工程问题建模与仿真——九阶数独问题

519021910418 王山木

一、 九阶数独问题的求解结果：

两个九阶数独问题的求解结果如图 1，图 2 所示：

5	6	7	1	4	8	2	9	3
8	4	9	7	3	2	6	1	5
2	3	1	6	9	5	4	8	7
3	1	5	4	2	9	7	6	8
7	9	4	3	8	6	5	2	1
6	2	8	5	7	1	3	4	9
4	5	2	9	1	7	8	3	6
1	8	6	2	5	3	9	7	4
9	7	3	8	6	4	1	5	2

图 1 九阶数独 1 的求解结果

6	4	7	9	2	8	5	1	3
2	8	1	3	5	7	4	9	6
9	3	5	6	1	4	8	2	7
3	1	4	7	9	2	6	8	5
7	2	8	1	6	5	3	4	9
5	6	9	4	8	3	2	7	1
4	5	6	2	7	1	9	3	8
1	9	3	8	4	6	7	5	2
8	7	2	5	3	9	1	6	4

图 2 九阶数独 2 的求解结果

二、Matlab 运行结果：

九阶数独 1 的运行如图三所示：

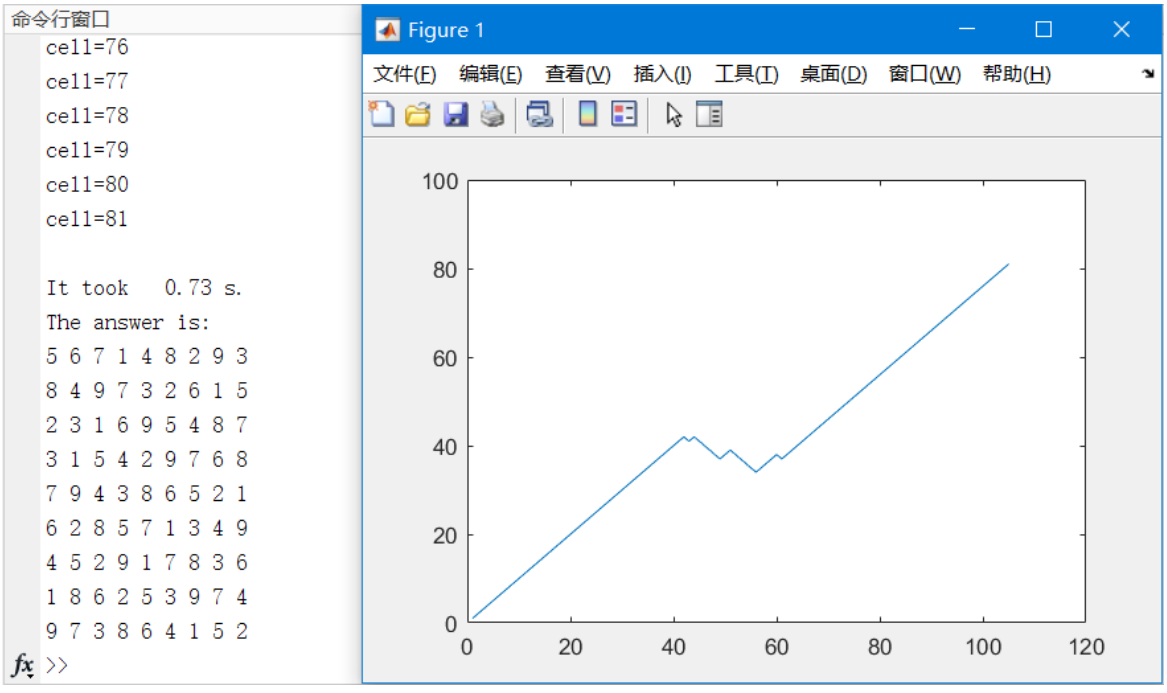

图 3 九阶数独 1 的 Matlab 运行结果

九阶数独 2 的运行结果如图 4 所示：

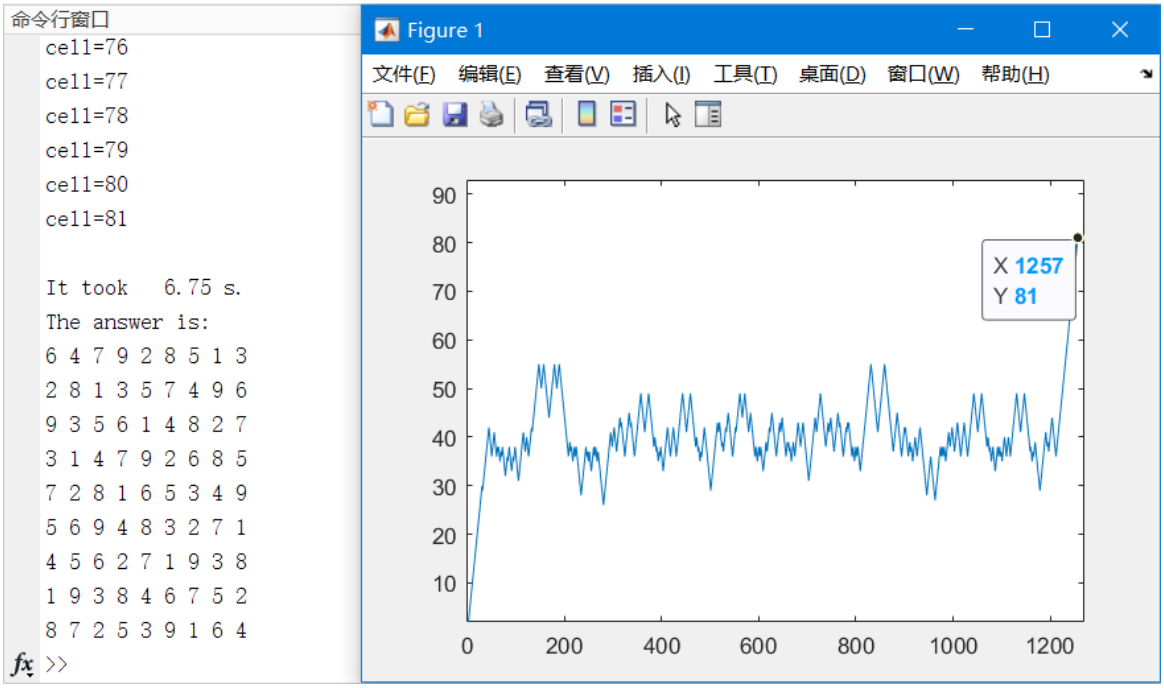

图 4 九阶数独 2 的 Matlab 运行结果

三、 优化思路：

1. 优化方向一：

首先需要明确的是，对于暴力遍历算法的优化应当着眼在减少分支数目，原始资料提供的代码的遍历顺序是“从左到右，从上到下”遍历每一个格子，但这样子的做法带来的后果就是分支数目的不确定性大大提高。例如，如果第一个格子共有6种可能填入的数字，那么总的分支数最坏的情况下就会达到6。但如果第一个格子的待选数字只有2种，那么分叉数最坏也不过两种。显然，先处理待选数字少的格子、后处理代选数字多的格子可以减少我们的尝试次数。

因此，我写了一个sorting.m文件，这个文件的主要作用是分别计算81（Order*Order）个格子的待选数字的个数，再进一步进行排序，并得到排序后每个格子的坐标。具体实现见另一文件中sorting.m文件。

2. 优化方向二：

数独游戏的一个解决策略是为宫内排除法，即对于一个格子，选择一个尚未被填入的数字，观察这一宫所在的行和列中是否出现了这个数字，如果出现了，则该行/列不能填入这个数字，最终若某一宫内只剩下一个位置可以填入该数字，则这个格子必然填入该数字。

因此，我们在填入了起始数字和每一次填入一个数字后，可以遍历当前状态，看是否存在某一个格子只能填入某一个数字，如果存在，则填入该数字。这样做可以有效减少后续的尝试次数。具体实现见另一文件中 init_dita.m 以及 oneround.m 中的处理（由于位置不同，两部分代码略有不同，具体见注释）。