composition

- Given an m-by-n matrix A and an n-by-r matrix B and a vector $\mathbf{v} \in \mathbb{R}^r$, then $A(B\mathbf{v})$ is a well defined vector in $\mathbb{R}^m$.
- This is simply the result of composition of the applications of maps $\mathcal{B}$ and $\mathcal{A}$ to $\mathbf{v}$.
- This composition, itself, describes a map $\mathcal{C}$, which we can verify is linear, and thus representable by a unique m-by-r matrix C.
- Let us define *matrix-matrix* multiplication to be the operation that sends AB to this C.
- so the matrix product AB, *is* the m-by-r matrix C, with the property that, for all $\mathbf{v}$, $C\mathbf{v} = A(B\mathbf{v})$.
- we don't yet know how to compute this product.
- This definition will automatically give us $A(B\mathbf{v}) = (AB)\mathbf{v}$ (associativity).
- Make sure you notice the pattern of dimensions on A B and C. The "inner" dimensions of A and B must match, and their "outer" dimensions give you the dimension of C.

how to compute C

- let us try to compute $\mathbf{c}_j$, the jth column of C.
- recall that $\mathbf{c}_j = C\mathbf{e}_j$.
- meanwhile by our defnition of matmatmul, $C\mathbf{e}_j = A(B\mathbf{e}_j)$
- but $B\mathbf{e}_j = \mathbf{b}_j$.
- so $C\mathbf{e}_j = A\mathbf{b}_j$
- so $\mathbf{c}_j = A\mathbf{b}_j$
- doing this for all of the j tells us all of C!
- Writing out all of the columns gives us

$$[\mathbf{c}_1..\mathbf{c}_r] = C = AB = A[\mathbf{b}_1..\mathbf{b}_r] = [A\mathbf{b}_1..A\mathbf{b}_r]$$

computing in column form

- Let us work out the details of computing a column of C

$$\mathbf{c}_j = A\mathbf{b}_j = [\mathbf{a}_1\mathbf{a}_2...\mathbf{a}_n]\mathbf{b}_j = \sum_k \mathbf{a}_k b_{kj} \tag{1}$$

- In the above, the kth entry of the vector $\mathbf{b}_j$ is also the matrix entry b_{kj}.
- In English, each column of C is a linear combination of the columns of A using weights from the corresponding column of B.
- useful form for understanding

computing in entry form

- Let us work out the details of computing the ith entry of $\mathbf{c}_j$, which is also the matrix entry c_{ij}.

$$c_{ij} = \sum_k a_{ik} b_{kj} \tag{2}$$

- In the above, the ith entry of the vector $\mathbf{a}_k$ is also the matrix entry A_{ik}.
- In English this says that we take the ith row of A and the jth column of B and run the two finger rule!
- useful form for calculations

one column

- Notice that if B is a matrix of size n-by-1, then C will be m-by-1 and we get

$$[\mathbf{c}_1] = C = AB = A[\mathbf{b}_1] = [A\mathbf{b}_1]$$

- So, (up to data typing) matrix-vector multiplication is a special case of matrix-matrix multiplication.

properties

Theorem. *Let A be m-by-n.*

- $I_m A = A$.
- $AI_n = A$.

Proof. For the first statement: the jth column of the product must be $I_m\mathbf{a}_j = \mathbf{a}_j$. For the second statement, the jth column of the product must be $A\mathbf{e}_j = \mathbf{a}_j$. □

Theorem. *Let A, B and C be any matrices of appropriate dimensions then*

- $A(BC) = (AB)C$
- $A(B+C) = AB + AC$
- $(B+C)A = BA + CA$
- $r(AB) = (rA)B = A(rB)$

caveats

- But not all typical manipulations are allowed.
- we are not guaranteed that (say for square matrices) AB is equal to BA.
  - Matrix multiplication is not commutative.
  - For example, x-scale followed by a rotation is different than rotation followed by an x-scale.
- Also, in general, we do not always have a way to "divide out" matrices.
  - So the fact that $AB = AC$ does not imply $B = C$.
- Finally $AB = 0$ does not in general imply that one of A or B must be 0.
  - For example, A might apply a zero-scale on the first coordinate, and B might apply a zero-scale on the rest of the coordinates.

rank

- rank of product lemmas.

  Lemma. *Suppose that $C = AB$. Then $Col(C) \subseteq Col(A)$ and so $Rank(C) \leq Rank(A)$.*

  Lemma. *Suppose that $C = AB$. Then $Null(C) \supseteq Null(B)$ and so $Nullity(C) \geq Nullity(B)$ and so $Rank(C) \leq Rank(B)$.*

- if $B\mathbf{v} = \mathbf{0}$, then $C\mathbf{v} = AB\mathbf{v} = \mathbf{0}$.
- C and B have the same number of columns, so with rank nullity, we can take a nullity inequality and flip it to get a rank inequality.

back to manipulation

- Given a matrix A. Suppose that there was some matrix R such that $AR = I$.
  - R is a kind of inverse of A.
- Them given the equation $BA = CA$, we could multiply both sides on by R on the right to get

$$\begin{aligned} BA &= CA \\ BAR &= CAR \\ BI &= CI \\ B &= C \end{aligned}$$

- if $BA = 0$ then we could compute

$$\begin{aligned} BA &= 0 \\ BAR &= 0R \\ BI &= 0 \\ B &= 0 \end{aligned}$$

- so having this kind of inverse allows for useful manipulations.

...

- Clearly not all matrices have this nice property.

$$A := \begin{bmatrix} 1 & 0 \\ 5 & 0 \end{bmatrix}$$

- Then for any B, let $C := AB$. We must have $\mathbf{c}_1 = k\mathbf{a}_1$ for some k.
- So there can not by any B such that $AB = I$.
- this property will soon be called "right invertibility"

Right Invertibility

Definition. Let A be an m-by-n matrix. We say that A is *right invertible* if there is an matrix n-by-m matrix R so that $AR = I_m$. ■

Theorem. *Let A be an m-by-n matrix. Then A is right invertible iff the map determined by A is surjective.*

proof

Proof. Surjectivity is the same as $Col(A) = \mathbb{R}^m$. So for any indicator vector $\mathbf{e}_j \in \mathbb{R}^m$, (fixing j), there are set of n coefficients r_{ij} (i varies) so that

$$\mathbf{e}_j = \sum_i r_{ij}\mathbf{a}_i$$

(The set of coefficients need not be unique).
This is the same as

$$\begin{aligned} \mathbf{e}_j &= [\mathbf{a}_1\mathbf{a}_2...\mathbf{a}_n]\begin{bmatrix} r_{1j} \\ r_{2j} \\ ... \\ r_{nj} \end{bmatrix} \\ &=: [\mathbf{a}_1\mathbf{a}_2...\mathbf{a}_n]\mathbf{r}_j \end{aligned}$$

And so we can find a set of $\mathbf{r}_j$ such that

$$[\mathbf{e}_1\mathbf{e}_2...\mathbf{e}_m] = [\mathbf{a}_1\mathbf{a}_2...\mathbf{a}_n][\mathbf{r}_1\mathbf{r}_2...\mathbf{r}_m]$$

Defining the n-by$m$ matrix R using the columns $\mathbf{r}_i$ this gives us $I_m = AR$.
...
For the other direction: Assume that A is right invertible, so there is a matrix R with $AR = I$. Let $\mathbf{v}$ be any vector in $\mathbb{R}^m$. From right invertibility we have

$$\begin{aligned} \mathbf{v} &= I\mathbf{v} \\ &= AR\mathbf{v} \\ &= A(R\mathbf{v}) \\ &=: A\mathbf{c} \end{aligned}$$

But this means that

$$\mathbf{v} = \sum_i c_i\mathbf{a}_i$$

putting $\mathbf{v} \in Col(A)$, and so we can obtain any vector from $\mathbb{R}^m$ in $Col(A)$. This is surjectivity. □

Left Invertibility

Definition. Let A be an m-by-n matrix. We say that A is *left invertible* if there is an matrix n-by-m matrix L so that $LA = I_n$. ■

Theorem. *Let A be an m-by-n matrix, with columns $\mathbf{a}_i$. Then A is left invertible iff the map determined by A is injective.*

proof

Proof. Assume that A is left invertible, so there is a matrix L with $LA = I_n$. Thus

$$(LA)\mathbf{k} = I_n\mathbf{k} = \mathbf{k}.$$

Thus the null space of LA must be trivial ($\{\mathbf{0}\}$). But from a rank of product lemma, we must have $Null(LA) \supseteq Null(A)$, makeing the null space of A trivial, giving it linearly independent columns. This is the same as injectivity. □

- the other direction is in the book.

invertible matrix theorem

- In summary:

  Theorem (invertible matrix theorem)**.** *Let A be an m-by-n matrix. The following properties are equivalent:*

  1. *The map determined by A is injective.*
  2. *$Nullity(A) = 0$.*
  3. *The columns, $\mathbf{a}_i$ form a linearly independent set.*
  4. *A is never ambiguous.*
  5. *A is left invertible*

  The following properties are also equivalent

  1. The map determined by A is surjective.
  2. $Rank(A) = m$.
  3. The columns, $\mathbf{a}_i$ span $\mathbb{R}^m$.
  4. A, is always solvable.
  5. A is right invertible

  If A is square, and so $m = n$, then all of these 10 properties are equivalent.

rank of invertible product

Theorem. *Suppose that $C = AB$. Suppose that B is right invertible. Then $Col(C) = Col(A)$ and so $Rank(C) = Rank(A)$.*

Theorem. *Suppose that $C = AB$. Suppose that A is left invertible. Then $Null(C) = Null(B)$ and so $Nullity(C) = Nullity(B)$ and so $Rank(C) = Rank(B)$.*

- how would you prove this?

invertiblilty

Lemma. *Let A be a square n-by-n matrix. Suppose that there is a matrix L so that $LA = I$ and also suppose that there is a matrix R so that $AR = I$. Then $L = R$.*

Proof. We can compute

$$L = LI = L(AR) = (LA)R = IR = R$$

□

- now suppose that you have two left inverses L_1 and L_2 and a right inverse R.
- then $L_1 = R = L_2$. so the left inverse is unique.

- and any right inverse must equal this unique matrix.
- So for a matrix that is both left and right invertible, there is a unique matrix that is both its left and right inverse.

non-singular

This then motivates the following

Definition. Let A be a square n-by-n matrix. We say that A is *invertible* or *non-singular* if there is a matrix, denoted by A^{-1} so that $A^{-1}A = AA^{-1} = I$. In this case we call such a A^{-1} the *inverse* of A. Otherwise we say that A is *singular*. ■

- we can add this to our above invertable matrix theorem.
- we often deal with the square case, and in this setting, we mostly just talk about invertibility and an inverse, and don't bother with the whole left-right issues.
- Later on in this book, we will learn about even more matrix properties that are equivalent to non-singularity.

properties

Theorem. *Behold:*

- *If A is an invertible matrix, then $(A^{-1})^{-1} = A$.*
- *If A and B are two invertible matrix of the same size, then AB is invertible with $(AB)^{-1} = B^{-1}A^{-1}$.*

Proof. The first statement follows from the definition of inverse. For the second statement, we observe that

$$\begin{aligned} I &= B^{-1}B \\ &= B^{-1}(A^{-1}A)B \\ &= (B^{-1}A^{-1})(AB) \end{aligned}$$

□

- BQ: how would you compute $(ABC)^{-1}$? [Hint: first think of AB as a single matrix. Use the above rule. Then use it again.]

Algorithms

- matrix-vector multiplication directly from the form of its efinition.
  - You should think of this (in the square case) as generally taking roughly kn^2 steps, for some k that does not depend on n.
  - We write this as $O(n^2)$.
- we can implement matrix-matrix multiplication, directly from the form of its definition.
  - You should think of this (in the square case) as generally taking roughly $O(n^3)$ steps
  - There are methods that are faster then this for *sparse* matrices, with lot and lots of 0 entries.
  - There are some methods that are faster than $O(n^3)$ for general matrices, which (theoretically) would be better to use for sufficiently giant matrices.

more algs

- there are efficient techniques that given A can compute $Rank(A)$, and $Nullity(A)$.
- There are efficient techniques to compute a basis for $Col(A)$ and $Null(A)$.
- for an invertible matrix A, there are efficient techniques to compute A^{-1}.
  - You should think of the inversion process as taking $O(n^3)$ steps.

solving a lin sys

- Suppose we with to solve a system of equations $A\mathbf{x} = \mathbf{b}$, where A is invertible. Then we know that there must be a solution $\mathbf{x}$. And for this $\mathbf{x}$ we can reason:

$$
\begin{aligned}
A\mathbf{x} &= \mathbf{b} \\
A^{-1}A\mathbf{x} &= A^{-1}\mathbf{b} \\
\mathbf{x} &= A^{-1}\mathbf{b}
\end{aligned}
$$

- Thus $A^{-1}\mathbf{b}$ must be the unique solution to this system.
- Suppose I change the rhs $\mathbf{b}$ vector for the linear system, but I do not change A.
- Then to solve the new system, I do not have to recompute the inverse A^{-1}. I just have to perform the matrix-vector multiply $A^{-1}\mathbf{b}$.
  - Matrix-vector multiplication is much faster than inverting a matrix, so the second system can be solved more quickly.

...

- there are other methods for solving non-singular linearly systems that considered superior to computing an inverse.
- In particular, some of these methods have superior numerical accuracy and/or are more efficient.
- Sometimes, there are special algorithms that work really well for certain special classes of matrices. This is an entire field of study.
- When in doubt, the conventional wisdom is that a good way to solve a linear system is using something called LU-decomposition, which we will develop now.

permuation matrix

Definition. A *permutation* matrix P is a square matrix with exactly one 1 per row and exactly one 1 per column. The remaining entries are all 0. ■

- The vector $P\mathbf{x}$ is the same as $\mathbf{x}$ except that its entries are permuted.
- The inverse of a permutation is also a permutation.

triangular

Definition. A square matrix U is *upper triangular* if $u_{ij} = 0$ for $i > j$. A square matrix L is *lower triangular* if $l_{ij} = 0$ for $i < j$. ■

- a triangular matrix is non-singular iff all of its diagonal entries are non-zero.
- if U is a non-singular upper triangular matrix we can solve a system $\mathbf{y} = U\mathbf{x}$ using back substitution.
- Similarly, if L is a non-singular lower triangular, we can solve a system $\mathbf{c} = L\mathbf{y}$ using a "forward substitution".

LU decomp

Theorem. *Given a square matrix A. There exists an upper triangular matrix U, a lower triangular matrix L and a permutation matrix P^{-1} so that $A = P^{-1}LU$. This expression is called the* LU-decomposition *of A.*

- (ex. if A is non-singular, then so too are L and U.)
- Moreover, there are $O(n^3)$ algorithms for computing an LU-decomposition.

solving with LU

- Suppose we want to solve a system

$$
\begin{aligned}
\mathbf{b} &= A\mathbf{x} = P^{-1}LU\mathbf{x} && (3) \\
P\mathbf{b} &= LU\mathbf{x} && (4)
\end{aligned}
$$

- Step 1: First let us compute $\mathbf{c} := P\mathbf{b}$. Now we wish to solve the system

$$
\mathbf{c} = LU\mathbf{x}
$$

- Step 2: To continue, let us next solve the system

$$\mathbf{c} = L\mathbf{y} \tag{5}$$

where $\mathbf{y}$ is a new variable. Suppose we find a solution $\mathbf{y}_1$, so we know that $\mathbf{c} = L\mathbf{y}_1$.

...

- Step 3: To continue, let us next solve the system

$$\mathbf{y}_1 = U\mathbf{x} \tag{6}$$

And Suppose we find a solution $\mathbf{x}_1$.

Now we can compute

$$\begin{aligned} P^{-1}LU\mathbf{x}_1 &= P^{-1}L(U\mathbf{x}_1) \\ &= P^{-1}L\mathbf{y}_1 \\ &= P^{-1}(L\mathbf{y}_1) \\ &= P^{-1}\mathbf{c} \\ &= \mathbf{b} \end{aligned}$$

and thus $\mathbf{x}_1$ must be a solution to our original linear system of Equation 3!

..

- the matrix U is upper triangular so this system is already in echelon form
- The U system can be solved using backsubstitution, which only takes $O(n^2)$ steps.
- Similarly L is lower triangular, so the L system can also be solved in $O(n^2)$ steps using a similar algorithm called forward substitution.
- So the only $O(n^3)$ step is the computation of the LU-decomposition.
- during substitution steps, we can also determine if system is unsolvable
- if I change my system by replacing the $\mathbf{b}$ vector, but I do not alter the A matrix, then the LU-decomposition does not need to be recomputed,
  - only have to do the remaining $O(n^2)$ work to solve the new system.

MATLAB

```
b = rand(3,1);
A = [2, -7, 2; 0, 1, -3; 0, 0,  1];
B = [20, -2, 2; 0, 1, -3; 0, 0,  1];
O = A*B*b;

Ainv = inv(A);
[L,U,P] = lu(A);
x=A\b;
```

- The command `rand(3,1)` creates a random 3-by-1 matrix, which we may also treat as a vector.
- The operator `*` applies matrix-matrix multiplication.
- the command `inv(A)` returns A^{-1} if it exists. If the matrix is singular, a warning is produced
- The command `[L,U,P] = lu(A)` computes the LU-decomposition of A. It returns three output matrices. Remember that the actual decomposition is $P^{-1}LU = A$.
- The command `x=A\b` solves the linear system $A\mathbf{x} = \mathbf{b}$. MATLAB will choose what algorithm to use. For example, when A is triangular, then it will use a fast substitution method.