分布式仿真平台 2.0 说明文档

清华大学自动华系 王奕凡 严虎

2020 年 1 月

一、 硬件环境

Windows 操作系统/Linux 操作系统/mac OSX 操作系统计算机一台

二、 软件环境

Windows 操作系统下运行 DSP.exe 必备的软件环境：

- Python3
- MySql
- pysnooperDB

所有操作系统运行源码 DSP.py 必备的软件环境：

- Python3
- MySql
- pysnooperDB
- requests
- PyQt5

- **Python3**

由于 Python 官方宣布自 2020.1.1 后不再对 Python 2 进行任何更新，各种第三方库也会停止对 Python 2 的维护，综合考虑后将分布式仿真平台迁移到 Python 3 上。

本仿真平台已在 python3.7.6 32 位，python3.8.1 64 位下测试通过，并且没有使用高版本 Python 3 的库函数，因此如电脑中已安装 Python 3 不必重新安装新版本。如有安装需求，到 https://www.python.org/downloads/下载适宜电脑的版本即可。

安装时需要将 python 添加到系统变量。

由于程序中调用 python 程序使用的是“python xxx.py”的命令，如电脑中有其他版本的 python，需要保证输入 python 后默认为目标版本。如下图所示

```
C:\Users\26245>python
Python 3.7.6 (tags/v3.7.6:43364a7ae0, Dec 18 2019, 23:46:00) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
```

说明默认 python 版本为 python3.7.6 32 位。

具体做法可参照【如何让 Python2 与 Python3 共存】https://www.cnblogs.com/davieyang/p/10108021.html

- **MySql**

本仿真平台相较于之前的版本最大的不同是有“调试功能”选项，这一功能建立在 MySql 的数据库环境下，因此需要电脑安装 MySql 关系型数据库管理系统。

安装教程可参考 https://www.runoob.com/mysql/mysql-install.html，windows 10 操作系统的详细安装说明: https://www.runoob.com/w3cnote/windows10-mysql-installer.html.

按说明安装后**需建立调试程序所用的数据库**，登录 MySQL 服务后后输入 SQL 语言"create DATABASE XXXXX"创建数据库，这里 XXXXX 指的是待创建数据库名。详细的 MySql 教程可参考 https://www.runoob.com/mysql/mysql-tutorial.html。

目前最新版本为 8.0.18，仿真平台在此版本下开发完成。

- **pysnooperDB**

pysnooperDB 为参照 python 调试工具 PySnooper 库修改设计的第三方库，使用 pip 命令安装即可。

pip install pysnooperDB

目前最新版本为 1.0.1，历史版本有些问题，1.0.1 及以后是没问题的。

- **requests**

requests 是 python 3 中用于 HTTP 连接保持和连接池，使用 cookie 保持会话的第三方库，使用 pip 命令安装即可。

pip install requests

- **PyQt5**

PyQt5 是用来创建 PythonGUI 应用程序的工具包，使用 pip 命令安装即可。

pip install PyQt5

三、 使用流程

程序的基本使用：

1. 解压 DSP2.0.zip
2. 配置好相应的软件环境
3. 双击打开 DSP.exe（Windows），也可由 IDE 运行 DSP.py
4. 填入需要的节点数量
5. 根据所示的拓扑模板导入正确 JSON 格式的拓扑，每个节点对象需拥有 ID，IP，PORT(端口)，adjID(邻接节点编号)，adjDirection(邻居节点方向)，datalist(外部导入数据列表)。(*与之前的差异在于现在加入了邻居节点方向，在程序编写时仅能获取邻居节点方向信息，无法直接获取邻居节点 ID)
6. 根据所示的算法模板上传正确的算法 python 文件，可使用的函数有 self.transmitdata(data)，self.sendUDP(str)，

elf.sendDataToDirection(direction,data)，self.syncNode()。

同步通信函数为 self.transmitData(data)，data 为需通信数据，data 为需通信数据，可为字符串、对象、数组等 JSON 格式能解析的数据类型，返回数据 feedback = self.transmitData(data)为二元数组，第一分量表示返回邻居方向 adjDirection，第二分量表示相应数据，与第一分量中方向一一对应。

异步通信函数为 self.sendDataToDirection(direction,data)，direction 为需要传递数据的方向，data 为需通信数据，可为字符串，对象，数组等 JSON 格式能解析的数据类型，调用该函数时可通过 self.adjData 获取邻居传递过来的数据信息，内容与同步通信函数返回值第二分量一致。

异步通信函数请勿与同步通信函数同时使用，使用异步通信函数时需要考虑到程序执行的异步性与延时，慎用！

调用异步函数时可以使用 self.syncNode()函数使得程序实现同步。

在算法设计中，可在运行状态区域打印出自己调试所需要的算法运行信息，函数为 self.sendUDP(str)，str 为字符串格式的数据。

(*与之前的差异为添加了异步通信函数与同步函数，且现在同步通信函数的返回值为邻居方向而非邻居 ID)

7. 上传前两者完毕后，点击运行按钮，即会开始执行程序，程序运行中的问题分布式平台将相应提示。

*调试模式的使用：

1. 算法程序的脚本文件需保证能够在 IDE 中编译通过不报错，否则仿真平台无法运行。调试模式只是从功能层面对算法进行调试，而非进行语法上的调试。
2. 同“程序的基本使用”的 1~6 步，输入节点数量，导入拓扑和算法程序。
3. 打开“调试模式开关”
4. 输入数据库用户名、数据库密码、数据库名（事先建立好用于调试的数据库）、数据表前缀名（后会加上节点 id 存储在数据库的不同表中），点击“显示所有数据表”检查输入信息是否有误，如果“调试信息显示”中出现输入数据库中的数据表名，那么所填信息无误。
5. 输入查看变量名，即算法程序中关心变量名（类似于 print 函数发挥的作用）

   需要用英文逗号分隔查看变量名，如

   m,data,adjData

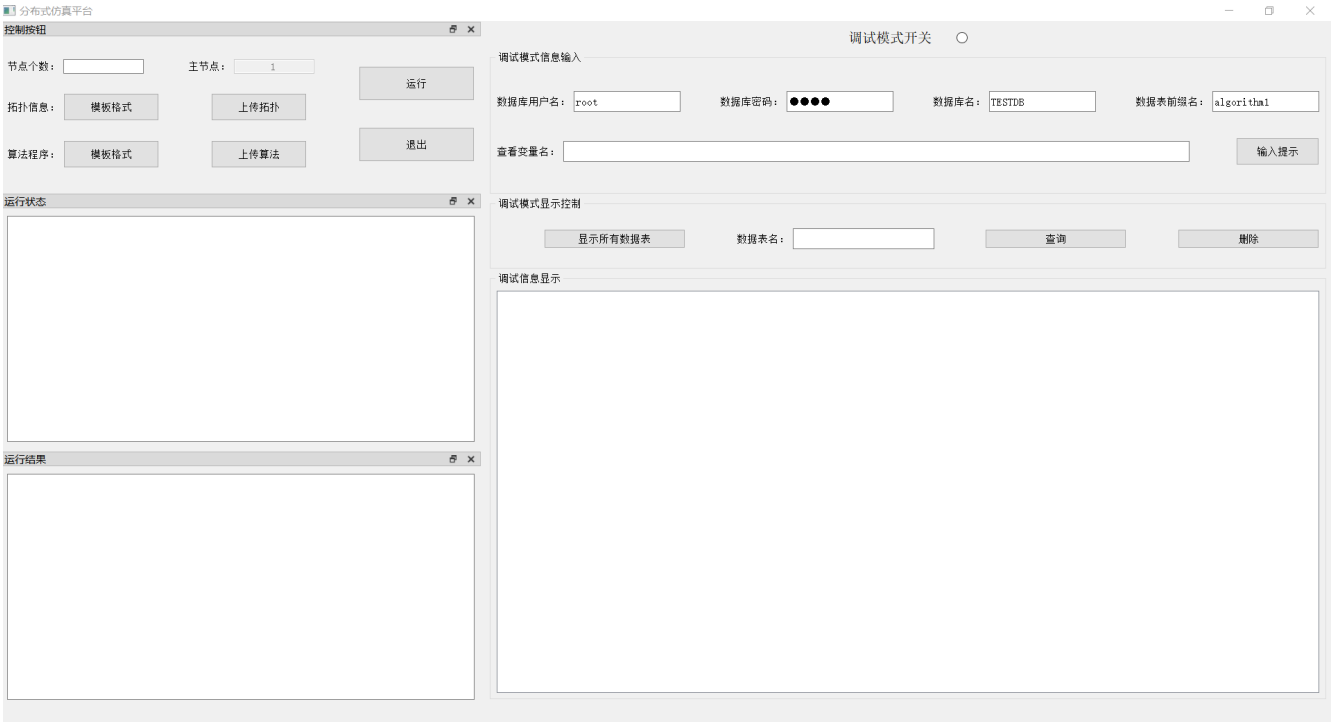
   就是合法的输入，如果查看变量名为空，则默认记录所有的变量变化情况。

   程序会记录待查看变量的所有变化情况，并将其上传至数据库。由于 Window 环境下 MySql 默认不区分大小写，建议避免出现只以大小写区分的变量，例如 Y 和 y。
6. 点击运行按钮，即会开始执行程序。由于需要把变量变化情况上传至数据库，程序运行速度会有所降低。
7. 程序运行结束后，点击“显示所有数据表”，将程序生成的数据表名输入“数据表名”中，点击查询获取数据表信息，点击删除删除该数据表。可依据变量的变化情况进行相应的调试。
8. 同时本地 log 文件夹下会生成各节点相应的 log 日志文件，有需要的可进行查看。

四、 使用流程演示

程序的基本使用

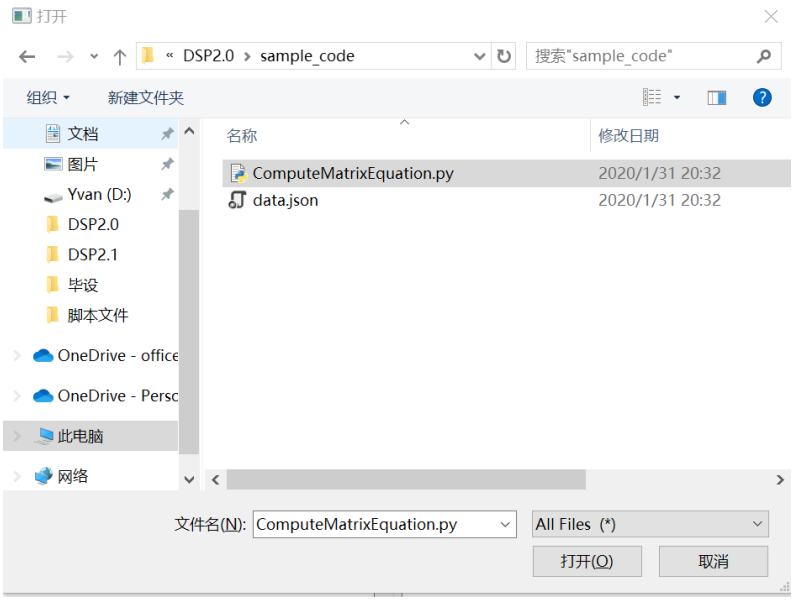
打开 DSP.exe（或运行 DSP.py）



这里控制按钮、运行状态、运行结果三个子窗口可根据用户习惯调整大小和位置

上传拓扑。我们提供了一个例程的拓扑和算法在文件夹 sample_code 中，这里我们将 data.json 的拓扑上传。

上传算法文件。上传 ComputeMatrixEquation.py 文件。



点击运行。等待返回结果：

该例程用 Jacobi 迭代法求解方程组

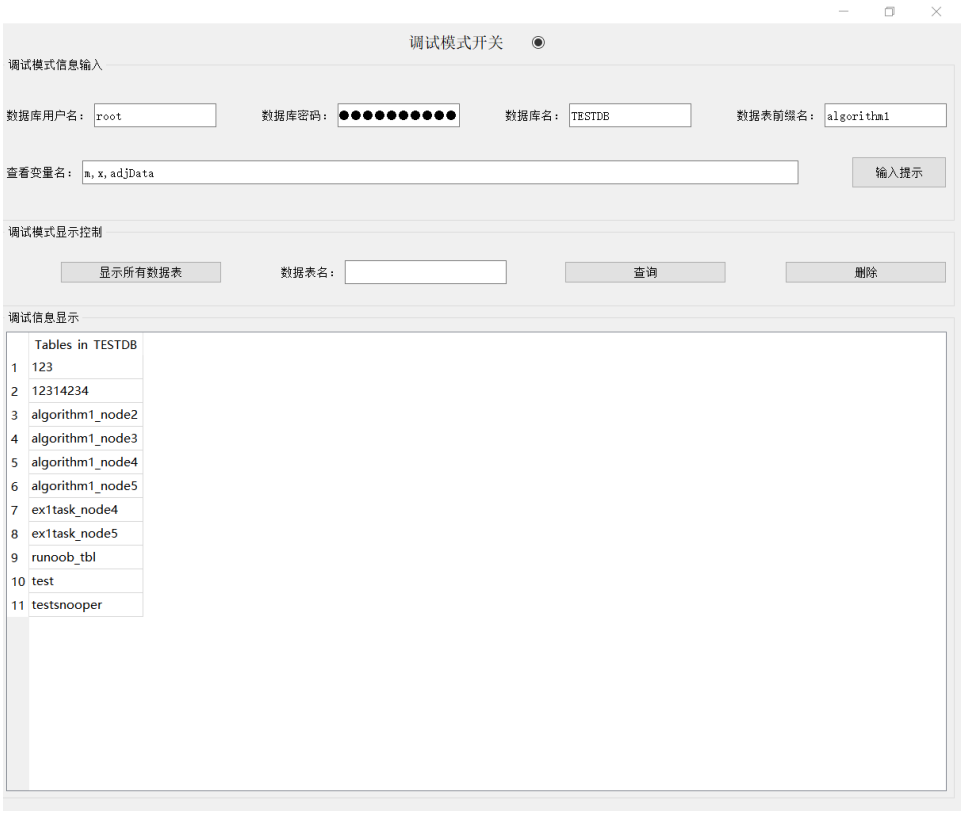
$$\begin{cases} 8x_1 - 3x_2 + 2x_3 = 20 \\ 4x_1 + 11x_2 - x_3 = 33 \\ 6x_1 + 3x_2 + 12x_3 = 36 \end{cases}$$

计算的最终结果为$x = (3.00032, 1.98399, 1.00097)^T$，与精确解$x^* = (3,2,1)^T$十分接近。

调试模式的使用：

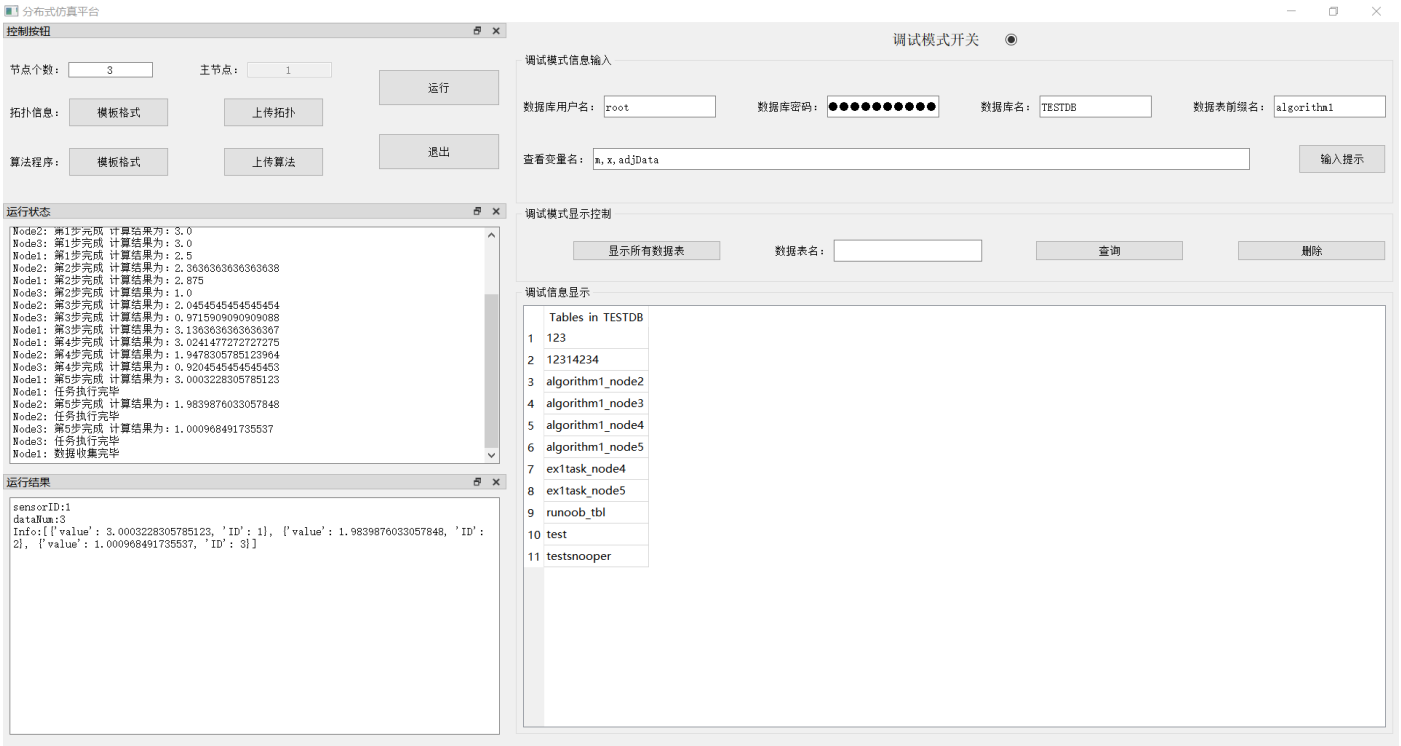
同上所示，输入相同的拓扑和算法程序。

打开“调试模式开关”， 输入数据库用户名、数据库密码、数据库名、数据表前缀名，点击“显示所有数据表”检查输入信息是否有误



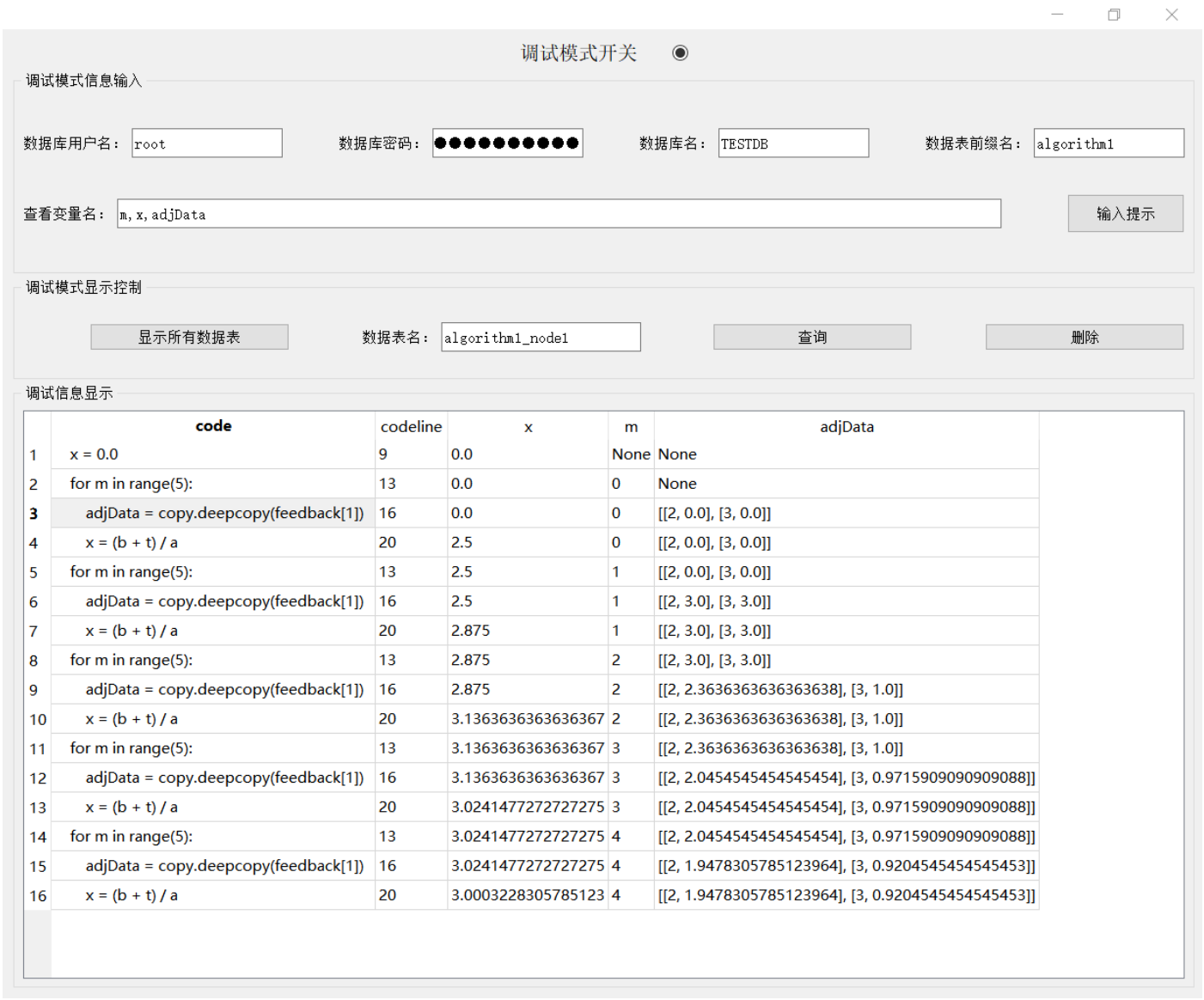
“调试信息显示”正确显示数据库中的所有数据表名，所填信息无误。

输入查看变量名 m,x,adjData，点击运行



程序左侧运行状态和运行结果与之前相同

再次点击“显示所有数据表”，将程序生成的数据表名输入“数据表名”中，点击查询

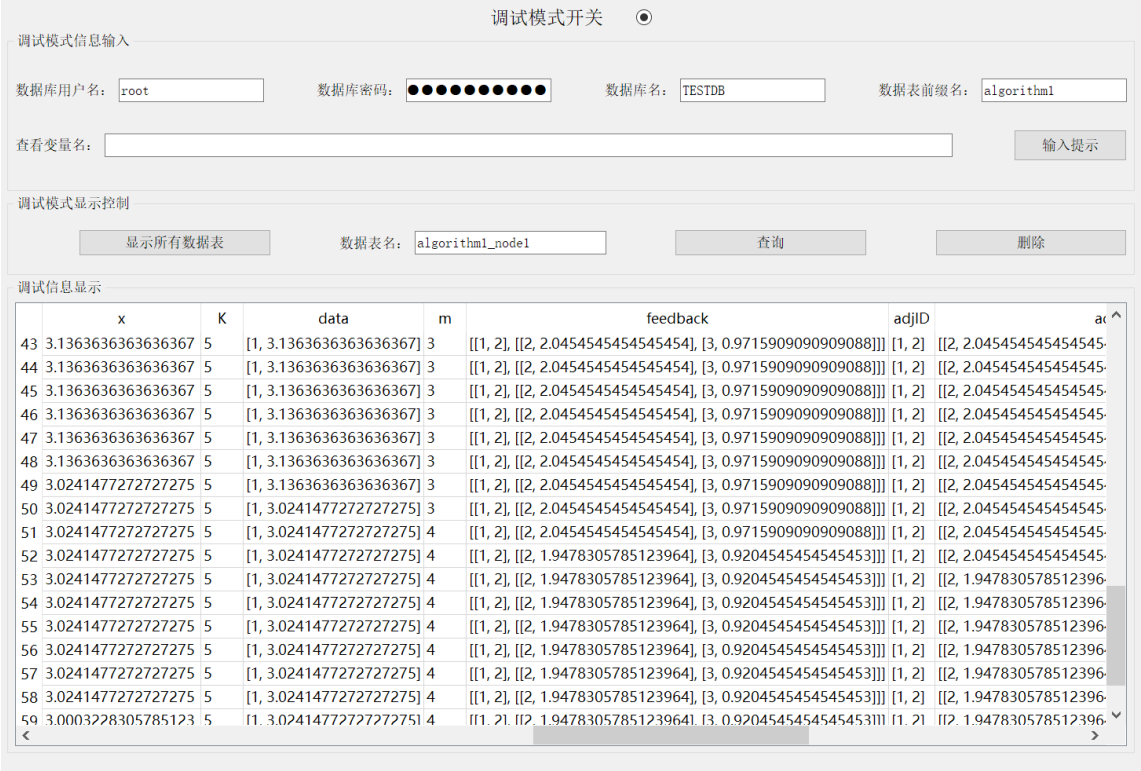


	code	codeline	x	m	adjData
1	x = 0.0	9	0.0	None	None
2	for m in range(5):	13	0.0	0	None
3	adjData = copy.deepcopy(feedback[1])	16	0.0	0	[[2, 0.0], [3, 0.0]]
4	x = (b + t) / a	20	2.5	0	[[2, 0.0], [3, 0.0]]
5	for m in range(5):	13	2.5	1	[[2, 0.0], [3, 0.0]]
6	adjData = copy.deepcopy(feedback[1])	16	2.5	1	[[2, 3.0], [3, 3.0]]
7	x = (b + t) / a	20	2.875	1	[[2, 3.0], [3, 3.0]]
8	for m in range(5):	13	2.875	2	[[2, 3.0], [3, 3.0]]
9	adjData = copy.deepcopy(feedback[1])	16	2.875	2	[[2, 2.3636363636363638], [3, 1.0]]
10	x = (b + t) / a	20	3.1363636363636367	2	[[2, 2.3636363636363638], [3, 1.0]]
11	for m in range(5):	13	3.1363636363636367	3	[[2, 2.3636363636363638], [3, 1.0]]
12	adjData = copy.deepcopy(feedback[1])	16	3.1363636363636367	3	[[2, 2.0454545454545454], [3, 0.9715090909090088]]
13	x = (b + t) / a	20	3.0241477272727275	3	[[2, 2.0454545454545454], [3, 0.9715090909090088]]
14	for m in range(5):	13	3.0241477272727275	4	[[2, 2.0454545454545454], [3, 0.9715090909090088]]
15	adjData = copy.deepcopy(feedback[1])	16	3.0241477272727275	4	[[2, 1.9478305785123964], [3, 0.9204545454545453]]
16	x = (b + t) / a	20	3.0003228305785123	4	[[2, 1.9478305785123964], [3, 0.9204545454545453]]

这里输出了 id 为 1 的节点“查看变量”每一次变化的变化情况，同时还输出了引起本次变化的代码和其行数。例如第 10 行的数据显示，由算法程序第 20 行的代码“ x = (b + t) / a”引起了查看变量 x 的变化，x 从 2.875 变为 3.1363636363636367。

通过这些数据，开发人员可查看算法程序中关心的变量的变化情况，从而完成相应的调试工作。比较经典的可以选择迭代次数、本地数据、邻居数据等重要数据。

需要说明的是任意查看变量的变化都会增加一行数据，所以如果“查看变量名”缺省输入，记录所有变量的话通常不利于调试，效果如下：



可以看到这里数据非常多，反而不方便调试，所以通常需要用户根据关心的变量设定查看变量名。当然用户也可根据需求，将所有变量输出后编写第三方数据库处理函数与进行深入层次的查看。