Finding the number of bases of a vector space $\mathbb{Z}_2^n$

Naïve idea:

Use backtracking to generate all the vectors of $\mathbb{Z}_2^n$, assigning a number to each, then doing another backtracking to generate linearly independent lists of the previously generated vectors.

→ This idea is not only naïve, but also bad and would stop working even for small values of n.

! Idea !

Every vector in $\mathbb{Z}_2^n$ is already an n-bit represent. of an unsigned integer. => We can generate all the combinations of unsigned integers for which the corresponding binary vectors are linearly independent. This alows us to get rid of the first problem, the generating of vectors. For simplicity, I will from now on refer to the number as the "zipped vector".

New task: Find out as many things as possible about the linear independence of a set of vectors without "unzipping" them.

→ What we can do: Check if a vector is 0
Check if 2 vectors are equal
Check if the sum of 2 vectors is equal to a third

We could continue, but it seems rather difficult to catch all exceptions and combinations.

→ What else we can do: Unzip the vectors, put them in a matrix, calculate its rank and check if its rank is equal to the number of vectors in the set.

Unzipping Method:

For python, a way to convert from base 10 to base 2 is the following: {0:0nb}.format(vector)
where 'n' is the number of bits and 'b' is the base specifier ('b' for 'binary')
This returns a string representing the value of the variable vector on n bits.
To make the string into a row on a matrix, we use list({0:0nb}.format(vector))

Calculating Rank

Rank is calculated using the numpy library.
After creating a matrix of all the unzipped vectors, we check if

rank(matrix) == len(list_of_vectors)
↓
length

Intuitive solution for the number of bases of $\mathbb{Z}_2^n$

$n = 1$ only base: (1) Condition $v_1 \neq 0 \Rightarrow$ Solution $= 2^1 - 1 = 1$

$n = 2$ 6 bases $\{((1,0),(0,1)), ((1,0),(1,1)), ((0,1),(1,0)), ((0,1),(1,1)), ((1,1),(0,1)), ((1,1),(1,0))\}$

Conditions $v_1 \neq 0 \Rightarrow 2^2 - 1$
$v_2 \neq v_1 \Rightarrow 2^2 - 1 - 1$
$v_2 \neq 0$
Solution $= (2^2-1)(2^2-2) = 3 \cdot 2 = 6$

$n = 3$? bases

Conditions $v_1 \neq 0 \Rightarrow 2^3 - 1$
$v_2 \neq 0, v_2 \neq v_1 \Rightarrow 2^3 - 2$
$v_3 \neq 0, v_3 \neq v_1, v_3 \neq v_2, v_3 \neq v_1 + v_2 \Rightarrow 2^3 - 4$

Solution <u>should be</u> $= 7 \cdot 6 \cdot 4 = 168$

However, the program outputs 174 :(

<u>Why??</u>

<u>Observation</u>: adding $v_1 + v_2$ in their zipped forms can result in a number that cannot fit on n bits!

$\Rightarrow$ Addition in python <u>does not</u> represent vector addition in $\mathbb{Z}_2^n$

Is there an operation that better represents it? (Other than applying modulo 2 to each result)?

Turns out, yes!

0 xor 0 = 0
0 xor 1 = 1
1 xor 0 = 1
1 xor 1 = 0

$\Rightarrow$ Not only is our method for checking if $v_3 \neq v_1 + v_2$ wrong in this context, but so is the function rank...

Example of the confusion that occured:

$\begin{pmatrix}1\\0\\1\end{pmatrix}\begin{pmatrix}0\\1\\1\end{pmatrix}\begin{pmatrix}1\\1\\0\end{pmatrix}$ got output as a possible basis

$\begin{vmatrix}1&0&1\\0&1&1\\1&1&0\end{vmatrix} = \begin{vmatrix}1&0&1\\0&1&1\\0&1&-1\end{vmatrix} = -2$ If we work with $\mathbb{Z}$!

$L_3' = L_3 - L_1$

but $= \begin{vmatrix}1&0&1\\0&1&1\\0&1&1\end{vmatrix} = 0$ If we work with $\mathbb{Z}_2$!

Getting rid of this mistake accounts for the 6 extra bases we got. $\Rightarrow$ Our formula seems to hold

Let's say we have k vectors from $\mathbb{Z}_2^n$

At this point in our program, we should have the certainty that the first $k-1$ vectors are indeed linearly independent.

As such, their sums must always yield different results

Now, we would have to check for v_k

$v_k \neq 0$ -> 1 vector excluded

$v_k \neq v_i \quad i = \overline{1, k-1}$ -> $k-1$ vectors excluded

$v_k \neq v_i + v_j \quad i = \overline{1, k-1}$, $j = \overline{1, k-1}$, $j \neq i$ -> $\frac{(k-1)(k-2)}{2}$ vectors excluded

$\frac{1}{2}$ because $v_i + v_j = v_j + v_i$

$v_k \neq v_i + v_j + v_l \quad i = \overline{1, k-1}$, $j = \overline{1, k-1}$, $l = \overline{1, k-1}$, $j \neq i, l \neq i, l \neq j$ -> $\frac{(k-1)(k-2)(k-3)}{3!}$ vectors excluded

$\frac{1}{3!}$ because $v_i + v_j + v_l = v_i + v_l + v_j = v_j + v_i + v_l =$

$= v_j + v_l + v_i = v_l + v_i + v_j = v_l + v_j + v_i$

. . . .

$v_k \neq v_i + v_j + v_l + \ldots + v_{final} \quad i, j, \ldots, final = \overline{1, k-1}$, $i \neq j, \ldots$ -> $\frac{(k-1)(k-2)\ldots 2 \cdot 1}{(k-1)!}$ vectors excluded

In total, we exclude $1 + (k-1) + \frac{(k-1)(k-2)}{2!} + \frac{(k-1)(k-2)(k-3)}{3!} + \ldots + \frac{(k-1)(k-2)\ldots 2 \cdot 1}{(k-1)!}$ vectors

$= \binom{0}{k-1} + \binom{1}{k-1} + \binom{2}{k-1} + \ldots + \binom{k-1}{k-1} = 2^{k-1}$

Total available values for v_k such that the list stays linearly independent is $2^n - 2^{k-1}$

⇓

Total number of bases for $\mathbb{Z}_2^n = \prod_{k=1}^{n} (2^n - 2^{k-1}) = \prod_{k=0}^{n-1} (2^n - 2^k)$

=> In conclusion, our program will use this general formula for all inputs of $n (\in \mathbb{N}^*)$ and, for $n \leq 4$ it will also generate the solutions based on the algorithm described previously, with the added checks for $n = 4$.