Title: Math 110B Project 1
Date: 08/30/2021
Group members: Daoyu Li, Jilong Lyu, Xueyi Wan, Yue Liu

Our method: Deterministic Optimization formulation[1]

The problem itself perfectly fits the framework of linear programming, but with integer constraint.

Let *xijk* indicate the event that the (*i*,*j*) element of the Sudoku grid contains *k*. If it is true, *xijk*=1, otherwise *xijk*=0. Then the constraints are:

- Column, $\sum_{i=1}^{9} x_{ijk} = 1$ for $1 \le j, k \le 9$
- Row, $\sum_{j=1}^{9} x_{ijk} = 1$ for $1 \le i, k \le 9$
- Box, $\sum_{j=3p-2}^{3p} \sum_{i=3q-2}^{3q} x_{ijk} = 1$ for $1 \le k \le 9$ and $1 \le p, q \le 3$.
- Grid, $\sum_{k=1}^{9} x_{ijk} = 1$ for $1 \le i, j \le 9$.
- Clues, should be given from the problem.

The integer constraint is *xijk*∈{0,1}.

The above integer programming does not require us to solve any objective function, if the problem permits a unique solution, then the above constraints already are sufficient. The model itself generates 9^3=729 variables and there are 4×81 plus **number-of-clues** constraints. However, integer optimization is usually NP-hard, we sometimes can relax the problem a little bit to allow floating point variables instead of integers.

The problem then can be replaced by

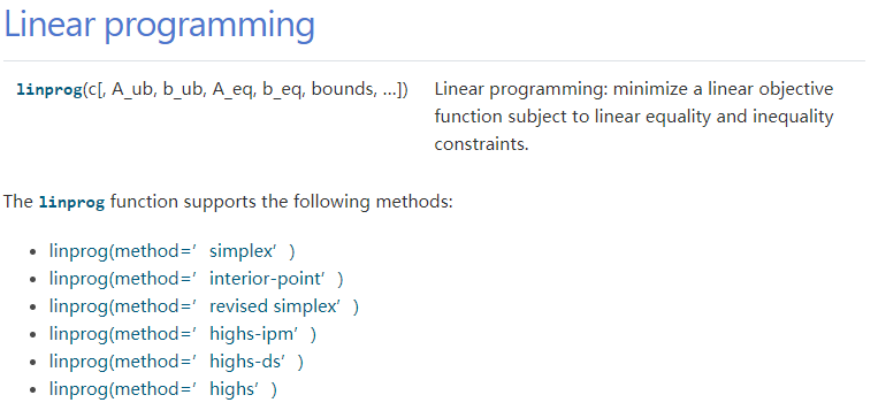
$$\min f(X)$$

subject to the linear constraints $AX=B$ for some chosen f. A and B can be computed from the previous linear constraints.

Why is Deterministic Optimization formulation more efficient?

The method we used in "**linprog**" function is "**linprog(method=' interior-point)**". (We don't label it in the code because it is the default setting.) We tried all the methods in "linprog" function and find out that "**interior-point**" has faster speed and higher accuracy than other methods in "**linprog**" function. Furthermore, we think our method is better than the genetic algorithm because the genetic algorithm can easily get local optimization (easily to find local minimum) and it is hard to find the selection crossover, but Deterministic Optimization formulation can easily get global optimization, which is more suitable to solve Sudoku problem. Moreover, the times of generation for the genetic algorithm are not stable and it easily costs a longer time on running the code, which means the genetic algorithm has a slower speed than our method.

- **Linear programming**



The success rate on data sets A and B

Small1:

```
Aver Time:   0.16 secs. Success rate: 5 / 5
Aver Time:   0.16 secs. Success rate: 10 / 10
Aver Time:   0.16 secs. Success rate: 15 / 15
Aver Time:   0.16 secs. Success rate: 20 / 20
Aver Time:   0.17 secs. Success rate: 24 / 24
```

Small2:

```
Aver Time:   0.17 secs. Success rate: 325 / 1010
Aver Time:   0.17 secs. Success rate: 325 / 1011
```

Large1: seed(42)

```
Aver Time:    0.16 secs. Success rate: 864 / 1000
```

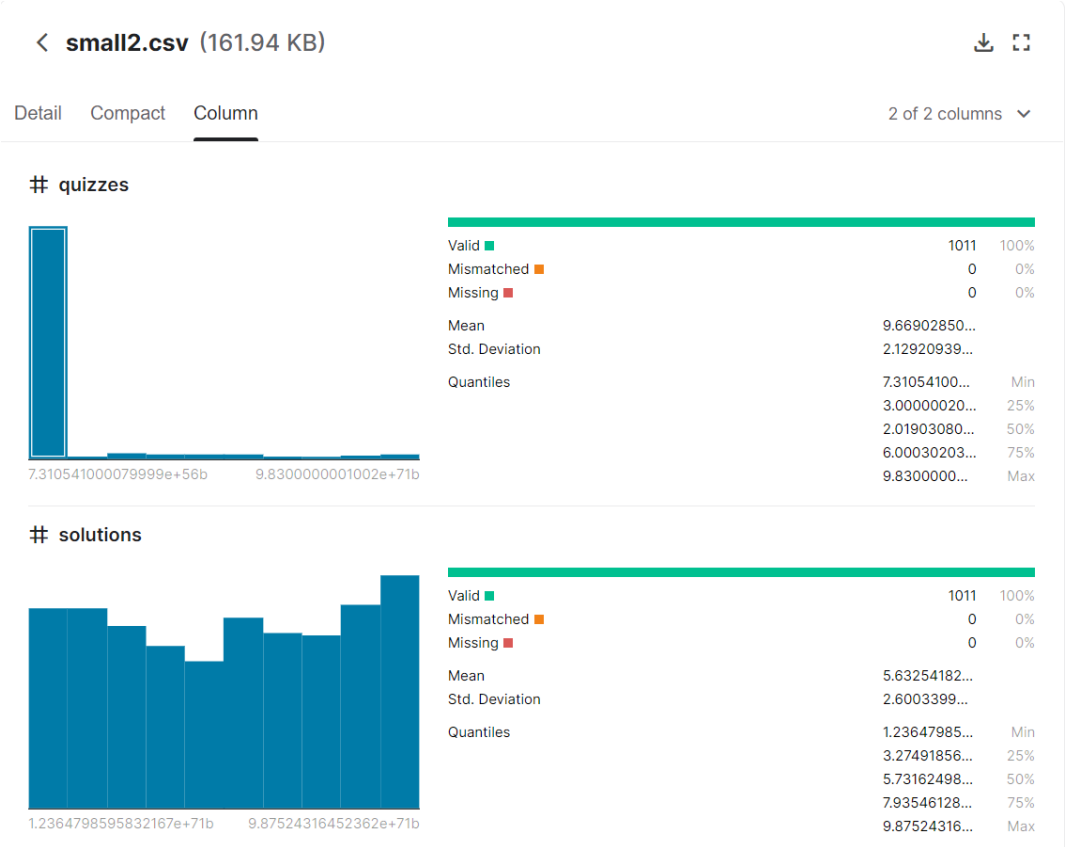
Large2: seed(42)

```
Aver Time:    0.16 secs. Success rate: 1000 / 1000
```

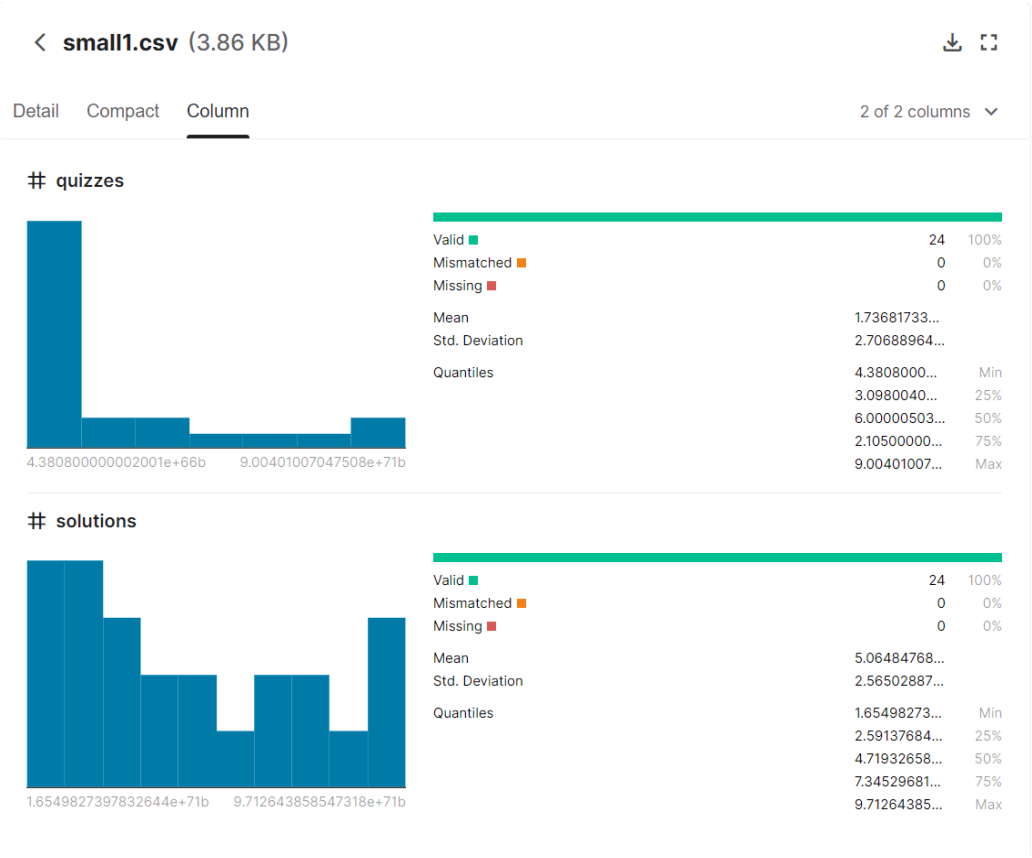
Why didn't the result match our group's expectations?

The result of the data set "small2" has low accuracy, which doesn't match our expectations. Because in the data set "small2", the probability of numbers 1-9 showing up in positions 1 to 81 is unbalanced and the distribution of the data set "small2" is more extreme than other data sets.
From the four pictures of the four data sets below, we can see that:
From the plot "quizzes" in the data set "small2", we can see that most of the data concentrate on the left side and only a little data is distributed on the right side. And the distributed pattern of the data set "small2" is the most extreme one among those four data sets.

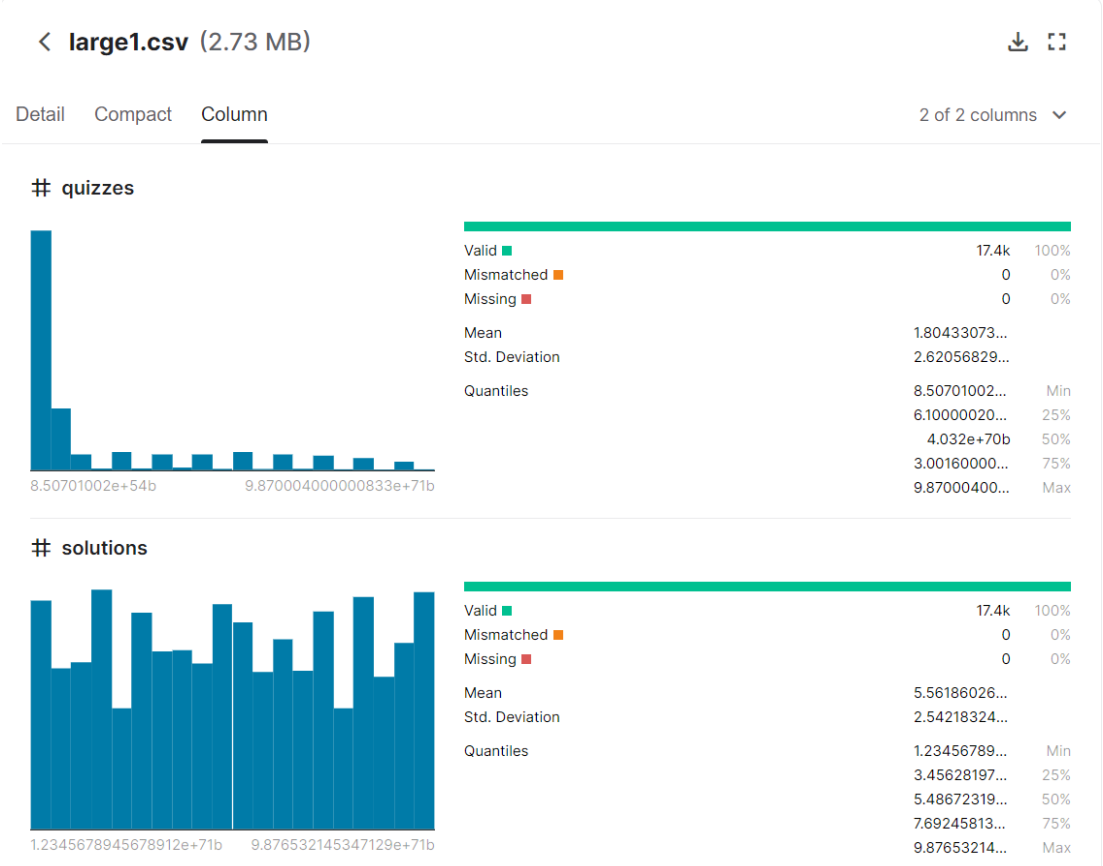
- **small2**



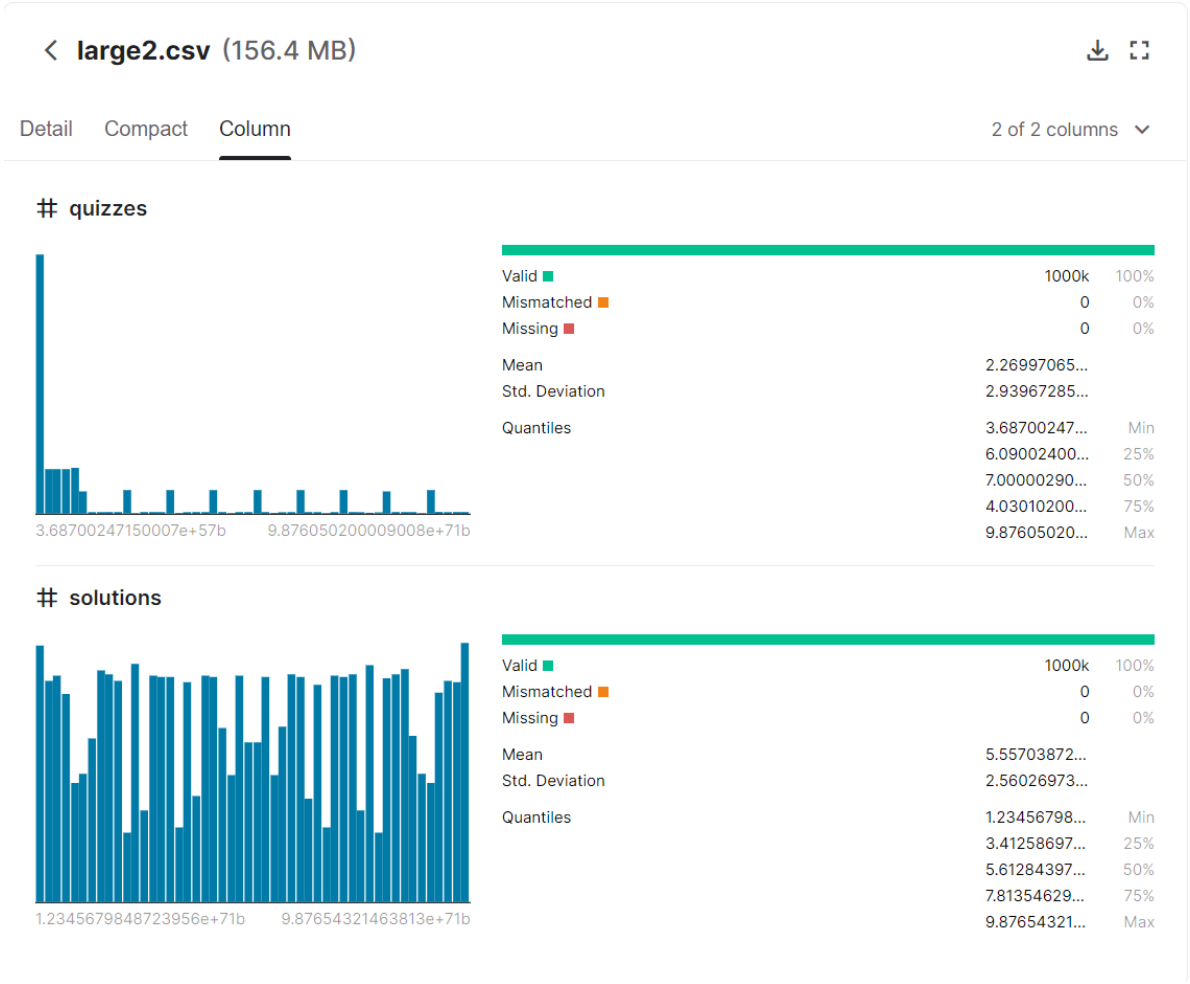
- **small1**



- **large1**



- **large2**



References

- [1]https://colab.research.google.com/drive/13J-cnjGuCljT22XW9_SbEKMYFL52mKOo?usp=sharing#scrollTo=BJtPUze84aPl
- [2]https://docs.scipy.org/doc/scipy/reference/reference/optimize.html#linear-programming