Quiz

- Give our two primary interpretations of matrix-vector multiplication.
- Give the matrix-vector definition of matrix-matrix multiplication.
- Let $A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & 1 & 2 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$.
  1. What is the rank of A?
  2. What is the dimension of the solution set of the equation $A\mathbf{x} = \mathbf{0}$?
  3. Give a basis for the solution set.
  4. Express the solution set of the equation $A\mathbf{x} = [15, 9, 3]$ as the translation of a subspace.

Vector-matrix multiplication

We interpret m-vector as $m \times 1$ matrix ("column vector") $\mathbf{v} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$

Can transpose to get $1 \times m$ matrix ("row vector") $\mathbf{v}^T = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$. We know two ways to interpret matrix-vector multiplication:

- **Linear-combinations interpretation:** $A\mathbf{v}$ is linear combination of columns of A
- **Dot-product interpretation:** Entry r of $A\mathbf{v}$ is dot-product of row r of A with $\mathbf{v}$.

What about multiplying a row vector by a matrix? $\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 1 & 5 \\ 1 & 10 \\ 1 & 4 \end{bmatrix}$

Use transpose rule ($(AB)^T = B^T A^T$):

$$\left(\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 1 & 5 \\ 1 & 10 \\ 1 & 4 \end{bmatrix} \right)^T = \begin{bmatrix} 1 & 5 \\ 1 & 10 \\ 1 & 4 \end{bmatrix}^T \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}^T = \begin{bmatrix} 1 & 1 & 1 \\ 5 & 10 & 4 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

Now can use familiar interpretations of matrix-vector product

Can we obtain interpretations of original vector-matrix product?

Vector-matrix multiplication

We interpret m-vector as $m \times 1$ matrix ("column vector") $\mathbf{v} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$

Can transpose to get $1 \times m$ matrix ("row vector") $\mathbf{v}^T = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$. We know two ways to interpret matrix-vector multiplication:

- **Linear-combinations interpretation:** $A\mathbf{v}$ is linear combination of columns of A
- **Dot-product interpretation:** Entry r of $A\mathbf{v}$ is dot-product of row r of A with $\mathbf{v}$.

What about multiplying a row vector by a matrix? $\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 1 & 5 \\ 1 & 10 \\ 1 & 4 \end{bmatrix}$

Use transpose rule ($(AB)^T = B^T A^T$):

$$\left(\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 1 & 5 \\ 1 & 10 \\ 1 & 4 \end{bmatrix} \right)^T = \begin{bmatrix} 1 & 5 \\ 1 & 10 \\ 1 & 4 \end{bmatrix}^T \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}^T = \begin{bmatrix} 1 & 1 & 1 \\ 5 & 10 & 4 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

Now can use familiar interpretations of matrix-vector product

Can we obtain interpretations of original vector-matrix product?

Vector-matrix multiplication

We interpret m-vector as $m \times 1$ matrix ("column vector") $\mathbf{v} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$

Can transpose to get $1 \times m$ matrix ("row vector") $\mathbf{v}^T = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$. We know two ways to interpret matrix-vector multiplication:

- **Linear-combinations interpretation:** $A\mathbf{v}$ is linear combination of columns of A
- **Dot-product interpretation:** Entry r of $A\mathbf{v}$ is dot-product of row r of A with $\mathbf{v}$.

What about multiplying a row vector by a matrix? $\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 1 & 5 \\ 1 & 10 \\ 1 & 4 \end{bmatrix}$

Use transpose rule ($(AB)^T = B^T A^T$):

$$\left(\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 1 & 5 \\ 1 & 10 \\ 1 & 4 \end{bmatrix} \right)^T = \begin{bmatrix} 1 & 5 \\ 1 & 10 \\ 1 & 4 \end{bmatrix}^T \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}^T = \begin{bmatrix} 1 & 1 & 1 \\ 5 & 10 & 4 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

Now can use familiar interpretations of matrix-vector product

Can we obtain interpretations of original vector-matrix product?

Quiz

For each item in the left column, list every letter corresponding to an item in the right column that *always* fits.
By *fits*, I don't mean that something is always true. I mean that it makes sense—that it "type-checks".

1. Dimension of ...
2. Rank of ...
3. Span of ...
4. Col space of ...
5. Row space of ...
6. Null space of ...
7. kernel of ...
8. ... is one-to-one
9. ... is onto
10. ... is invertible
11. ... is trivial

(a) vector space/subspace
(b) affine space
(c) vector
(d) matrix
(e) linear combination
(f) linear transformation
(g) set of vectors

Quiz, continued: Uniqueness of representation.

Let $\mathbf{b}_1, \ldots, \mathbf{b}_n$ be a basis for a vector space $\mathcal{V}$. Use the definition of *basis* to show that each vector in $\mathcal{V}$ has exactly one (at least one and at most one) representation in terms of $\mathbf{b}_1, \ldots, \mathbf{b}_n$.

Echelon form

Definition: An $m \times n$ matrix A is in *echelon form* if it satisfies the following condition: for any row, if that row's first nonzero entry is in position k then every previous row's first nonzero entry is in some position less than k.

This definition implies that, as you iterate through the rows of A, the first nonzero entries per row move strictly right, forming a sort of staircase that descends to the right.

$$\begin{bmatrix} 0 & 2 & 3 & 0 & 5 & 6 \\ 0 & 0 & 1 & 0 & 3 & 4 \\ 0 & 0 & 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 0 & 0 & 9 \end{bmatrix}$$

2	1	0	4	1	3	9	7
0	6	0	1	3	0	4	1
0	0	0	0	2	1	3	2
0	0	0	0	0	0	0	1

$$\begin{bmatrix} 4 & 1 & 3 & 0 \\ 0 & 3 & 0 & 1 \\ 0 & 0 & 1 & 7 \\ 0 & 0 & 0 & 9 \end{bmatrix}$$

Sorting rows by position of the leftmost nonzero

Goal: a method of transforming a rowlist into one that is in echelon form.

First attempt: Sort the rows by position of the leftmost nonzero entry.

We will use a naive algorithm of sorting:

- first choose a row with a nonzero in first column,
- then choose a row with a nonzero in second column,

⋮

accumulating these in a list `new_rowlist`, initially empty:

```
new_rowlist = []
```

The algorithm maintains the set of indices of rows remaining to be sorted, `rows_left`, initially consisting of all the row indices:

```
rows_left = set(range(len(rowlist)))
```

Sorting rows by position of the leftmost nonzero

```
col_label_list = sorted(rowlist[0].D, key=str)
new_rowlist = []
rows_left = set(range(len(rowlist)))
```

- Algorithm iterates through the column labels in order.
- In each iteration, algorithm finds a list

  `rows_with_nonzero`

  of indices of the remaining rows that have nonzero entries in the current column
- Algorithm selects one of these rows (the *pivot row*), adds it to `new_rowlist`, and removes its index from `rows_left`.

```
for c in col_label_list:
  rows_with_nonzero = [r for r in rows_left if rowlist[r][c] != 0]
  pivot = rows_with_nonzero[0]
  new_rowlist.append(rowlist[pivot])
  rows_left.remove(pivot)
```

Sorting rows by position of the leftmost nonzero

```
for c in col_label_list:
    rows_with_nonzero = [r for r in rows_left if rowlist[r][c] != 0]
    if rows_with_nonzero != []:
        pivot = rows_with_nonzero[0]
        new_rowlist.append(rowlist[pivot])
        rows_left.remove(pivot)
```

Run the algorithm on

$$\begin{bmatrix} 0 & 2 & 3 & 4 & 5 \\ 0 & 0 & 0 & 0 & 5 \\ 1 & 2 & 3 & 4 & 5 \\ 0 & 0 & 0 & 4 & 5 \end{bmatrix}$$

new_rowlist

- After first two iterations, new_rowlist is $[[1,2,3,4,5],[0,2,3,4,5]]$, and rows_left is $\{1,3\}$.
- The algorithm runs into trouble in third iteration since none of the remaining rows have a nonzero in column 2.
- In this case, the algorithm should just move on to the next column without changing new_rowlist or rows_left.

Sorting rows by position of the leftmost nonzero

```
for c in col_label_list:
    rows_with_nonzero = [r for r in rows_left if rowlist[r][c] != 0]
    if rows_with_nonzero != []:
        pivot = rows_with_nonzero[0]
        new_rowlist.append(rowlist[pivot])
        rows_left.remove(pivot)
```

Run the algorithm on

$$\begin{bmatrix} 0 & 2 & 3 & 4 & 5 \\ 0 & 0 & 0 & 0 & 5 \\ 1 & 2 & 3 & 4 & 5 \\ 0 & 0 & 0 & 4 & 5 \end{bmatrix}$$

`new_rowlist`

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \end{bmatrix}$$

- After first two iterations, `new_rowlist` is $[[1, 2, 3, 4, 5], [0, 2, 3, 4, 5]]$, and `rows_left` is $\{1, 3\}$.
- The algorithm runs into trouble in third iteration since none of the remaining rows have a nonzero in column 2.
- In this case, the algorithm should just move on to the next column without changing `new_rowlist` or `rows_left`.

Sorting rows by position of the leftmost nonzero

```
for c in col_label_list:
    rows_with_nonzero = [r for r in rows_left if rowlist[r][c] != 0]
    if rows_with_nonzero != []:
        pivot = rows_with_nonzero[0]
        new_rowlist.append(rowlist[pivot])
        rows_left.remove(pivot)
```

Run the algorithm on

$$\begin{bmatrix} 0 & 2 & 3 & 4 & 5 \\ 0 & 0 & 0 & 0 & 5 \\ 1 & 2 & 3 & 4 & 5 \\ 0 & 0 & 0 & 4 & 5 \end{bmatrix}$$

new_rowlist

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 0 & 2 & 3 & 4 & 5 \end{bmatrix}$$

- After first two iterations, new_rowlist is $[[1,2,3,4,5],[0,2,3,4,5]]$, and rows_left is $\{1,3\}$.
- The algorithm runs into trouble in third iteration since none of the remaining rows have a nonzero in column 2.
- In this case, the algorithm should just move on to the next column without changing new_rowlist or rows_left.

Sorting rows by position of the leftmost nonzero

```
for c in col_label_list:
    rows_with_nonzero = [r for r in rows_left if rowlist[r][c] != 0]
    if rows_with_nonzero != []:
        pivot = rows_with_nonzero[0]
        new_rowlist.append(rowlist[pivot])
        rows_left.remove(pivot)
```

Run the algorithm on

$$\begin{bmatrix} 0 & 2 & 3 & 4 & 5 \\ 0 & 0 & 0 & 0 & 5 \\ 1 & 2 & 3 & 4 & 5 \\ 0 & 0 & 0 & 4 & 5 \end{bmatrix}$$

`new_rowlist`

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 0 & 2 & 3 & 4 & 5 \\ 0 & 0 & 0 & 4 & 5 \end{bmatrix}$$

- After first two iterations, `new_rowlist` is $[[1,2,3,4,5],[0,2,3,4,5]]$, and `rows_left` is $\{1,3\}$.
- The algorithm runs into trouble in third iteration since none of the remaining rows have a nonzero in column 2.
- In this case, the algorithm should just move on to the next column without changing `new_rowlist` or `rows_left`.

Sorting rows by position of the leftmost nonzero

```
for c in col_label_list:
    rows_with_nonzero = [r for r in rows_left if rowlist[r][c] != 0]
    if rows_with_nonzero != []:
        pivot = rows_with_nonzero[0]
        new_rowlist.append(rowlist[pivot])
        rows_left.remove(pivot)
```

Run the algorithm on

$$\begin{bmatrix} 0 & 2 & 3 & 4 & 5 \\ 0 & 0 & 0 & 0 & 5 \\ 1 & 2 & 3 & 4 & 5 \\ 0 & 0 & 0 & 4 & 5 \end{bmatrix}$$

new_rowlist

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 0 & 2 & 3 & 4 & 5 \\ 0 & 0 & 0 & 4 & 5 \\ 0 & 0 & 0 & 0 & 5 \end{bmatrix}$$

- After first two iterations, new_rowlist is $[[1, 2, 3, 4, 5], [0, 2, 3, 4, 5]]$, and rows_left is $\{1, 3\}$.
- The algorithm runs into trouble in third iteration since none of the remaining rows have a nonzero in column 2.
- In this case, the algorithm should just move on to the next column without changing new_rowlist or rows_left.

Flaw in sorting

```
for c in col_label_list:
    rows_with_nonzero = [r for r in rows_left if rowlist[r][c] != 0]
    if rows_with_nonzero != []:
        pivot = rows_with_nonzero[0]
        new_rowlist.append(rowlist[pivot])
        rows_left.remove(pivot)
```

$$\overset{\texttt{rowlist}}{\begin{bmatrix} 0 & 2 & 3 & 4 & 5 \\ 0 & 0 & 0 & 3 & 2 \\ 1 & 2 & 3 & 4 & 5 \\ 0 & 0 & 0 & 6 & 7 \end{bmatrix}} \Rightarrow \overset{\texttt{new\_rowlist}}{\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 0 & 2 & 3 & 4 & 5 \\ 0 & 0 & 0 & 3 & 2 \\ 0 & 0 & 0 & 6 & 7 \end{bmatrix}}$$

Result is not in echelon form.

Need to introduce another transformation....

Elementary row-addition operations

$$\begin{bmatrix} 0 & 2 & 3 & 4 & 5 \\ 0 & 0 & 0 & 3 & 2 \\ 1 & 2 & 3 & 4 & 5 \\ 0 & 0 & 0 & 6 & 7 \end{bmatrix} \Rightarrow \begin{bmatrix} 0 & 2 & 3 & 4 & 5 \\ 0 & 0 & 0 & 3 & 2 \\ 1 & 2 & 3 & 4 & 5 \\ 0 & 0 & 0 & 0 & 3 \end{bmatrix}$$

Repair the problem by *changing* the rows:

Subtract twice the second row

$$2\,[0, 0, 0, 3, 2]$$

from the fourth

$$[0, 0, 0, 6, 7]$$

gettting new fourth row

$$[0, 0, 0, 6, 7] - 2\,[0, 0, 0, 3, 2] = [0, 0, 0, 6 - 6, 7 - 4] = [0, 0, 0, 0, 3]$$

The 3 in the second row is called the *pivot element*.
That element is used to zero out another element in same column.

Elementary row-addition operations

Transformation is multiplication by a *elementary row-addition matrix*:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -2 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 2 & 3 & 4 & 5 \\ 0 & 0 & 0 & 3 & 2 \\ 1 & 2 & 3 & 4 & 5 \\ 0 & 0 & 0 & 6 & 7 \end{bmatrix} = \begin{bmatrix} 0 & 2 & 3 & 4 & 5 \\ 0 & 0 & 0 & 3 & 2 \\ 1 & 2 & 3 & 4 & 5 \\ 0 & 0 & 0 & 0 & 3 \end{bmatrix}$$

Such a matrix is invertible:

$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -2 & 0 & 1 \end{bmatrix}$ and $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 2 & 0 & 1 \end{bmatrix}$ are inverses.

We will show: **Proposition:** If $MA = B$ where M is invertible then Row A = Row B.

Therefore change to row causes no change in row space.

Therefore basis for changed rowlist is also a basis for original rowlist.

Preserving row space

Lemma: Row $NA \subseteq$ Row A.

Proof: Let $\mathbf{v}$ be any vector in Row NA.
That is, $\mathbf{v}$ is a linear combination of the rows of NA.
By the linear-combinations definition of vector-matrix multiplication, there is a vector $\mathbf{u}$ such that

$$\mathbf{v} = \left[\quad \mathbf{u}^T \quad \right] \left(\left[\quad N \quad \right] \left[\quad A \quad \right] \right)$$

$$= \left(\left[\quad \mathbf{u}^T \quad \right] \left[\quad N \quad \right] \right) \left[\quad A \quad \right] \qquad \text{by associativity}$$

which shows that $\mathbf{v}$ can be written as a linear combination of the rows of A. QED

Preserving row space

Lemma: Row $NA \subseteq$ Row A.

Proposition: If M is invertible then Row $MA =$ Row A

Proof: Must show Row $MA \subseteq$ Row A and Row $A \subseteq$ Row MA

- Lemma shows Row $MA \subseteq$ Row A.
- Let $B = MA$
- M has an inverse M^{-1} $\quad\Rightarrow\quad$ $M^{-1}B = A$
- Lemma shows Row $\underbrace{M^{-1}B}_{A} \subseteq$ Row $\underbrace{B}_{MA}$
- That is, Row $A \subseteq$ Row MA QED

Gaussian elimination

Applying elementary row-addition operations does not change the row space.

Incorporate into the algorithm

```
for c in col_label_list:
  rows_with_nonzero = [r for r in rows_left if rowlist[r][c] != 0]
    if rows_with_nonzero != []:
      pivot = rows_with_nonzero[0]
      rows_left.remove(pivot)
      new_rowlist.append(rowlist[pivot])
      add suitable multiple of rowlist[pivot] to each row in rows_with_nonzero[1:]
```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 5 \\ 3 & 4 & 5 & 6 \\ 4 & 5 & 6 & 7 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & -1 & -2 & -3 \\ 0 & -2 & -4 & -6 \\ 0 & -3 & -6 & -9 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & -1 & -2 & -3 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

To avoid problems due to round-off error, let's work over GF(2)

Gaussian elimination

Applying elementary row-addition operations does not change the row space.

Incorporate into the algorithm

```
for c in col_label_list:
  rows_with_nonzero = [r for r in rows_left if rowlist[r][c] != 0]
    if rows_with_nonzero != []:
      pivot = rows_with_nonzero[0]
      rows_left.remove(pivot)
      new_rowlist.append(rowlist[pivot])
      for r in rows_with_nonzero[1:]:
        multiplier = rowlist[r][c]/rowlist[pivot][c]
        rowlist[r] -= multiplier * rowlist[pivot]
```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 5 \\ 3 & 4 & 5 & 6 \\ 4 & 5 & 6 & 7 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & -1 & -2 & -3 \\ 0 & -2 & -4 & -6 \\ 0 & -3 & -6 & -9 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & -1 & -2 & -3 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

To avoid problems due to round-off error, let's work over GF(2)

Gaussian elimination for $GF(2)$

		A	B	C	D
	0	0	0	1	1
✓	1	1	0	1	1
	2	1	0	0	1
	3	1	1	1	1

A: Select row 1 as pivot.
Put it in `new_rowlist`
Since rows 2 and 3 have nonzeroes, we must add row 1 to rows 2 and 3.

`new_rowlist`

$$\begin{bmatrix} 1 & 0 & 1 & 1 \end{bmatrix}$$

		A	B	C	D
	0	0	0	1	1
✓	1	1	0	1	1
	2	0	0	1	0
✓	3	0	1	0	0

B: Select row 3 as pivot.
Put it in `new_rowlist`
Other remaining rows have zeroes in column B, so no row additions needed.

`new_rowlist`

$$\begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

		A	B	C	D
✓	0	0	0	1	1
✓	1	1	0	1	1
	2	0	0	1	0
✓	3	0	1	0	0

C: Select row 0 as pivot .
Put it in `new_rowlist`.
Only other remaining row is row 2, and we add row 0 to row 2.

`new_rowlist`

$$\begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

Gaussian elimination for $GF(2)$

		A	B	C	D
✓	0	0	0	1	1
✓	1	1	0	1	1
✓	2	0	0	0	1
✓	3	0	1	0	0

We are done.

D: Only remaining row is row 2, so select it as pivot row.
Put it in `new_rowlist`
No other rows, so no row additions.

`new_rowlist`

$$\begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

`new_rowlist`

$$\begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$