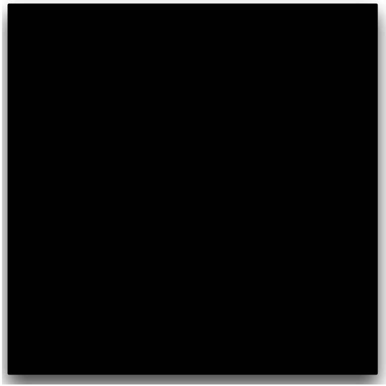
[7] Gaussian Elimination

Starting to peek inside the black box

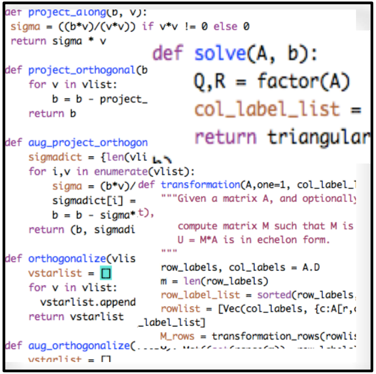
So far `solve(A, b)` is a black box.



With Gaussian elimination, we begin to find out what's inside.

Starting to peek inside the black box

So far `solve(A, b)` is a black box.



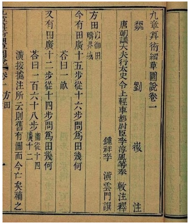
With Gaussian elimination, we begin to find out what's inside.

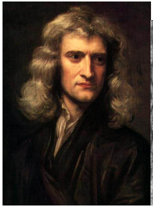
Gaussian Elimination: Origins

Method illustrated in Chapter Eight of a Chinese text, *The Nine Chapters on the Mathematical Art*, that was written roughly two thousand years ago.

Rediscovered in Europe by Isaac Newton (England) and Michel Rolle (France)

Gauss called the method *eliminiationem vulgarem* ("common elimination")
Gauss adapted the method for another problem (one we study soon) and developed notation.


九章筭術細草圖說卷一
魏 劉 徽 注
唐朝議大夫行太史令上輕車都尉臣李淳風等奉 敕注釋
鍾祥李 潢雲門譔
方田 以御田疇界域
今有田廣十五步從十六步問爲田幾何
荅曰一畝
又有田廣十二步從十四步問爲田幾何
荅曰一百六十八步 圖從十四廣十二
潢按據注所云則舊有圖而今亡矣補之



Gaussian elimination: Uses

- *Finding a basis for the span of given vectors*. This additionally gives us an algorithm for rank and therefore for testing linear dependence.
- *Solving a matrix equation*,which is the same as *expressing a given vector as a linear combination of other given vectors*, which is the same as *solving a system of linear equations*
- *Finding a basis for the null space of a matrix*, which is the same as *finding a basis for the solution set of a homogeneous linear system*, which is also relevant to representing the solution set of a general linear system.

Echelon form

Echelon form a generalization of triangular matrices

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

Note that

- the first nonzero entry in row 0 is in column 1,
- the first nonzero entry in row 1 is in column 2,
- the first nonzero entry in row 2 is in column 4, and
- the first nonzero entry in row 4 is in column 5.

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.

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.

$$\begin{array}{|c|c|c|c|c|c|c|c|}\hline 2 & 1 & 0 & 4 & 1 & 3 & 9 & 7 \\ \hline 0 & 6 & 0 & 1 & 3 & 0 & 4 & 1 \\ \hline 0 & 0 & 0 & 0 & 2 & 1 & 3 & 2 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ \hline \end{array}$$

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

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 any previous row's first nonzero entry is in some position less than k.

If a row of a matrix in echelon form is all zero then every subsequent row must also be all zero, e.g.

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

Uses of echelon form

What good is it having a matrix in echeleon form?

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

For example, a basis for the row space of

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

is $\{[0, 2, 3, 0, 5, 6], [0, 1, 0, 3, 4]\}$.
In particular, if every row is nonzero, as in each of the matrices

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

then the rows form a basis of the row space.

Uses of echelon form

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

It is obvious that the nonzero rows span the row space. We need only show that these vectors are linearly independent. We prove it using the Grow algorithm:

```
def GROW(𝒱)
  S = ∅
  repeat while possible:
      find a vector v in 𝒱 that is not in Span S, and put it in S
```

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

We run the Grow algorithm, adding rows of matrix in reverse order to S:

- Since Span $\emptyset$ does not include $[0, 0, 0, 9]$, the algorithm adds this vector to S.
- Now $S = \{[0, 0, 0, 9]\}$. Every vector in Span S has zeroes in positions 0, 1, 2, so Span S does not contain $[0, 0, 1, 7]$, so the algorithm adds this vector to S.
- Now $S = \{[0, 0, 0, 9], [0, 0, 1, 7]\}$. Every vector in Span S has zeroes in positions 0, 1, so Span S does not contain $[0, 3, 0, 1]$, so the algorithm adds it.
- Now $S = \{[0, 0, 0, 9], [0, 0, 1, 7], [0, 3, 0, 1]\}$. Every vector in Span S has a zero in position 0, so Span S does not contain $[4, 1, 3, 0]$, so the algorithm adds it, and we are done.

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=str)
```

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

	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.

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

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

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

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

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

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

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

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

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 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:]
```

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

This algorithm is mathematically correct...

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 $GF(2)$

`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`

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

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.

Gaussian Elimination: Solving system of equations

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

Given matrix A, compute matrices M and U such that $MA = U$

- U is in echelon form
- M is invertible

To solve $A\mathbf{x} = \mathbf{b}$:

- Compute M and U so that $MA = U$
- Compute the matrix-vector product $M\mathbf{b}$, and solve $U\mathbf{x} = M\mathbf{b}$.

Claim: This gives correct solution to $A\mathbf{x} = \mathbf{b}$

Proof: Suppose $\mathbf{v}$ is a solution to $U\mathbf{x} = M\mathbf{b}$, so $U\mathbf{v} = M\mathbf{b}$

- Multiply both sides by M^{-1}: $M^{-1}(U\mathbf{v}) = M^{-1}M\mathbf{b}$
- Use associativity: $(M^{-1}U)\mathbf{v} = (M^{-1}M)\mathbf{b}$
- Cancel M^{-1} and M: $(M^{-1}U)\mathbf{v} = \mathbb{1}\mathbf{b}$
- Use $M^{-1}U = A$: $A\mathbf{v} = \mathbb{1}\mathbf{b} = \mathbf{b}$

How to solve $U\mathbf{x} = M\mathbf{b}$?

- If U is triangular, can solve using *back-substitution* (`triangular_solve`)
- In general, can use similar algorithm

Gaussian Elimination: Finding basis for null space

Instead of finding basis for null space of A, find basis for $\{\mathbf{u} \;:\; \mathbf{u} * A = \mathbf{0}\} = \text{Null } A^T$

Input:

	A	B	C	D
0	1	0	1	0
1	1	1	1	0
2	0	1	0	1
3	1	1	1	1
4	0	0	0	1

Find M such that the matrix $U = MA$ is in echelon form and M is invertible

$$\underbrace{\begin{array}{c|ccccc} & 0 & 1 & 2 & 3 & 4 \\ \hline 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 2 & 1 & 1 & 1 & 0 & 0 \\ 3 & 1 & 0 & 1 & 1 & 0 \\ 4 & 1 & 1 & 1 & 0 & 1 \end{array}}_{M} * \underbrace{\begin{array}{c|cccc} & A & B & C & D \\ \hline 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 2 & 0 & 1 & 0 & 1 \\ 3 & 1 & 1 & 1 & 1 \\ 4 & 0 & 0 & 0 & 1 \end{array}}_{A} = \underbrace{\begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 2 & 0 & 0 & 0 & 1 \\ 3 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 \end{array}}_{U}$$

Last two rows of U are zero vectors

- Row 3 of U is (row 3 of M) * A
- Row 4 of U is (row 4 of M) * A

Gaussian Elimination: Finding basis for null space

Find M such that the matrix $U = MA$ is in echelon form and M is invertible

$$\underbrace{\begin{array}{c|ccccc} & 0 & 1 & 2 & 3 & 4 \\ \hline 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 2 & 1 & 1 & 1 & 0 & 0 \\ 3 & 1 & 0 & 1 & 1 & 0 \\ 4 & 1 & 1 & 1 & 0 & 1 \end{array}}_{M} * \underbrace{\begin{array}{c|cccc} & A & B & C & D \\ \hline 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 2 & 0 & 1 & 0 & 1 \\ 3 & 1 & 1 & 1 & 1 \\ 4 & 0 & 0 & 0 & 1 \end{array}}_{A} = \underbrace{\begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 2 & 0 & 0 & 0 & 1 \\ 3 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 \end{array}}_{U}$$

Last two rows of U are zero vectors

- Row 3 of U is (row 3 of M) * A
- Row 4 of U is (row 4 of M) * A

Therefore two rows in $\{\mathbf{u} \ : \ \mathbf{u} * A = \mathbf{0}\}$ are rows 3 and 4 of M
To show that these two rows form a basis for $\{\mathbf{u} \ : \ \mathbf{u} * A = \mathbf{0}\}$....
$\dim \text{Row } A = 3$
By Rank-Nullity Theorem, $\dim \text{Row } A + \dim \text{Null } A^T =$ number of rows $= 5$
Shows that $\dim \text{Null } A^T = 2$
Since M is invertible, all its rows are linearly independent.

Gaussian elimination: recording the transformations

$$\begin{bmatrix} & & \\ & A & \\ & & \end{bmatrix} = \begin{bmatrix} & & \\ & U_1 & \\ & & \end{bmatrix}$$

$$\begin{bmatrix} & & \\ & M_1 & \\ & & \end{bmatrix}\begin{bmatrix} & & \\ & A & \\ & & \end{bmatrix} = \begin{bmatrix} & & \\ & U_2 & \\ & & \end{bmatrix}$$

$$\begin{bmatrix} & & \\ & M_2 & \\ & & \end{bmatrix}\begin{bmatrix} & & \\ & M_1 & \\ & & \end{bmatrix}\begin{bmatrix} & & \\ & A & \\ & & \end{bmatrix} = \begin{bmatrix} & & \\ & U_3 & \\ & & \end{bmatrix}$$

$$\begin{bmatrix} & & \\ & M_3 & \\ & & \end{bmatrix}\begin{bmatrix} & & \\ & M_2 & \\ & & \end{bmatrix}\begin{bmatrix} & & \\ & M_1 & \\ & & \end{bmatrix}\begin{bmatrix} & & \\ & A & \\ & & \end{bmatrix} = \begin{bmatrix} & & \\ & U_4 & \\ & & \end{bmatrix}$$

Gaussian elimination: recording the transformations

$$\begin{bmatrix} 0 & 2 & 4 & 2 & 8 \\ 2 & 1 & 0 & 5 & 4 \\ 4 & 1 & 2 & 4 & 2 \\ 5 & 0 & 0 & 2 & 8 \end{bmatrix} = \begin{bmatrix} 0 & 2 & 4 & 2 & 8 \\ 2 & 1 & 0 & 5 & 4 \\ 4 & 1 & 2 & 4 & 2 \\ 5 & 0 & 0 & 2 & 8 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & \text{-2} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 2 & 4 & 2 & 8 \\ 2 & 1 & 0 & 5 & 4 \\ 4 & 1 & 2 & 4 & 2 \\ 5 & 0 & 0 & 2 & 8 \end{bmatrix} = \begin{bmatrix} 0 & 2 & 4 & 2 & 8 \\ 2 & 1 & 0 & 5 & 4 \\ 0 & -1 & 2 & -6 & -6 \\ 5 & 0 & 0 & 2 & 8 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & \text{-2.5} & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & \text{-2} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 2 & 4 & 2 & 8 \\ 2 & 1 & 0 & 5 & 4 \\ 4 & 1 & 2 & 4 & 2 \\ 5 & 0 & 0 & 2 & 8 \end{bmatrix} = \begin{bmatrix} 0 & 2 & 4 & 2 & 8 \\ 2 & 1 & 0 & 5 & 4 \\ 0 & -1 & 2 & -6 & -6 \\ 0 & -2.5 & 0 & -10.5 & -2 \end{bmatrix}$$

Gaussian elimination: recording the transformations

$$\begin{bmatrix} 0 & 2 & 4 & 2 & 8 \\ 2 & 1 & 0 & 5 & 4 \\ 4 & 1 & 2 & 4 & 2 \\ 5 & 0 & 0 & 2 & 8 \end{bmatrix} = \begin{bmatrix} 0 & 2 & 4 & 2 & 8 \\ 2 & 1 & 0 & 5 & 4 \\ 4 & 1 & 2 & 4 & 2 \\ 5 & 0 & 0 & 2 & 8 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & \text{-2} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 2 & 4 & 2 & 8 \\ 2 & 1 & 0 & 5 & 4 \\ 4 & 1 & 2 & 4 & 2 \\ 5 & 0 & 0 & 2 & 8 \end{bmatrix} = \begin{bmatrix} 0 & 2 & 4 & 2 & 8 \\ 2 & 1 & 0 & 5 & 4 \\ 0 & -1 & 2 & -6 & -6 \\ 5 & 0 & 0 & 2 & 8 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & \text{-2} & 1 & 0 \\ 0 & \text{-2.5} & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 2 & 4 & 2 & 8 \\ 2 & 1 & 0 & 5 & 4 \\ 4 & 1 & 2 & 4 & 2 \\ 5 & 0 & 0 & 2 & 8 \end{bmatrix} = \begin{bmatrix} 0 & 2 & 4 & 2 & 8 \\ 2 & 1 & 0 & 5 & 4 \\ 0 & -1 & 2 & -6 & -6 \\ 0 & -2.5 & 0 & -10.5 & -2 \end{bmatrix}$$

Gaussian elimination: recording the transformations

$$\begin{bmatrix} 0 & 2 & 4 & 2 & 8 \\ 2 & 1 & 0 & 5 & 4 \\ 4 & 1 & 2 & 4 & 2 \\ 5 & 0 & 0 & 2 & 8 \end{bmatrix} = \begin{bmatrix} 0 & 2 & 4 & 2 & 8 \\ 2 & 1 & 0 & 5 & 4 \\ 4 & 1 & 2 & 4 & 2 \\ 5 & 0 & 0 & 2 & 8 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & \text{-2} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 2 & 4 & 2 & 8 \\ 2 & 1 & 0 & 5 & 4 \\ 4 & 1 & 2 & 4 & 2 \\ 5 & 0 & 0 & 2 & 8 \end{bmatrix} = \begin{bmatrix} 0 & 2 & 4 & 2 & 8 \\ 2 & 1 & 0 & 5 & 4 \\ 0 & -1 & 2 & -6 & -6 \\ 5 & 0 & 0 & 2 & 8 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & \text{-2} & 1 & 0 \\ 0 & \text{-2.5} & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 2 & 4 & 2 & 8 \\ 2 & 1 & 0 & 5 & 4 \\ 4 & 1 & 2 & 4 & 2 \\ 5 & 0 & 0 & 2 & 8 \end{bmatrix} = \begin{bmatrix} 0 & 2 & 4 & 2 & 8 \\ 2 & 1 & 0 & 5 & 4 \\ 0 & -1 & 2 & -6 & -6 \\ 0 & -2.5 & 0 & -10.5 & -2 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ .5 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & \text{-2} & 1 & 0 \\ 0 & \text{-2.5} & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 2 & 4 & 2 & 8 \\ 2 & 1 & 0 & 5 & 4 \\ 4 & 1 & 2 & 4 & 2 \\ 5 & 0 & 0 & 2 & 8 \end{bmatrix} = \begin{bmatrix} 0 & 2 & 4 & 2 & 8 \\ 2 & 1 & 0 & 5 & 4 \\ 0 & 0 & 4 & -5 & -2 \\ 0 & -2.5 & 0 & -10.5 & -2 \end{bmatrix}$$

Gaussian elimination: recording the transformations

$$\begin{bmatrix} 0 & 2 & 4 & 2 & 8 \\ 2 & 1 & 0 & 5 & 4 \\ 4 & 1 & 2 & 4 & 2 \\ 5 & 0 & 0 & 2 & 8 \end{bmatrix} = \begin{bmatrix} 0 & 2 & 4 & 2 & 8 \\ 2 & 1 & 0 & 5 & 4 \\ 4 & 1 & 2 & 4 & 2 \\ 5 & 0 & 0 & 2 & 8 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & \text{-2} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 2 & 4 & 2 & 8 \\ 2 & 1 & 0 & 5 & 4 \\ 4 & 1 & 2 & 4 & 2 \\ 5 & 0 & 0 & 2 & 8 \end{bmatrix} = \begin{bmatrix} 0 & 2 & 4 & 2 & 8 \\ 2 & 1 & 0 & 5 & 4 \\ 0 & -1 & 2 & -6 & -6 \\ 5 & 0 & 0 & 2 & 8 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & \text{-2} & 1 & 0 \\ 0 & \text{-2.5} & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 2 & 4 & 2 & 8 \\ 2 & 1 & 0 & 5 & 4 \\ 4 & 1 & 2 & 4 & 2 \\ 5 & 0 & 0 & 2 & 8 \end{bmatrix} = \begin{bmatrix} 0 & 2 & 4 & 2 & 8 \\ 2 & 1 & 0 & 5 & 4 \\ 0 & -1 & 2 & -6 & -6 \\ 0 & -2.5 & 0 & -10.5 & -2 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ .5 & \text{-2} & 1 & 0 \\ 0 & \text{-2.5} & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 2 & 4 & 2 & 8 \\ 2 & 1 & 0 & 5 & 4 \\ 4 & 1 & 2 & 4 & 2 \\ 5 & 0 & 0 & 2 & 8 \end{bmatrix} = \begin{bmatrix} 0 & 2 & 4 & 2 & 8 \\ 2 & 1 & 0 & 5 & 4 \\ 0 & 0 & 4 & -5 & -2 \\ 0 & -2.5 & 0 & -10.5 & -2 \end{bmatrix}$$

Gaussian elimination: recording the transformations

- Maintain M (initially identity) and U (initially A)
- Whatever transformations you do to U, do same transformations to M

$$
\begin{array}{c|cccc}
 & 0 & 1 & 2 & 3 \\ \hline
0 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 \\
2 & 0 & 0 & 1 & 0 \\
3 & 0 & 0 & 0 & 1
\end{array}
\;*\;
\begin{array}{c|ccccc}
 & A & B & C & D & \\ \hline
0 & 0 & 0 & 1 & 1 & \\
1 & 1 & 0 & 1 & 1 & \checkmark \\
2 & 1 & 0 & 0 & 1 & \\
3 & 1 & 1 & 1 & 1 &
\end{array}
\;=\;
\begin{array}{c|cccc}
 & A & B & C & D \\ \hline
0 & 0 & 0 & 1 & 1 \\
1 & 1 & 0 & 1 & 1 \\
2 & 1 & 0 & 0 & 1 \\
3 & 1 & 1 & 1 & 1
\end{array}
$$

ColumnA:
select row 1
add it to rows 2,3

$$
\begin{array}{c|cccc}
 & 0 & 1 & 2 & 3 \\ \hline
0 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 \\
2 & 0 & 1 & 1 & 0 \\
3 & 0 & 1 & 0 & 1
\end{array}
\;*\;
\begin{array}{c|ccccc}
 & A & B & C & D & \\ \hline
0 & 0 & 0 & 1 & 1 & \checkmark \\
1 & 1 & 0 & 1 & 1 & \checkmark \\
2 & 1 & 0 & 0 & 1 & \\
3 & 1 & 1 & 1 & 1 & \checkmark
\end{array}
\;=\;
\begin{array}{c|cccc}
 & A & B & C & D \\ \hline
0 & 0 & 0 & 1 & 1 \\
1 & 1 & 0 & 1 & 1 \\
2 & 0 & 0 & 1 & 0 \\
3 & 0 & 1 & 0 & 0
\end{array}
$$

ColumnB:
select row 3
add it to no rows

ColumnC:
select row 0
add it to row 2

$$
\begin{array}{c|cccc}
 & 0 & 1 & 2 & 3 \\ \hline
0 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 \\
2 & 1 & 1 & 1 & 0 \\
3 & 0 & 1 & 0 & 1
\end{array}
\;*\;
\begin{array}{c|ccccc}
 & A & B & C & D & \\ \hline
0 & 0 & 0 & 1 & 1 & \checkmark \\
1 & 1 & 0 & 1 & 1 & \checkmark \\
2 & 1 & 0 & 0 & 1 & \checkmark \\
3 & 1 & 1 & 1 & 1 & \checkmark
\end{array}
\;=\;
\begin{array}{c|cccc}
 & A & B & C & D \\ \hline
0 & 0 & 0 & 1 & 1 \\
1 & 1 & 0 & 1 & 1 \\
2 & 0 & 0 & 0 & 1 \\
3 & 0 & 1 & 0 & 0
\end{array}
$$

ColumnD:
select row 2
done

Code for finding transformation to echelon form

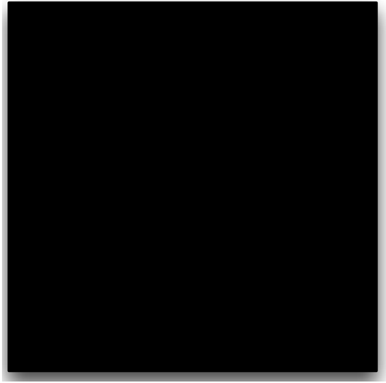
- Initialize `rowlist` to be list of rows of A
- Initialize `M_rowlist` to be list of rows of identity matrix

```
for c in sorted(col_labels, key=str):
    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_M_rowlist.append(M_rowlist[pivot])
        for r in rows_with_nonzero[1:]:
            multiplier = rowlist[r][c]/rowlist[pivot][c]
            rowlist[r] -= multiplier*rowlist[pivot]
            M_rowlist[r] -= multiplier*M_rowlist[pivot]

    for r in rows_left: new_M_rowlist.append(M_rowlist[r])
```

Finally, return matrix M formed from `M_rowlist`
Code provided in module `echelon`

The black box starts to become less opaque

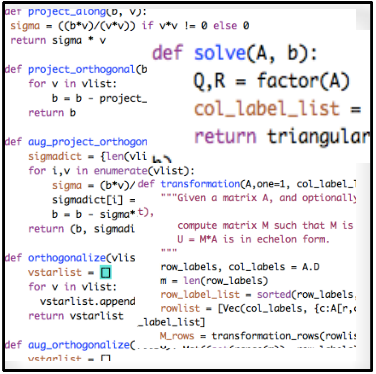


The modules `independence` and `solver` both Gaussian elimination when working over $GF(2)$:

- The procedure `solve(A, b)` computes a matrix M such that MA is in echelon form, and uses M to try to find a solution.
- The procedure `rank(L)` converts to echelon form and counts the nonzero rows to find the rank of L.

We saw that Gaussian elimination can be used to find a nonzero vector in the null space of a matrix.... You will use this in an algorithm for factoring integers.

The black box starts to become less opaque



The modules `independence` and `solver` both Gaussian elimination when working over $GF(2)$:

- The procedure `solve(A, b)` computes a matrix M such that MA is in echelon form, and uses M to try to find a solution.
- The procedure `rank(L)` converts to echelon form and counts the nonzero rows to find the rank of L.

We saw that Gaussian elimination can be used to find a nonzero vector in the null space of a matrix.... You will use this in an algorithm for factoring integers.

Factoring integers

Prime Factorization Theorem: For every integer $N \geq 1$, there is a unique bag of prime numbers whose product is N.

Example:

- 75 is the product of the elements in the bag $\{3, 5, 5\}$
- 126 is the product of the elements in the bag $\{2, 3, 3, 7\}$
- 23 is the product of the elements in the bag $\{23\}$

All the elements in a bag must be prime. If N is itself prime, the bag for N is just $\{N\}$.

"The problem of distinguishing prime numbers from composite numbers and of **resolving the latter into their prime factors** is known to be one of the most important and useful in arithmetic. It has engaged the industry and wisdom of ancient and modern geometers to such an extent that it would be superfluous to discuss the problem at length Further, the dignity of the science itself seems to require solution of a problem so elegant and so celebrated."

Carl Friedrich Gauss, Disquisitiones Arithmeticae, 1801

Factoring integers

Prime Factorization Theorem: For every integer $N \geq 1$, there is a unique bag of prime numbers whose product is N.

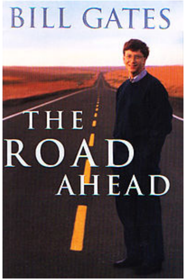
Example:

- 75 is the product of the elements in the bag $\{3, 5, 5\}$
- 126 is the product of the elements in the bag $\{2, 3, 3, 7\}$
- 23 is the product of the elements in the bag $\{23\}$

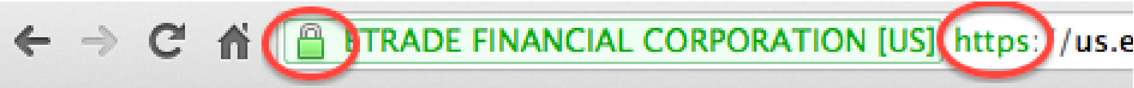
All the elements in a bag must be prime. If N is itself prime, the bag for N is just $\{N\}$.

"Because both the system's privacy and the security of digital money depend on encryption, a breakthrough in mathematics or computer science that defeats the cryptographic system could be a disaster. The obvious mathematical breakthrough would be the development of an easy way to factor large prime numbers."

(Bill Gates, The Road Ahead, *1995).*



Secure Sockets Layer



Secure communication with websites uses HTTPS (Secure HTTP)

which is based on SSL (Secure Sockets Layer)

which is based on the RSA (Rivest-Shamir-Adelman) cryptosystem

which depends on the computational difficulty of factoring integers

Factoring integers

Testing whether a number is prime is now well-understood and easy.

Here's a one-line Python script that gives false positives when input is a Carmichael number (rare) and otherwise with probability $\frac{1}{2^{20}}$:

```
def is_prime(p, n=20): return all([pow(randint(1,p-1),p-1,p) == 1
                                   for i in range(n)])}
```

With a few more lines, can get correct answers for Carmichael numbers as well.

The hard part of factoring seems to be this: given an integer N, find any *nontrivial* divisor (divisor other than 1 and N).

If you can do that reliably, you can factor.

Factoring integers the naive way

```
def factor(N):
  for d in range(2, N-1):
    if N % d == 0: return d
```

If d is a divisor of N then so is N/d.

$\min\{d, N/d\} \leq \sqrt{N}$

This shows that it suffices to search among $2, 3, \ldots, \text{int}(\sqrt{N})$

```
def factor(N):
  for d in range(2, intsqrt(N)):
    if N % d == 0: return d
```

where `intsqrt(N)` is a procedure I provide

Useful subroutine: `gcd(m,n)`

`gcd(m,n)` return the greatest common divisor of positive integers m and n. This algorithm is attributed to Euclid, and it is very fast. Here's the code:

```
def gcd(x,y): return x if y == 0 else gcd(y, x % y)
```

Example:

- $\gcd(12, 16)$ is 4
- $\gcd(276534813447635747652, 333070702552660863114)$ is 18172055646

Using square roots to factor N

Find integers a and b such that

$$a^2 - b^2 = N$$

for then

$$(a-b)(a+b) = N$$

so $a-b$ and $a+b$ are divisors (ideally nontrivial)

How to find such integers? Naive approach...

- Choose integer a slightly more than $\sqrt{N}$
- Check if $\sqrt{a^2 - N}$ is an integer.
- If so, let $b = \sqrt{a^2 - N}$ Success! ☺
  Now $a - b$ is a divisor of N
- If not, repeat with another value for a

Example: $N = 77$

$a = 9$
$\sqrt{a^2 - N} = \sqrt{4} = 2$
so let $b = 2$
$a - b = 7$ is a divisor of N

Example: $N = 23 \cdot 41$
$a = 31 \Rightarrow a^2 - N = 18$ ☹
$a = 32 \Rightarrow a^2 - N = 81$ ☺

For large N, it takes too long to find a good integer a. ☹
We will show how **linear algebra** ☺ helps us synthesize a good integer a.

Using square roots to factor N

Find a and b such that

$$a^2 - b^2 = kN$$

for some integer k. Then

$$(a-b)(a+b) = kN$$

$$\{\text{prime factors of } a-b\} \cup \{\text{prime factors of } a+b\} = \{\text{prime factors of } k\} \cup \{\text{prime factors of } N\}$$

Suppose $\{\text{prime factors of } N\} = \{p, q\}$.

If

- p and q are both factors of $a-b$, or
- p and q are both factors of $a+b$

then $\gcd(a-b, N)$ will not find a nontrivial divisor.
However, if

- p is a factor of $a-b$ and q is a factor of $a+b$, or
- p is a factor of $a+b$ and q is a factor of $a-b$

then $\gcd(a-b, N)$ **will** find a nontrivial divisor.

Example:: $N = 7 \cdot 11$
$k = 2 \cdot 3 \cdot 5 \cdot 13$

if $a - b = 2 \cdot 7 \cdot 11$ and
$a + b = 3 \cdot 5 \cdot 13$
then $\gcd(a-b, N) = N$ ☹

if $a - b = 2 \cdot 5 \cdot 11$ and
$a + b = 3 \cdot 7 \cdot 13$

then $\gcd(a-b, N) = 11$ ☺

How to find integers a, b such that $a^2 - b^2 = kN$

Idea: Start by finding the first thousand prime numbers $p_1, \ldots, p_{1000}$.

- Choose a
- Compute $a^2 - N$.
- See if $a^2 - N$ can be factored using only $p_1, \ldots, p_{1000}$
- If not, throw it away.
- If so, record a and the factorization of $a^2 - N$

Repeat a thousand and one times

a	$a * a - N$	factorization
51	182	$2 \cdot 7 \cdot 13$
52	285	$3 \cdot 5 \cdot 19$
53	390	$2 \cdot 3 \cdot 5 \cdot 13$
58	945	$3^3 \cdot 5 \cdot 7$
61	1302	$2 \cdot 3 \cdot \cdot 7 \cdot 13$
62	1425	$3 \cdot 5^2 \cdot 19$
63	1550	$2 \cdot 5^2 \cdot 31$
67	2070	$2 \cdot 3^2 \cdot 5 \cdot 23$
68	2205	$3^2 \cdot 5 \cdot 7^2$
71	2622	$2 \cdot 3 \cdot 19 \cdot 23$

Now we want to find a subset $\{a_1, \ldots, a_k\}$ such that $(a_1^2 - N) \cdots (a_k^2 - N)$ is a perfect square.

Combine $a_1 = 52$, $a_2 = 67$, $a_3 = 71$

$$(a_1^2 - N)(a_2^2 - N)(a_3^2 - N) =$$
$$(3 \cdot 5 \cdot 19)(2 \cdot 3^2 \cdot 5 \cdot 23)(2 \cdot 3 \cdot 19 \cdot 23)$$
$$= 2^2 \cdot 3^4 \cdot 5^2 \cdot 19^2 \cdot 23^2 = (2 \cdot 3^2 \cdot 5 \cdot 19 \cdot 23)^2$$

How to find a subset that works?

Finding a subset that works

Represent each factorization as a vector over $GF(2)$:
Represent $p_1^{a_1} p_2^{a_2} \cdots p_k^{a_k}$ by $\{p_1 : (a_1 \ \% \ 2), p_2 : (a_2 \ \% \ 2) \ldots, p_k : (a_k \ \% \ 2)\}$
Let $A =$ matrix whose rows are these vectors.
A subset of factorizations whose product is a perfect square = a subset of A's rows whose sum is the zero vector
Therefore need to find a nonzero vector in $\{\mathbf{u} \ : \ \mathbf{u} * A = \mathbf{0}\}$
If number of rows > rank of matrix then there exists such a nonzero vector.

a	$a * a - N$	factorization	vector.f
51	182	$2 \cdot 7 \cdot 13$	$\{2 : \texttt{one}, 13 : \texttt{one}, 7 : \texttt{one}\}$
52	285	$3 \cdot 5 \cdot 19$	$\{19 : \texttt{one}, 3 : \texttt{one}, 5 : \texttt{one}\}$
53	390	$2 \cdot 3 \cdot 5 \cdot 13$	$\{2 : \texttt{one}, 3 : \texttt{one}, 5 : \texttt{one}, 13 : \texttt{one}\}$
58	945	$3^3 \cdot 5 \cdot 7$	$\{3 : \texttt{one}, 5 : \texttt{one}, 7 : \texttt{one}\}$
61	1302	$2 \cdot 3 \cdot \cdot 7 \cdot 13$	$\{31 : \texttt{one}, 2 : \texttt{one}, 3 : \texttt{one}, 7 : \texttt{one}\}$
62	1425	$3 \cdot 5^2 \cdot 19$	$\{19 : \texttt{one}, 3 : \texttt{one}, 5 : 0\}$
63	1550	$2 \cdot 5^2 \cdot 31$	$\{2 : \texttt{one}, 5 : 0, 31 : \texttt{one}\}$
67	2070	$2 \cdot 3^2 \cdot 5 \cdot 23$	$\{2 : \texttt{one}, 3 : 0, 5 : \texttt{one}, 23 : \texttt{one}\}$
68	2205	$3^2 \cdot 5 \cdot 7^2$	$\{3 : 0, 5 : \texttt{one}, 7 : 0\}$
71	2622	$2 \cdot 3 \cdot 19 \cdot 23$	$\{19 : \texttt{one}, 2 : \texttt{one}, 3 : \texttt{one}, 23 : \texttt{one}\}$

Other uses of Gaussian elimination over $GF(2)$

Simple examples of other uses of Gaussian elimination over $GF(2)$:

- Solving *Lights Out* puzzles.
- Attacking Python's pseudo-random-number generator:

  ```
  >>> import random
  >>> random.getrandbits(32)
  1984256916
  >>> random.getrandbits(32)
  4135536776
  >>> random.getrandbits(32)
  [redacted]
  ```

  What are the next thirty-two bits to be generated? Using Gaussian elimination, you can predict them accurately.
- Breaking simple authentication scheme (playing the role of Eve)....

Improving on the simple authentication scheme

- Password is an n-vector $\hat{\mathbf{x}}$ over $GF(2)$
- **Challenge:** Computer sends random n-vector $\mathbf{a}$
- **Response:** Human sends back $\mathbf{a} \cdot \hat{\mathbf{x}}$.

Repeated until Computer is convinced that Human knowns password $\hat{\mathbf{x}}$.

Eve eavesdrops on communication,
learns m pairs $\mathbf{a}_1, b_1, \ldots, \mathbf{a}_m, b_m$
such that b_i is right response to challenge $\mathbf{a}_i$

The password $\hat{\mathbf{x}}$ is a solution to

$$\underbrace{\begin{bmatrix} \mathbf{a}_1 \\ \hline \vdots \\ \hline \mathbf{a}_m \end{bmatrix}}_{A} \begin{bmatrix} \mathbf{x} \end{bmatrix} = \underbrace{\begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix}}_{\mathbf{b}}$$

Once rank A reaches n, the solution is unique, and Eve can use Gaussian elimination to find it, obtaining the password.

Making the scheme more secure:

The way to make the scheme more secure is to introduce mistakes.

- In about $1/6$ of the rounds, randomly, Human sends the *wrong* dot-product.
- Computer is convinced if Human gets the right answers 75% of the time.

Even if Eve knows that Human is making mistakes, she doesn't know which rounds involve mistakes.

Gaussian elimination does not find the solution when some of the right-hand side values b_i are wrong.

In fact, we don't know *any* efficient algorithm Eve can use to find the solution, even if Eve observes many, many rounds.

Threshold secret-sharing

All-or-nothing secret-sharing is a method to split the secret into two pieces so that both are required to recover the secret.

We could generalize to split the secret among four teaching assistants (TAs), so that jointly they could recover the secret but any three cannot.

However, it is risky to rely on all four TAs showing up for a meeting.

Instead we want a *threshold* secret-sharing scheme: share a secret among four TAs so that

- any three TAs could jointly recover the secret, but
- any two TAs could not.

There are such schemes that use fields other than $GF(2)$, but let's see if we can do it using $GF(2)$.

Threshold secret-sharing using five 3-vectors over $GF(2)$

Failing attempt: Here's an idea: select five 3-vectors over $GF(2)$ $\mathbf{a}_0, \mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3, \mathbf{a}_4$.

These vectors are supposed to satisfy the following requirement:

Requirement: every set of three are linearly independent.

To share a one-bit secret s among the TAs, I randomly select a 3-vector $\mathbf{u}$ such that $\mathbf{a}_0 \cdot \mathbf{u} = s$. I keep $\mathbf{u}$ secret, but I compute the other dot-products:

$$\begin{aligned} \beta_1 &= \mathbf{a}_1 \cdot \mathbf{u} \\ \beta_2 &= \mathbf{a}_2 \cdot \mathbf{u} \\ \beta_3 &= \mathbf{a}_3 \cdot \mathbf{u} \\ \beta_4 &= \mathbf{a}_4 \cdot \mathbf{u} \end{aligned}$$

I give the bit β_1 to TA 1, I give β_2 to TA 2, I give β_3 to TA 3, and I give β_4 to TA 4. The bit given to a TA is her *share*.

Can any three TAs recover the secret? For example, suppose TAs 1, 2, and 3 want to recover the secret. They solve the matrix-vector equation

$$\begin{bmatrix} \mathbf{a}_1 \\ \hline \mathbf{a}_2 \\ \hline \mathbf{a}_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix}$$

Since the matrix is square and the rows $\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3$ are linearly independent, the matrix is is invertible, so $\mathbf{u}$ is the only solution. The TAs use `solve` to recover $\mathbf{u}$, and take the dot-product with $\mathbf{a}_0$ to get the secret $\mathbf{s}$.

Is the secret safe?

Now suppose two rogue TAs, TA 1 and TA 2, decide they want to obtain the secret without involving either of the other TAs. They know β_1 and β_2. Can they use these to get the secret s? The answer is no: their information is consistent with both $s = 0$ and $s = 1$. Since the matrix

$$\begin{bmatrix} \mathbf{a}_0 \\ \hline \mathbf{a}_1 \\ \hline \mathbf{a}_2 \end{bmatrix}$$

is invertible, each of the two matrix equations

$$\begin{bmatrix} \mathbf{a}_0 \\ \hline \mathbf{a}_1 \\ \hline \mathbf{a}_2 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ \beta_1 \\ \beta_2 \end{bmatrix}$$

$$\begin{bmatrix} \mathbf{a}_0 \\ \hline \mathbf{a}_1 \\ \hline \mathbf{a}_2 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ \beta_1 \\ \beta_2 \end{bmatrix}$$

has a solution. The solution to the first equation is a vector $\mathbf{v}$ such that $\mathbf{a}_0 \cdot \mathbf{v} = 0$, and the solution to the second equation is a vector $\mathbf{v}$ such that $\mathbf{a}_0 \cdot \mathbf{v} = 1$.

Threshold secret-sharing with five pairs of 6-vectors

The proposed scheme seems to work. The catch is that first step:

- Select five 3-vectors over $GF(2)$ $\mathbf{a}_0, \mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3, \mathbf{a}_4$ satisfying

Requirement: every set of three are linearly independent.

Unfortunately, there are no such five vectors.

Instead, we seek ten 6-vectors $\mathbf{a}_0, \mathbf{b}_0, \mathbf{a}_1, \mathbf{b}_1, \mathbf{a}_2, \mathbf{b}_2, \mathbf{a}_3, \mathbf{b}_3, \mathbf{a}_4, \mathbf{b}_4$ over $GF(2)$.

We think of them as forming five pairs:

- Pair 0 consists of $\mathbf{a}_0$ and $\mathbf{b}_0$,
- Pair 1 consists of $\mathbf{a}_1$ and $\mathbf{b}_1$,
- Pair 2 consists of $\mathbf{a}_2$ and $\mathbf{b}_2$, and
- Pair 3 consists of $\mathbf{a}_3$ and $\mathbf{b}_3$.
- Pair 4 consists of $\mathbf{a}_4$ and $\mathbf{b}_4$.

The requirement is as follows:

Requirement: For any three pairs, the corresponding six vectors are linearly independent.

To share two bits s and t:

- I choose a secret 6-vector $\mathbf{u}$ such that $\mathbf{a}_0 \cdot \mathbf{u} = s$ and $\mathbf{b}_0 \cdot \mathbf{u} = t$.
- I give TA 1 the two bits $\beta_1 = \mathbf{a}_1 \cdot \mathbf{u}$ and $\gamma_1 = \mathbf{b}_1 \cdot \mathbf{u}$, I give TA 2 the two bits $\beta_2 = \mathbf{a}_2 \cdot \mathbf{u}$ and $\gamma_2 = \mathbf{b}_2 \cdot \mathbf{u}$, and so on.

Each TA's share consists of a pair of bits.

Threshold secret-sharing with five pairs of 6-vectors: recoverability

Any three TAs jointly can solve a matrix-vector equation with a 6×6 matrix to obtain $\mathbf{u}$, whence they can obtain the secret bits s and t. Suppose, for example, TAs 1, 2, and 3 came together. Then they would solve the equation

$$\begin{bmatrix} \mathbf{a}_1 \\ \hline \mathbf{b}_1 \\ \hline \mathbf{a}_2 \\ \hline \mathbf{b}_2 \\ \hline \mathbf{a}_3 \\ \hline \mathbf{b}_3 \end{bmatrix} \begin{bmatrix} \\ \\ \mathbf{x} \\ \\ \\ \end{bmatrix} = \begin{bmatrix} \beta_1 \\ \gamma_1 \\ \beta_2 \\ \gamma_2 \\ \beta_3 \\ \gamma_3 \end{bmatrix}$$

to obtain $\mathbf{u}$ and thereby obtain the secret bits. Since the vectors $\mathbf{a}_1, \mathbf{b}_1, \mathbf{a}_2, \mathbf{b}_2, \mathbf{a}_3, \mathbf{b}_3$ are linearly independent, the matrix is invertible, so there is a unique solution to this equation.

Threshold secret-sharing with five pairs of 6-vectors: recoverability

Suppose TAs 1 and 2 go rogue and try to recover s and t. They possess the bits $\beta_1, \gamma_1, \beta_2, \gamma_2$. Are these bits consistent with $s = 0$ and $t = 1$? They are if there is a vector $\mathbf{u}$ that solves the equation

$$\begin{bmatrix} \mathbf{a}_0 \\ \hline \mathbf{b}_0 \\ \hline \mathbf{a}_1 \\ \hline \mathbf{b}_1 \\ \hline \mathbf{a}_2 \\ \hline \mathbf{b}_2 \end{bmatrix} \begin{bmatrix} \\ \mathbf{x} \\ \\ \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ \beta_1 \\ \gamma_1 \\ \beta_2 \\ \gamma_2 \end{bmatrix}$$

where the first two entries of the right-hand side are the guessed values of s and t.

Since the vectors $\mathbf{a}_0, \mathbf{b}_0, \mathbf{a}_1, \mathbf{b}_1, \mathbf{a}_2, \mathbf{b}_2$ are linearly independent, the matrix is invertible, so a solution exists.

Similarly, no matter what you put in the first two entries of the right-hand side, there is exactly one solution.

This shows that the shares of TAs 1 and 2 tell them nothing about the true values of s and t. The secret is safe.