



本源量子 QLAS 库 V1.1.1

说明书

本源量子计算科技（合肥）股份有限公司

目录

1. 引言	1
1.1 目的	1
1.2 背景技术介绍	1
2. 软件介绍	1
2.1 算法概述	2
2.2 整体结构	2
2.3 算法流程	3
2.4 运行环境	4
3. 使用方法	4
3.1 安装（使用源码构建）	4
3.1.1 Linux 和 MacOS	4
3.1.2 Windows	6
3.2 函数接口	7
3.3 配置参数文件	11

1. 引言

1.1 目的

开发一种通用、容错、稳定的基于量子虚拟机或真实量子计算机的线性求解器函数库，易于调用，并实现端到端的线性方程组的高精度求解。

1.2 背景技术介绍

主要功能和特点：

量子线性代数求解器（Quantum Linear Algebra Solver）是一款应用于线性方程组求解的 C++ 函数库，提供了使用量子算法求解线性方程组的接口。

QLAS 库内置多种量子线性求解算法，可以使用量子计算框架 QPanda2，量子稀疏态 QRAM 模拟器 进行量子计算模拟，亦可以接入本源量子云 QCloud 使用量子真实芯片进行求解。

为了实现端到端的量子计算，库中加入了层析算法，用于解决真机、虚拟机的量子信息到经典信息的提取。

当前计划中，QLAS 库 V1.1.1 版本提供的具体量子线性求解器如表 1 所示。

表 1 QLAS 库 V1.1.1 版本内置 QLS 明细表

QLS 名称	说明	底层依赖
VIRTUAL_HHL	HHL 算法模拟	QPanda
VIRTUAL_VQLS	VQLS 算法模拟	Eigen
VIRTUAL_QVQLS	VQLS 算法模拟	QPanda
VIRTUAL_CKS	CKS 算法模拟	稀疏态、QRAM 模拟器
NISQ_VQLS	VQLS 真机求解	QPanda、CURL

2. 软件介绍

2.1 算法概述

根据输入的经典线性方程组系数矩阵 A 和目标向量 b ，根据外部的配置参数 `config`，结合经典的迭代算法，或子空间算法，应用不同的量子线性算法，实现量子-线性求解。

2.2 整体结构

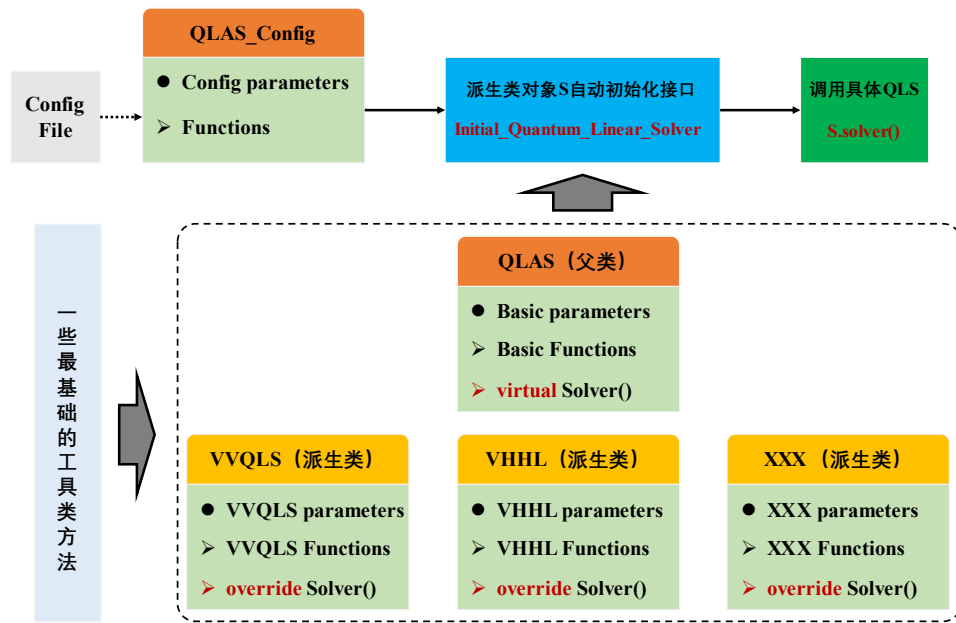


图 1 算法整体框架

算法整体框架如图所示，所有类型和方法都在命名空间 `ORIGINQ` 中。具体地，包含如下关键模块：

- `QLAS_Config` 类：包含所有控制参数和相关方法，初始化该类对象会按指定路径读取配置文件 `config.cfg`；
- 基础工具方法：在代码 `tools.h` 和 `tools.cpp` 中，包含了一些 QLS 会用到，但和求解线性方程组关系不大的方法；这些方法也可以写成 `QLAS` 主类的成员方法，但为了 `QLAS` 类的简洁，这里选择将其作为基础工具方法；
- `QLAS` 主类：包含线性方程组的相关变量和例如迭代求解、子空间方法、预处理方法等类成员方法，是构建具体 QLS 派生类的基础；
- QLS 派生类：是某 QLS 的具体实现，包含具体 QLS 的相关变量和方法，通过重写 `solver()` 接口实现多态特性；

- 派生类对象自动初始化接口：用户直接调用该接口，生成与指定 QLS 对应的派生类对象，进一步调用该对象的 `solver()` 实现指定 QLS 方法。

2.3 算法流程

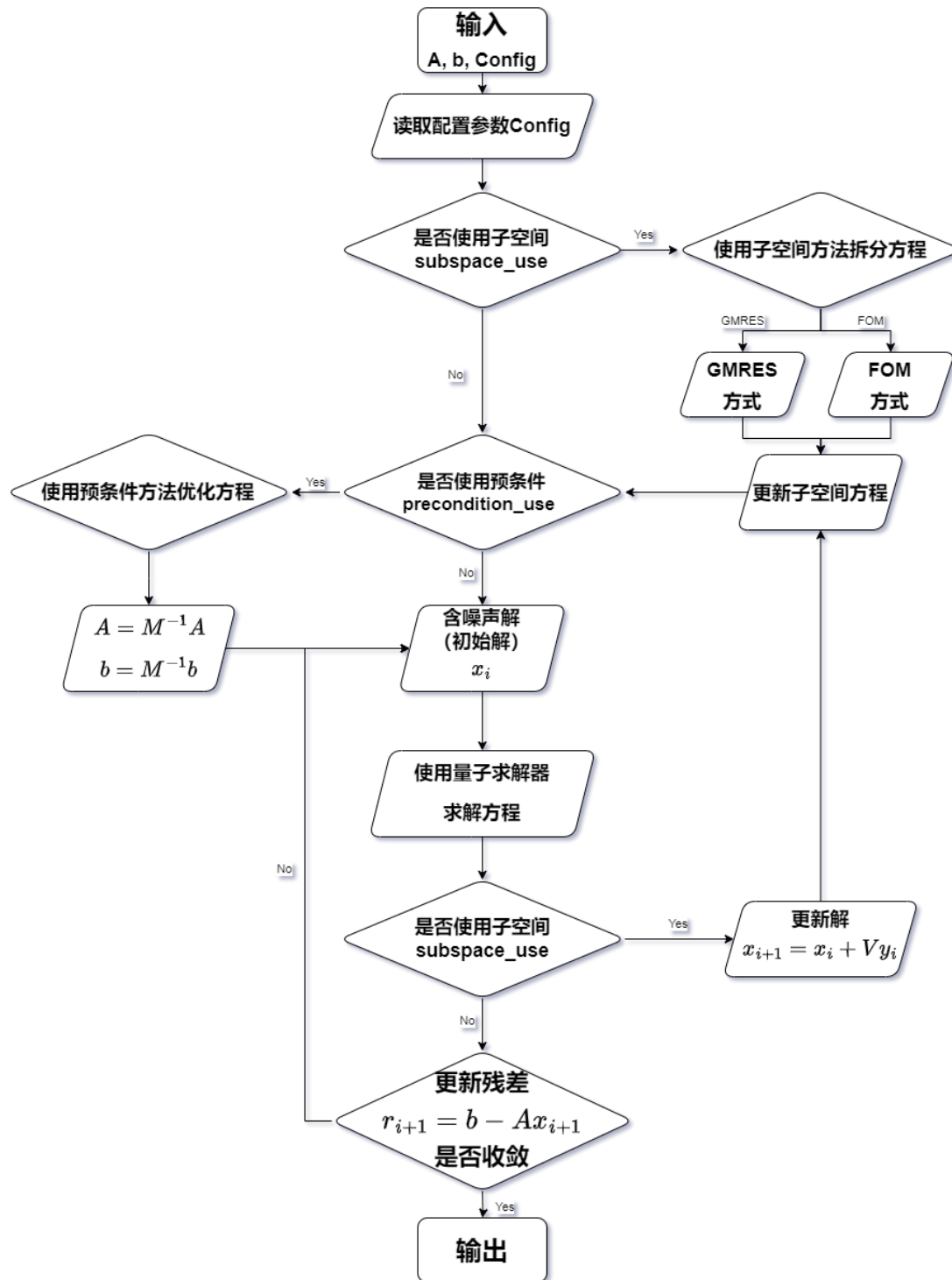


图 2 算法整体流程

算法整体框架如图 2 所示。

2.4 运行环境

硬件环境：

CPU Intel(R) Core(TM) i5-10400 CPU @2.90GHz

内存 16GB 2666MGz

软件环境：

Windows 10 专业版 22H2 19045.4529

Visual Studio Community 2022(17.8.6)

开发依赖环境：

QPanda2(2.8.10)

CURL

QHeTu

稀疏态 QRAM 模拟器（可选）

3. 使用方法

3.1 安装（使用源码构建）

以下是在通用平台下构建、编译安装 QLAS 函数库的快速入门：

3.1.1 Linux 和 MacOS

环境要求：

大多数 Linux 发行版都安装了所有必需的工具。但有时会出现例外情况，所以可以检查是否具有以下环境：

- gcc/g++/gfortran 编译器
- CMake 3.0 以上
- pkg-config 工具
- QPanda2 (HHL、VQLS 真机依赖环境)
- CURL (量子云连接功能，VQLS 真机依赖环境)
- QHeTu (量子云连接功能，VQLS 真机依赖环境)

以及一些可选项:

- 稀疏态 QRAM 模拟器 (QDALS、CKS 算法模拟器, 可选)
- MPI (在集群上运行量子虚拟机, 可选)
- CUDA 9 以上 (在 GPU 上运行量子虚拟机, 可选)

编译:

在完成 Git 获取源码后, 进入 QLAS 库跟目录, 输入以下命令:

```
mkdir -p build
cd build
cmake -DUSE_QPANDA2=ON ..
make
```

如果有 QRAM 需求, 可以添加-DUSE_QRAM=ON

如果有需求, 用户通过命令 -DCMAKE_INSTALL_PREFIX 修改 QLAS 的安装路径, 配置方法如下所示:

```
mkdir -p build
cd build
cmake -DCMAKE_INSTALL_PREFIX=/usr/local -DUSE_QPANDA2=ON ..
make
```

也可以根据需求将可配置项设置为 OFF

安装:

完成编译后, 可以通过命令完成安装

```
make install
```

备注:

- 在不通过 -DCMAKE_INSTALL_PREFIX 制定安装路径的情况下, 安装

路径默认在 /usr/local

- 在启用 -DUSE_QPANDA2 及 -DUSE_QRAM 选项时，也要使用 -DQPANDA_INSTALL_DIR= 及 -DQRAM_INSTALL_DIR= 指定其安装路径
- MacOS 可参考 Linux 系统下进行部署

3.1.2 Windows

在 Windows 平台下，QLAS 函数库推荐使用 Visual Studio 编译、安装。

Visual Studio:

在 Visual Studio 组件中安装 CMake 组件，用 VS 打开安装 QLAS 函数库源码的文件夹，选择 x64-Release 的模式下编译。

右键，用 Visual studio 打开，在 “选项-项目-QLAS 的 CMake 设置 ” 中选定 CMAKE_INSTALL_PREFIX 设置安装目录及编译选项。

随后点击 “选项-生成-全部生成 ” 后完成编译，并点击 “选项-生成-安装 QLAS ” 完成安装。

Mingw32:

环境要求:

- Mingw32 8.0.0
- CMake 3.0 以上
- QPanda2 (HHL、VQLS 真机依赖环境)
- CURL (量子云连接功能，VQLS 真机依赖环境)
- QHeTu (量子云连接功能，VQLS 真机依赖环境)

以及一些可选项:

- 稀疏态 QRAM 模拟器 (QDALS、CKS 算法模拟器，可选)
- MPI (在集群上运行量子虚拟机，可选)
- CUDA 9 以上 (在 GPU 上运行量子虚拟机，可选)

总体和 Linux 系统下相同，但所有 make 命令需要替换为 mingw32-make。

测试:

在编译完毕后的 `build\bin` 文件夹（Windows 下应该是 `build\x64-Release\bin`），应当有相关的测试程序的可执行文件，直接运行测试。

```
./QLAS_test
```

Visual Studio 下可以直接通过选择启动项，启动相应的可执行测试程序。

3.2 函数接口

从用户角度而言，使用 QLAS 库包含以下步骤：

- 写好 `config.cfg` 配置文件；
- 初始化 `QLAS_Config` 类型对象；
- 调用自动初始化接口；
- 调用 `solver()`接口。

1、调用头文件

```
#include "QLAS.h"
```

2、创建 STL 或 Eigen 库的矩阵 A、b（QLASV1.1.1 支持多种输入格式）；

```
Eigen::SparseMatrix<double, 1, int> A;
```

```
Eigen::VectorXd b;
```

或

```
std::vector<std::vector<double>>> A;
```

```
std::vector<double> b;
```

或

```
Eigen::MatrixXd A;
```

```
Eigen::VectorXd b;
```

3、创建 Config 对象

```
QLAS_Config config;
```

4、使用初始化构建 solver 求解器对象

```
auto solver = Initial_Quantum_Linear_Solver(A, b, config);
```

5、调用求解器求解模块得到解

```
Eigen::vectorXd x = solver -> solver();
```

6、编译链接：

假设文件构成如下：

```
project
├── CMakeLists.txt
├── include
│   └── xxxxxx
├── src
│   └── main.cpp
└── build
```

main.src 为上一节的代码，include 为预留的框架，这里可以为空。

该例子的 CMakeLists 的配置示例如下：

```
cmake_minimum_required(VERSION 3.1)
project(depQLAS test)

set(CMAKE_CXX_STANDARD 17)
if(MSVC)
    list(APPEND EXTRA_LDFLAGS
        -D_SILENCE_TR1_NAMESPACE_DEPRECATION_WARNING
        -D_CRT_SECURE_NO_WARNINGS
        /bigobj)
```

```

if (NOT USE_MSVC_RUNTIME_LIBRARY_DLL)
    foreach (flag
        CMAKE_C_FLAGS
        CMAKE_C_FLAGS_DEBUG
        CMAKE_C_FLAGS_RELEASE
        CMAKE_C_FLAGS_MINSIZEREL
        CMAKE_C_FLAGS_RELWITHDEBINFO
        CMAKE_CXX_FLAGS
        CMAKE_CXX_FLAGS_DEBUG
        CMAKE_CXX_FLAGS_RELEASE
        CMAKE_CXX_FLAGS_MINSIZEREL
        CMAKE_CXX_FLAGS_RELWITHDEBINFO)

        if (${flag} MATCHES "/MDd")
            string(REGEX REPLACE "/MDd" "/MT" ${flag}
"${${flag}}")
        endif()

        if (${flag} MATCHES "/MD")
            string(REGEX REPLACE "/MD" "/MT" ${flag}
"${${flag}}")
        endif()

        if (${flag} MATCHES "/W3")
            string(REGEX REPLACE "/W3" "/W0" ${flag}
"${${flag}}")
        endif()
    endforeach()
endif()

elseif (MINGW)
    list(APPEND EXTRA_LDFLAGS
        -w
        -DGTEST_USE_OWN_TR1_TUPLE=1
        -fpermissive)

    set(CMAKE_CXX_FLAGS_DEBUG "$ENV{CXXFLAGS} -O0 -g -Og
-ggdb -m64 -Wa, -mbig-obj")
    set(CMAKE_CXX_FLAGS_RELEASE "$ENV{CXXFLAGS} -O3 -
DNDEBUG")

elseif (UNIX)
    list(APPEND EXTRA_LDFLAGS

```

```

        -w
        -DGTEST_USE_OWN_TR1_TUPLE=1
        -fPIC
        -fpermissive)

    if (NOT APPLE)
        list(APPEND LIB_LIST
            dl
            pthread)
    endif ()

    set(CMAKE_CXX_FLAGS_DEBUG "$ENV{CXXFLAGS} -O0 -g -
ggdb")
    set(CMAKE_CXX_FLAGS_RELEASE "$ENV{CXXFLAGS} -O3 -
DNDEBUG")
    endif()

    set(LIBRARY_OUTPUT_PATH ${PROJECT_BINARY_DIR}/lib)
    set(EXECUTABLE_OUTPUT_PATH ${PROJECT_BINARY_DIR}/bin)
    set(CMAKE_RUNTIME_OUTPUT_DIRECTORY_DEBUG
${PROJECT_BINARY_DIR}/bin)
    set(CMAKE_RUNTIME_OUTPUT_DIRECTORY_RELEASE
${PROJECT_BINARY_DIR}/bin)
    set(CMAKE_ARCHIVE_OUTPUT_DIRECTORY_DEBUG
${PROJECT_BINARY_DIR}/lib)
    set(CMAKE_ARCHIVE_OUTPUT_DIRECTORY_RELEASE
${PROJECT_BINARY_DIR}/lib)

    find_package(CURL REQUIRED)

    option(FIND_OPENMP "find openMP" ON)
    if (FIND_OPENMP)
        find_package(OpenMP)
    endif(FIND_OPENMP)
    if(OPENMP_FOUND)
        option(USE_OPENMP "find OpenMP" ON)
        message("OPENMP FOUND")
        set(CMAKE_C_FLAGS "${CMAKE_C_FLAGS}
${OpenMP_C_FLAGS}")
        set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS}
${OpenMP_CXX_FLAGS}")
        set(CMAKE_EXE_LINKER_FLAGS
"${CMAKE_EXE_LINKER_FLAGS} ${OpenMP_EXE_LINKER_FLAGS}")
    else(OPENMP_FOUND)

```

```
option(USE_OPENMP "not find OpenMP" OFF)
endif(OPENMP_FOUND)

#需要用户自己配置的部分-----

set(CMAKE_PREFIX_PATH XXXXXXXXXXXXXXXXXXXXX )
find_package(QLAS REQUIRED)

include_directories(include
                        ${QLAS_INCLUDE_DIRS}
                        ${QLAS_TP_INC_DIRS})

link_directories(${QLAS_LIBRARY_DIRS})
add_executable(demo src/main.cpp)
target_link_libraries(demo ${QLAS_LIBRARIES})
```

3.3 配置参数文件

QLAS_Config 类的类成员变量说明如表 2 所示。

表 2 QLAS_Config 类成员变量明细表

QLAS_Config 类成员变量		
变量名	变量类型	详细说明
额外功能控制参数		
check_config_file	bool	= true: 输出最终读取的配置参数
write_solution_file	bool	= true: 输出求解过程和结果文件，将在 root_path 目录下创建 QLAS 文件夹存放输出文件
screen_print_monitor	bool	= true: 求解过程中开启屏幕输出
set_omp_threads	long	设置 OpenMP 线程数 合法值≥1，非法值将强制转化为 1
exception_handling	string	异常处理 默认为 NO
auto_exception_handling_interval	double	异常处理参数
restart_calculation	bool	=true: 使用续算功能
write_restart_file	bool	=true: 写续算文件
sleep_to_run	double	休眠时间 合法范围：≥0E0
子空间方法参数		

subspace_use	bool	= true: 使用子空间方法
subspace_method	string	= FOM: 使用 FOM 方法 否则使用 GMRES 方法
subspace_dimension	long	子空间维度 合法范围: ≥ 2 且为 2 的指数次 小于 2 的输入将强制转化为 2
subspace_maxstep	long	子空间方法的最大迭代次数 ≤ 0 : 无最大迭代次数限制
subspace_precision	double	子空间方法的收敛精度 合法范围: $> 0E0$ 非法输入将强制转化为 $1E-3$
预处理参数		
precondition_use	bool	= true: 使用预处理方法
preconditioner	string	= ILUT: 使用 ILUT 预条件 = INV: 使用快速求逆预条件
precond_ilut_p	long	L、U 矩阵保留的绝对值最大的非 0 元个数 合法范围: ≥ 1
precond_ilut_tau	double	舍弃阈值 合法范围: $> 0E0$
量子求解器基础参数		
quantum_method	string	声明使用的 QLS 方法, 合法输入有 = VIRTUAL_HHL = VIRTUAL_VQLS = VIRTUAL_CKS = VIRTUAL_QDALS = NISQ_VQLS = CLASS 非法值强制转为 VIRTUAL_VQLS
quantum_maxstep	long	QLS 最大迭代次数 ≤ 0 : 无最大迭代次数限制
quantum_precision	double	QLS 收敛精度 合法范围: $> 0E0$ 非法输入强制转化为 $1E-4$
VIRTUAL_HHL 参数 (quantum_method = VIRTUAL_HHL)		
hhl_precision	long	HHL 精度, 特征值可表示范围 合法范围: ≥ 0
VIRTUAL_QRAM 参数 (quantum_method = VIRTUAL_CKS&QDALS)		
qram_data_size	long	QRAM 的编码尺寸 合法范围: > 0
qram_eps	double	算法的误差参数 合法范围: $> 0E0$
VIRTUAL_VQLS 参数 (quantum_method = VIRTUAL_VQLS&QVQLS)		

vv_random_ansatz	long	初始随机拟设参数个数 ≤0: 初始拟设参数全 0
vv_diff_step	double	差分梯度的差分步长 合法范围: ≥1E-6 非法输入强制转化为 5E-2
vv_loss_monitor	double	损失值收敛判断标准 合法范围: ≥1E-6 非法输入强制转化为 1E-3
vv_learn_rate	vec	学习率列表 合法范围: 非空列表, 且元素均>0 非法输入强制转化为列表: [1E0]
vv_use_multi_ansatz	bool	=true: 开启多个拟设, 稍微增强求解器性能
vv_write_file	bool	=true: 开启 VIRTUAL_VQLS 求解器的计算细节文件输出
模拟器噪声参数		
noise_use	bool	=true: 开启 QPanda 的噪声模拟
ry_noise_error	double	RY 门的噪声系数
cnot_noise_error	double	CNOT 门的噪声系数
NISQ_VQLS 参数 (quantum_method = NISQ_VQLS)		
nv_random_ansatz	long	初始随机拟设参数个数 ≤0: 初始拟设参数全 0
nv_diff_step	double	差分梯度的差分步长 合法范围: ≥1E-6 非法输入强制转化为 5E-2
nv_loss_monitor	double	损失值收敛判断标准 合法范围: ≥1E-6 非法输入强制转化为 1E-3
nv_learn_rate	vec	学习率列表 合法范围: 非空列表, 且元素均>0 非法输入强制转化为列表: [1E0]
nv_task_way	string	真机接入方式 默认: "CLOUD" 可选: "PILOT", "CLOUD"
nv_api	string	用户云平台 API
nv_shots	long	每个线路的 shots 数量 合法范围: [1E2, 1E4]
nv_query_interval	double	计算结果异步查询间隔 合法范围: [2E0, 2E2]
nv_query_timeout	double	计算结果异步查询时间上限 合法范围: [2E2, +\infty)
nv_use_remap	bool	=true, 开启用户自定义比特映射, 否则使用自动映射

nv_qubit_remap	int_map	比特映射列表 格式为{0:0, 1:1}，冒号前为逻辑比特， 冒号后为物理比特
nv_qubit_amend	bool	= true，开启读取噪声缓解
nv_task_limit	long	允许提交的最大任务数 = 0 表示无任务数量限制
nv_sleep	long	求解器运行的延迟时间，单位为秒
nv_write_file	bool	= true，表示开启计算过程中中间数据的 输出
层析相关参数		
tomography_use	bool	= true，表示开启量子层析
tomography_method	string	使用的量子层析解决方案 = SPARSE，表示使用稀疏态层析 = DENSE，表示使用密度矩阵层析
文件路径参数		
root_path	string	config.cfg 配置文件路径
floder_path	string	如果 write_solution_file = true = root_path + “\\QLAS\\”
check_config_path	string	check_config.cfg 配置文件检测路径
result_floder_path	string	结果文件夹存储路径
restart_floder_path	string	续算文件夹存储路径
rst_config_path	string	续算配置文件路径

配置文件中的参数与成员变量一致，配置文件已“END”字符表示结束。一个具体的配置文件示例如表 3 所示。

表 3 QLAS V1.1.0 配置文件示例

<pre>%----- % THIS IS CONFIG FILE OF QLAS V1.1.1% %----- CHECK_CONFIG_FILE = YES WRITE_SOLUTION_FILE = NO SCREEN_PRINT_MONITOR = YES SET_OMP_THREADS = 1 EXCEPTION_HANDLING = NO AUTO_EXCEPTION_HANDLING_INTERVAL = 6E2 RESTART_CALCULATION = NO WRITE_RESTART_FILE = YES SLEEP_TO_RUN = 0E0 %----- SUBSPACE_USE = NO SUBSPACE_METHOD = GMRES SUBSPACE_DIMENSION = 4</pre>
--

```

SUBSPACE_MAXSTEP = 0
SUBSPACE_PRECISION = 1E-3
%-----
PRECONDITION_USE = YES
PRECONDITIONER = ILUT
PRECON_ILUT_P = 4
PRECON_ILUT_TAU = 1E-3
%-----
QUANTUM_METHOD = VIRTUAL_VQLS
QUANTUM_MAXSTEP = 0
QUANTUM_PRECISION = 1E-4
%-----
VV_RANDOM_ANGSATZ = 100
VV_DIFFERENCE_STEP = 1E-3
VV_LOSS_MONITOR = 1E-3
VV_LEARN_RATE = [1E-3, 5E-3, 1E-2, 2E-2, 4E-2, 7E-2, 1E-1, 2E-1, 3E-1, 5E-1]
VV_USE_MULTI_ANGSATZ = NO
VV_WRITE_FILE = NO
%-----
NV_RANDOM_ANGSATZ = 5
NV_DIFFERENCE_STEP = 5E-2
NV_LOSS_MONITOR = 1E-3
NV_LEARN_RATE = [1E-2, 5E-2, 1E-1, 2E-1, 4E-1, 7E-1, 1E0, 2E0, 3E0, 4E0]
NV_TASK_WAY = CLOUD
NV_API =
NV_SHOTS = 10000
NV_QUERY_INTERVAL = 2E0
NV_QUERY_TIMEOUT = 2E4
NV_USE_REMAP = NO
NV_QUBIT_REMAP = {0:0, 1:1, 2:2, 3:3}
NV_USE_AMEND = NO
NV_TASK_LIMIT = 0
NV_WRITE_FILE = NO
%-----
TOMOGRAPHY_USE = YES
TOMOGRAPHY_METHOD = SPARSE
%-----
HHL_PRECISION = 0
%-----
NOISE_USE = YES
RY_NOISE_ERROR = 0.1
CNOT_NOISE_ERROR = 0.1
%-----

```

```
GRAM_DATA_SIZE = 32
GRAM_EPS = 1E-3
%-----
END
```