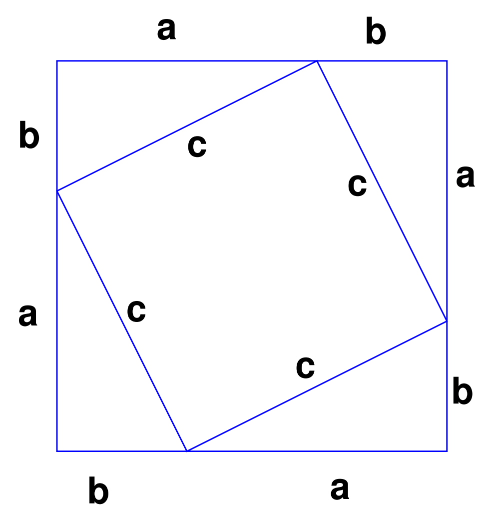
Proof by construction (constructive proof)

The statement is that some object exists. We need to construct this object.

By a purely logical argument

Example:

➤ Pythagoras' theorem from a purely geometric argument



Behold!

✍ Show that for two sets A, B we have $\overline{A \cup B} = \overline{A} \cap \overline{B}$

A few terms/symbols used

$x \in X$ x belongs to set X

$\forall x$ for all x

$\sum_{i=1}^{n}$ Summation from $i = 1$ to $i = n$

$A \rightarrow B$ Assertion A implies assertion B

A `iff` B A is true If and only if B is true [i.e., $A \rightarrow B$ and $B \rightarrow A$]

- Greek letters α , β, γ, ... represent scalars
- Lower case latin letters $u, v, \ldots$ often represent vectors
- Upper case letters $A, B, ..$ often represent matrices
- More will be introduced on the way

Algorithms - complexity

- Not emphasized in text

✍ Find (google) the origin of the word 'Algorithm'

An algorithm is a sequence of instructions given to a machine (typically a computer) to solve a given problem

An example: Finding the square root of a number.

Method: calculate

$$x_{new} = 0.5\left(x_{old} + \frac{a}{x_{old}}\right)$$

... until x_{new} no longer changes much. Start with $x = a$

➤ There are different ways of implementing this

➤ Some ways may be more 'economical' than others

➤ Some ways will lead to more numerical errors [not in this particular case]

ALGORITHM : 1. *Algorithm for Square Root Finding*

0. Input: $\boldsymbol{a}$*, tolerance* $\boldsymbol{tol}$
1. Start: $\boldsymbol{xn := a}$
2. Do :
3. $\quad \boldsymbol{x := xn}$ *[To save* $\boldsymbol{xn}$*]*
4. $\quad \boldsymbol{xn := 0.5 * (x + a/x)}$
5. until ($\boldsymbol{|xn - x| < xn * tol}$*) [stopping criterion]*

Matlab function for square root

➤ Matlab functions will be seen later [recitations or class] – this is given just as an illustration.

```
function x = mysqrt(a, tol, maxits)
% x = mysqrt(a, tol, maxits)
% computes the square root of a (a must be > 0 )
   if (a <0)
       error(' *** a < 0')
   end
%%-------------------- main loop
   for i=1:maxits
       x = xn ;
       xn = 0.5*(x+a/x)
       if (abs(xn-x) < tol*xn)
         break
       end
   end
```

✍ You can find slightly better implementations

The issue of cost ('complexity')

➤ For small problems cost may not be important - except when the operation is repeated many times.

➤ For systems of equations in the thousands, then the algorithm could make a huge difference.

What to count?

- Memory copy / move.
- Comparisons of numbers (integers, floating-points)
- Floating point operations: add, multiply, divide (more expensive)
- Intrinsic functions: $\sin, \cos, exp, \sqrt{\ }$, etc.. a few times more expensive than add/ multiply.

Example: Assume we have 4 algorithms whose costs (number of operations) are $\frac{n^3}{6}$, $\frac{n^2}{2}$, $n \log_2 n$, and n respectively, where n is the 'size' of the problem. Compare the times for the 4 algorithms to execute when $n = 1000$

Answer: [assume one operation costs $1\mu sec$]

$\frac{n^3}{6} \quad \rightarrow \quad \frac{10^9}{6}\mu sec = \frac{1000}{6}\ sec \approx 2.78mn$

$\frac{n^2}{2} \quad \rightarrow \quad \frac{10^6}{2}\ \mu sec \approx \frac{1}{2} sec.$

$n \log n \quad \rightarrow \quad 10^3 \log n\ \mu sec \approx 10^3 \times 10\ \mu sec = 10ms$

$n \quad \rightarrow \quad 1\ ms.$

➤ In matrix computations (this course) we only count floating point operations: $(*, +, /)$

- Cost $=$ number of operations to complete a given algorithm $=$ function of n the problem size
- Will find something like [example]

$$C(n) = 2n^3 + n^2 - 3n$$

- We are interested in cases with large values of n
- Major point: only the leading term $2n^3$ matters - because the rest is small (relatively to $2n^3$) when n is large.
- We will say that the cost is of order $2n^3$ or even order n^3 [meaning that it increases like the cube of n as n increases]

✍ Compare $C(100)$, $C(200)$ and $8C(100)$. Explain

✍ Suppose it takes 1 sec. run the algorithm for a certain value of n (large), how long would it take to run the same algorithm on a problem of size $2n$?

LINEAR EQATIONS [1.1] +

Linear systems

- A linear equation in the variables $x_1, \cdots, x_n$ is an equation that can be written in the form

$$a_1x_1 + a_2x_2 + \cdots + a_nx_n = b,$$

- b and the coefficients $a_1, \cdots, a_n$ are known real or complex numbers.

Example: $x_1 + 2x_2 = -1$

- In the above equation x_1 and x_2 are the unknowns or variables. The equation is satisfied when $x_1 = 1, x_2 = -1$.

- It is also satisfied for $x_1 = -3, x_2 = ?$

➤ A system of linear equations (or a linear system) is a collection of one or more linear equations involving the same variables – say, $x_1, ..., x_n$.

➤ A solution of the system is a list $(s_1, s_2, ..., s_n)$ of values for $x_1, x_2, ..., x_n$, respectively, which make the equations satisfied.

Example: Here is a system involving 2 unknowns:

$$\begin{cases} 2x_1 & +x_2 & = 4 \\ -x_1 & +2x_2 & = 3 \end{cases}$$

➤ The values $x_1 = 1, x_2 = 2$ satisfy the system of equations. $s_1 = 1, s_2 = 2$ is a solution.

➤ The equation $2x_1 + x_2 = 4$ represents a line in the plane. $-x_2 + 2x_2 = 3$ represents another line. The solution represents the point where the two lines intersect.

Example:

Three winners of a competition labeled G, S, B (for gold, silver, bronze) are to share as a prize 30 coins. The conditions are that 1) G's share of the coins should equal the shares of S and B combined and 2) The difference between the shares of G and S equals the difference between the shares of S and B.

- How many coins should each of G, S, B receive?
- Should formulate as a system of equations:
  - 3 conditions $\longrightarrow$ result will be 3 equations
  - 3 unknowns (# coins for each of winner)

- Let $\left| \begin{array}{l} x_1 = \text{number of coins to be won by } G, \\ x_2 = \text{number of coins to be won by } S, \text{ and} \\ x_3 = \text{number of coins to be won by } B \end{array} \right.$

- The conditions give us 3 equations which are:

1) Total number of coins $=$ 30
2) G's share $=$ sum of S and B
3) differences G -S same as S-B

$$\boxed{\begin{array}{l} x_1+x_2+x_3 = 30 \\ x_1 = x_2 + x_3 \\ x_1 - x_2 = x_2 - x_3 \end{array}}$$

System of equations:

$$\begin{cases} x_1 & +x_2 & +x_3 & = 30 \\ x_1 & -x_2 & -x_3 & = 0 \\ x_1 & -2x_2 & +x_3 & = 0 \end{cases}$$

- We will see later how to solve this system

- The set $s_1 = 15, s_2 = 10, s_3 = 5$ is a solution

- It is the only solution

➤ The set of all possible solutions is called the solution set of the linear system.

➤ Two linear systems are called equivalent if they have the same solution set.

➤ A system of linear equations can have:

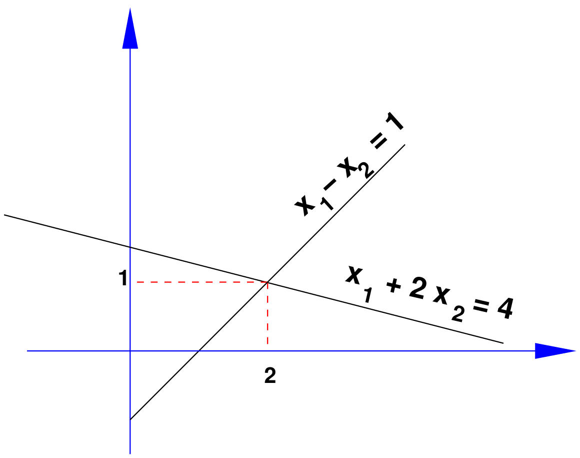
1. no solution, or
2. exactly one solution, or
3. infinitely many solutions.

[The above result will be seen in detail later in this class]

Definition: *A system of linear equations is said to be inconsistent if it has no solution (Case 1 above). It is consistent if it has at least one solution (Case 2 or Case 3 above).*

Example: Consider the following three systems of equations:

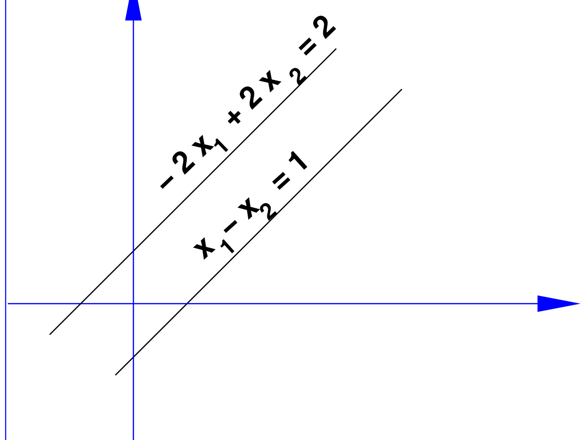
$$\begin{cases} x_1 \quad -x_2 = 1 \\ x_1 \ +2x_2 = 4 \end{cases}$$



Exactly one solution

Consistent

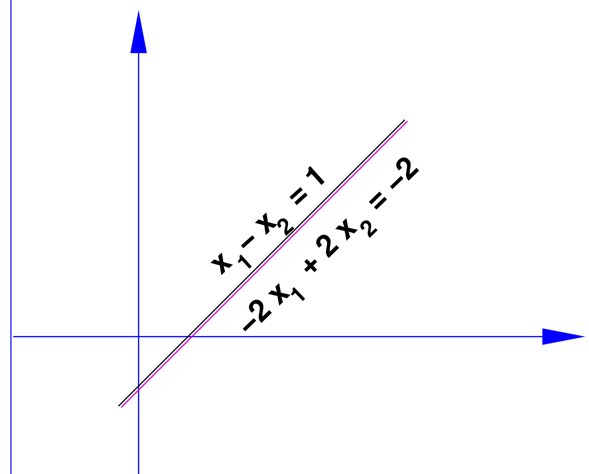
$$\begin{cases} \quad x_1 \quad -x_2 = 1 \\ -2x_1 \ +2x_2 = 2 \end{cases}$$



No solution

Inconsistent

$$\begin{cases} \quad x_1 \quad -x_2 = 1 \\ -2x_1 \ +2x_2 = -2 \end{cases}$$



Inifinitely many solutions

Consistent

Matrix Notation

➤ The essential information of a linear system is recorded compactly in a rectangular array called a matrix.

➤ For the following system of equations:

$$\begin{cases} x_1 + x_2 + x_3 = 30 \\ x_1 - x_2 - x_3 = 0 \\ x_1 - 2x_2 + x_3 = 0 \end{cases}$$

The array to the right is called the coefficient matrix of the system:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & -1 \\ 1 & -2 & 1 \end{bmatrix}$$

And the right-hand side is:

$$\begin{bmatrix} 30 \\ 0 \\ 0 \end{bmatrix}$$

➤ An augmented matrix of a system consists of the coefficient matrix with the R.H.S. added as a last column

➤ Note: R.H.S. or RHS = short for right-hand side column.

- For the above system the augmented matrix is

$$\left[\begin{array}{ccc|c} 1 & 1 & 1 & 30 \\ 1 & -1 & -1 & 0 \\ 1 & -2 & 1 & 0 \end{array}\right] \quad \text{or} \quad \begin{bmatrix} 1 & 1 & 1 & 30 \\ 1 & -1 & -1 & 0 \\ 1 & -2 & 1 & 0 \end{bmatrix}$$

- You can think of the array on the left as the set of 3 "rows" each representing an equation:

$$\begin{array}{ccc|c} x_1 & x_2 & x_3 & b_1 \\ \hline 1 & 1 & 1 & 30 \end{array} \qquad \begin{array}{ccc|c} x_1 & x_2 & x_3 & b_2 \\ \hline 1 & -1 & -1 & 0 \end{array} \qquad \begin{array}{ccc|c} x_1 & x_2 & x_3 & b_3 \\ \hline 1 & -2 & 1 & 0 \end{array}$$

- To solve systems of equations we manipulate these "rows" to get equivalent equations that are easier to solve.

✍ Can we add two equations/rows? Add equations 1 and 2. What do you get?

✍ Now add equations 2 and 3. What do you get? Can you compute x_2?

✍ Finally obtain x_3

➤ This shows an "ad-hoc" [intuitive] way of manipulating equations to solve the system.

➤ Gaussian Elimination [coming shortly] shows a systematic way

➤ Basic Strategy: replace a system with an equivalent system (i.e., one with the same solution set) that is easier to solve.

Terminology on matrices

- An $m \times n$ matrix is a rectangular array of numbers with m rows and n columns. We say that A is of size $m \times n$ (The number of rows always comes first.)

- In matlab: $[m, n] = \mathtt{size}(A)$ returns the size of A

- If $m = n$ the matrix is said to be square otherwise it is rectangular

- The case when $n = 1$ is a special case where the matrix consists of just one column. The matrix then becomes a vector and this will be revisited later. The right-hand side column is one such vector.

- Thus a linear system consists of a coefficient matrix A and a right-hand side vector b.

Equivalent systems

We do not change the solution set of a linear system if we

* Permute two equations
* Multiply a whole equation by a nonzero scalar
* Add an equation to another.

➤ Text: Two systems are row-equivalent if one is obtained from the other by a succession of the above operations

➤ Eliminating an unknown consists of combining rows so that the coefficients for that unknown in the equations become zero.

➤ Gaussian Elimination: performs eliminations to reduce the system to a "triangular form"

$$\left[\begin{array}{cccc|c} * & * & * & * & * \\ \mathbf{0} & * & * & * & * \\ \mathbf{0} & \mathbf{0} & * & * & * \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & * & * \end{array}\right]$$

Triangular linear systems are easy to solve

Example:

$$\begin{cases} 2x_1 + 4x_2 + 4x_3 = 2 \\ \qquad\quad 5x_2 - 2x_3 = 1 \\ \qquad\qquad\quad 2x_3 = 4 \end{cases} \qquad \left[\begin{array}{rrr|r} 2 & 4 & 4 & 2 \\ 0 & 5 & -2 & 1 \\ 0 & 0 & 2 & 4 \end{array}\right]$$

- One equation can be trivially solved: the last one.

$$x_3 = 2$$

- x_3 is known we can now solve the 2nd equation:

$$5x_2 - 2x_3 = 1 \;\rightarrow\; 5x_2 - 2 \times 2 = 1 \;\rightarrow\; x_2 = 1$$

- Finally x_1 can be determined similarly:

$$2x_1 + 4 \times 1 + 4 \times 2 = 2 \rightarrow \cdots \rightarrow x_1 = -5$$

Triangular linear systems - Algorithm

- Upper triangular system of size n

ALGORITHM : 2. *Back-Substitution algorithm*

For $i = n : -1 : 1$ *do:*
  $t := b_i$
  For $j = i + 1 : n$ *do*
    $t := t - a_{ij} x_j$
  End
  $x_i = t / a_{ii}$
End

- We must require that each $a_{ii} \neq 0$

x_1	x_2	x_2	x_4	x_5	b
a_{11}	a_{12}	a_{13}	a_{14}	a_{15}	b_1
	a_{22}	a_{23}	a_{24}	a_{25}	b_2
		a_{33}	a_{34}	a_{35}	b_3
			a_{44}	a_{45}	b_4
				a_{55}	b_5

$i = 5$ $\quad x_5 = b_5/a_{55}$

$i = 4$ $\quad x_4 = [b_4 - a_{45}x_5]/a_{44}$

$i = 3$ $\quad x_3 = [b_3 - a_{34}x_4 - a_{35}x_5]/a_{33}$

$i = 2$ $\quad x_2 = [b_2 - a_{23}x_3 - a_{24}x_4 - a_{25}x_5]/a_{22}$

$i = 1$ $\quad x_1 = [b_2 - a_{12}x_2 - a_{13}x_3 - a_{14}x_4 - a_{15}x_5]/a_{11}$

- For example, when $i = 3$, x_4, x_5 are already known, so

$$a_{33}x_3 + \underbrace{a_{34}x_4 + a_{35}x_5}_{\text{known}} = b_3 \rightarrow x_3 = \frac{b_3 - a_{34}x_4 - a_{35}x_5}{a_{33}}$$

✍ Write a matlab version of the algorithm

✍ Cost: How many operations $(+, *, /)$ are needed altogether to solve a triangular system? [Hint: visualize the operations on the augmented array. What does step i cost?]

✍ If n is large and the $n \times n$ system is solved in 2 seconds, how long would it take you to solve a new system of size $(2n) \times (2n)$?