Transforming a matrix to echelon form

Lemma: If matrix is in echelon form, the nonzero rows form a basis for row space.

Suggests an approach: To find basis for row space of a matrix A, iteratively transform A into a matrix B

- in echelon form
- with no zero rows
- whose row space is the same as that of A.

We will represent current matrix as a rowlist.

Assume `rowlist` has been initialized with a list of Vecs, e.g..

$$\texttt{rowlist} = \left[\begin{array}{ccc} A & B & C \\ \hline 0 & 1 & 2 \end{array}, \quad \begin{array}{ccc} A & B & C \\ \hline 1 & 2 & 3 \end{array}, \quad \begin{array}{ccc} A & B & C \\ \hline 0 & 0 & 1 \end{array} \right]$$

We will mutate this variable.

To handle Vecs with arbitrary D, must decide on an ordering:

```
col_label_list = sorted(rowlist[0].D, key=hash)
```

First attempt: Sorting rows by position of the leftmost nonzero

Goal: Transform a matrix `rowlist` into a matrix `new_rowlist` in echelon form.

Here's an easy matrix to start with:

	A	B	C	D	E	F
0	0	0	0	0	1	2
1	0	2	3	0	5	6
2	0	0	0	0	0	0
3	0	0	1	0	3	4

$\longrightarrow$

	A	B	C	D	E	F
0	0	2	3	0	5	6

Suggests an algorithm: sort the rows according to position of leftmost nonzero entry.

First attempt: Sorting rows by position of the leftmost nonzero

Goal: Transform a matrix `rowlist` into a matrix `new_rowlist` in echelon form.

Here's an easy matrix to start with:

	A	B	C	D	E	F
0	0	0	0	0	1	2
1	0	2	3	0	5	6
2	0	0	0	0	0	0
3	0	0	1	0	3	4

$\longrightarrow$

	A	B	C	D	E	F
0	0	2	3	0	5	6
1	0	0	1	0	3	4

Suggests an algorithm: sort the rows according to position of leftmost nonzero entry.

First attempt: Sorting rows by position of the leftmost nonzero

Goal: Transform a matrix `rowlist` into a matrix `new_rowlist` in echelon form.

Here's an easy matrix to start with:

	A	B	C	D	E	F
0	0	0	0	0	1	2
1	0	2	3	0	5	6
2	0	0	0	0	0	0
3	0	0	1	0	3	4

$\longrightarrow$

	A	B	C	D	E	F
0	0	2	3	0	5	6
1	0	0	1	0	3	4
2	0	0	0	0	1	2

Suggests an algorithm: sort the rows according to position of leftmost nonzero entry.

First attempt: Sorting rows by position of the leftmost nonzero

Goal: Transform a matrix `rowlist` into a matrix `new_rowlist` in echelon form.

Here's an easy matrix to start with:

	A	B	C	D	E	F
0	0	0	0	0	1	2
1	0	2	3	0	5	6
2	0	0	0	0	0	0
3	0	0	1	0	3	4

$\longrightarrow$

	A	B	C	D	E	F
0	0	2	3	0	5	6
1	0	0	1	0	3	4
2	0	0	0	0	1	2
3	0	0	0	0	0	0

Suggests an algorithm: sort the rows according to position of leftmost nonzero entry.

First attempt: Sorting rows by position of the leftmost nonzero

Goal: Transform a matrix `rowlist` into a matrix `new_rowlist` in echelon form.

Here's an easy matrix to start with:

	A	*B*	*C*	*D*	*E*	*F*
0	0	0	0	0	1	2
1	0	2	3	0	5	6
2	0	0	0	0	0	0
3	0	0	1	0	3	4

$\longrightarrow$

	A	*B*	*C*	*D*	*E*	*F*
0	0	2	3	0	5	6
1	0	0	1	0	3	4
2	0	0	0	0	1	2
3	0	0	0	0	0	0

Suggests an algorithm: sort the rows according to position of leftmost nonzero entry.

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,

$\vdots$

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=hash)
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]
    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]
    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]
    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]
    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.

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)
```

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

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

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

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

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

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:

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

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

$$\mathbf{v} = \begin{bmatrix} \quad \mathbf{u}^T \quad \end{bmatrix} \left(\begin{bmatrix} \\ \quad N \quad \\ \\ \end{bmatrix} \begin{bmatrix} \\ \quad A \quad \\ \\ \end{bmatrix} \right)$$

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

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 $\underbrace{\text{Row } M^{-1}B}_{A} \subseteq \underbrace{\text{Row } B}_{MA}$
- That is, Row $A \subseteq MA$ 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:]
```

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

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]
```

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

This algorithm is mathematically correct...

Failure of Gaussian elimination

But we compute using floating-point numbers!

$$\begin{bmatrix} 10^{-20} & 0 & 1 \\ 1 & 10^{20} & 1 \\ 0 & 1 & -1 \end{bmatrix} \Rightarrow \begin{bmatrix} 10^{-20} & 0 & 1 \\ 0 & 10^{20} & 1-10^{20} \\ 0 & 1 & -1 \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} 10^{-20} & 0 & 1 \\ 0 & 10^{20} & -10^{20} \\ 0 & 1 & -1 \end{bmatrix} \Rightarrow \begin{bmatrix} 10^{-20} & 0 & 1 \\ 0 & 10^{20} & -10^{20} \\ 0 & 0 & 0 \end{bmatrix}$$

Gaussian elimination got the wrong answer due to round-off error.

These problems can be mitigated by choosing the pivot element carefully:

- *Partial pivoting:* Among rows with nonzero entries in column c, choose row with entry having *largest* absolute value.
- *Complete pivoting:* Instead of selecting order of columns beforehand, in each iteration choose column to maximize absolute value of pivot element.

In this course, we won't study these techniques in detail.
Instead, we will use Gaussian elimination only for $GF(2)$.

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	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	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	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	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	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	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	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	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 $GF(2)$

new_rowlist

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

		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

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

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`

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

Using Gaussian elimination for other problems

So far:

- we know how to use Gaussian elimination to transform a matrix into echelon form;
- nonzero rows form a basis for row space of original matrix

We can do other things with Gaussian elimination:

- Solve linear systems (used in e.g. *Lights Out*)
- Find vectors in null space (used in e.g. integer factoring)

Key idea: keep track of transformations performed in putting matrix in echelon form.