# SudokuOptim

This project solves a Sudoku board using numerical optimization.

**To see the LaTeX open readme.pdf**

## Motivations

This approach is compatible with DWave quantum computers, and by proxy IBM quantum computers and could be used to solve sudoku using quantum computers. This serves as a showcase of reformulating problems into different forms so that they may be solved using different and more optimal methods. This also helped me understand metaprogramming in Julia. I have been learning many programming languages including C#, Python, C++, Java, Javascript, however metaprogramming is a feature that is unique to Julia and really interesting and refreshing.

## Explanation

This section will discuss the implementation of the algorithm.

**Basic Ideas**

Let's say that we have a four by four sudoku grid:

$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix}$$

Some cells' values are known aka:

$a = 1$, $b = 2$, and etc.

However the solution must satisfy some constraints:

- All rows and columns have unique numbers from 1 to 4
- All 2x2 blocks have unique numbers from 1 to 4

There is a mathematical trick we can use to our advantage when describing these constraints: multiplication is communitive.

So we can rewrite the first row as the equation:

$abcd = 4!$ because $abcd = 1 \cdot 2 \cdot 3 \cdot 4$

So using this fact we can say the constraints are:

$abcd = 4!$
$efgh = 4!$
$\vdots$

For all the rows. And

$aeim = 4!$
$bfjn = 4!$
$\vdots$

For all the columns. And

$$
\begin{aligned}
&abef = 4! \\
&cdgh = 4! \\
&\vdots
\end{aligned}
$$

For all the 2x2 blocks.

Given that $a \dots p \in \{1, 2, 3, 4\}$ these are all the constraints necessary to fully define the system and have only one solution.

**To An Optimization Problem**

To turn this into an optimization problem, we need to create a function where when we minimize it, it will represent the solution to our problem. So we can use subtraction. To apply this to our row constraints:

$$
\begin{aligned}
&f(a, b, c, d, e \dots) = \\
&(abcd - 4!) \\
&+ (efgh - 4!) \\
&\vdots
\end{aligned}
$$

However, there is an issue! if we just use subtraction, this doesn't penalize if $abcd < 4!$. Also there could be some combination like: $(abcd - 4!) = -(efgh - 4!)$ so that $(abcd - 4!) + (efgh - 4!) = 0$ which is also undesirable. So the logical solution would be to use the absolute value, however, the absolute value function isn't 'smooth.' To put this in mathematic terms, the derivative of the absolute value function has a discontinuity. So the commonly accepted alternative is to square the difference. So applying this to our row constraints:

$$
\begin{aligned}
&f(a, b, c, d, e \dots) = \\
&(abcd - 4!)^2 \\
&+ (efgh - 4!)^2 \\
&\vdots
\end{aligned}
$$

So when $f(a, b, c, d, e \dots) = 0$ the row constraints are met.

However, since most numerical optimization methods don't work for integer parameters, we need to add even more constraints to ensure convergence. Example for the rows:

$$
\begin{aligned}
&f(a, b, c, d, e \dots) = \\
&(a + b + c + d - 10)^2 \\
&+ ((a + b + c + d)^2 - 10^2)^2 \\
&\vdots \\
&+ ((a + b + c + d)^4 - 10^4)^2
\end{aligned}
$$

If you do this for all the rows, columns, and 2x2 blocks, there would be more than enough constraints than variables, meaning that all the problem will only have one solution (specifically the integer one that we want).

# Implementation

When I started working on a solution I initially wanted to use Python, however, I realized if I used Julia (a new programming language that supports metaprogramming) not only would I not have to write repetitive code, I would also be able to have variable sized boards for free! The basic structure of the program constructs all the constrains for rows columns and blocks by appending operations and variables to an expression object.

## Result

This is obviously not a good way to solve Sudoku on a classical computer, however, this method can be transcribed into a QUBO and solved by not only a general purpose quantum computer, but also a sub class of quantum computers which operate on the principal of quantum annealing. For a classical computer, this method works well for $4 \times 4$ Sudoku boards, however, for anything bigger it usually fails to converge.

*Note: if translated into a QUBO or similar form, the computer will have an innate quant and no extra constraints are needed to keep the solutions integers*