A quick user guide to the QCMaquis software suite for Molcas

Sebastian Keller, Stefan Knecht, Yingjin Ma, Christopher Stein,
and Markus Reiher
ETH Zürich, Laboratorium für Physikalische Chemie, Vladimir-Prelog-Weg 2,
CH-8093 Zürich

December 5, 2015

release version 1.0

We kindly request that, for reproducibility reasons, any use of the QCMaquis software suite for density matrix renormalization group (DMRG) calculations in Molcas that results in published material should cite the set-up steered by settings and warm-up procedures described in:

Check for a preprint on arXiv.org (to appear soon).
Y. Ma, S. Keller, C. Stein, S. Knecht, R. Lindh, M. Reiher, *in preparation* .

The DMRG calculations are then conducted with the software QCMaquis that requires a citation. It is described in the following paper:

Check for the journal article asap on arXiv.org.
S. Keller, M. Dolfi, M. Troyer, M. Reiher, arXiv:1510.02026 [physics.comp-ph] .

QCMaquis builds upon the ALPS MPS project. The ALPS MPS codes implement the DMRG algorithm for variational ground and low-lying excited state search as well as time

evolution of arbitrary one- and two-dimensional models in a matrix-product-state representation. They have been developed at ETH Zurich by Michele Dolfi and Bela Bauer in the group of Matthias Troyer with contributions from Sebastian Keller and Alexandr Kosenkov and at the University of Geneva by Timothe Ewart and Adrian Kantian in the group of Thierry Giamarchi.

For further information on the ALPS project, please visit alps.comp-phys.org.

Refer to the original ALPS MPS paper:

M. Dolfi, B. Bauer, S. Keller, A. Kosenkov, T. Ewart, A. Kantian, T. Giamarchi, M. Troyer, *Comp. Phys. Commun.* **2014**, 12, 3430. doi:10.1016/j.cpc.2014.08.019

ALPS is a general open-source framework for the description of strongly correlated many-particle systems.

B. Bauer, *et al.* (ALPS Collaboration), The ALPS project release 2.0: open source software for strongly correlated systems, *J. Stat. Mech.* **2011** P05001. http://dx.doi.org/10.1088/1742-5468/2011/05/P05001.

Contents

1 Introduction to the QCMaquis Software Suite

1.1 Overview and Goals

The QCMaquis software suite allows for an efficient optimization of a matrix product state (MPS) wave function based on a second-generation DMRG algorithm [1]. The quantum-chemical operators are represented as matrix product operators (MPOs) which provides the necessary flexibility to accommodate abelian and non-abelian symmetries as well as the implementation of non-relativistic and relativistic quantum chemical Hamiltonians [2], respectively, in a unified framework. We have implemented the special unitary group of degree 2 (SU(2)) in the MPO representation of the non-relativistic Hamiltonian to ensure spin conservation [3]. The current implementation of QCMaquis allows for efficient full-CI-type calculations of active space sizes beyond capabilities ("CAS(18,18)") of standard CI approaches.

The QCMaquis software suite is also available [4] within the framework of Molcas [5] where we have implemented a state-specific and state-average DMRG self-consistent field (DMRG-SCF) algorithm, the possibility to include solvent effects in DMRG calculations and provide analytic gradients for state-specific calculations. The latter enables structure optimization within the QCMaquis DMRG framework.

Advice: The break-even point wrt computational cost for a DMRG-CI/-SCF calculation compared to a "traditional" CAS-CI/-SCF calculation is approximately reached for a CAS(14,14) space. For active spaces smaller than CAS(14,14) we recommend to choose a traditional Molcas CAS approach.

1.2 What features are included in the release version 1.0?

1.2.1 QCMaquis standalone version

The release version 1.0 of the QCMaquis software suite includes:

- optimization of spin-adapted SU(2) MPS wave functions with the DMRG algorithm
- non-relativistic and scalar-relativistic quantum-chemical Hamiltonians

- calculation of excited states
- a python tool set to analyze the MPS wave function and its quantum entanglement

1.2.2 QCMaquis in Molcas

The release version 1.0 of the QCMaquis software suite in Molcas supports:

- DMRG-CI and DMRG-SCF calculations w/wo reaction field (e.g. PCM)
- State-specific and state-averaged DMRG-SCF calculations
- Analytic gradients for state-specific DMRG-SCF calculations

1.3 Organization of this document

In the following we list and briefly summarize the remaining sections of this document.

- Section 2, Software Requirements & Registration, guides through the software requirements and the registration process for QCMaquis.
- Section 3, Installation, guides through the installation process for QCMaquis (and possibly the host program Molcas).
- Section 4, Maintenance, summarizes general information concerning the QCMaquis software suite including how to ask for user support, retreive future patches for the code and where to send bug reports.
- Section 5, General Considerations for Running a QCMaquis DMRG Calculation, introduces the basic workflow for QCMaquis DMRG calculations and provides details about memory requirements as well as input and output data required and generated by QCMaquis, respectively.
- Section 6, Input Keywords, provides in the first part a list of keywords and options to control QCMaquis DMRG calculation in Molcas. The second part explains in detail all mandatory and optional QCMaquis keywords and their usage. In addition we introduce several tools that are part of the QCMaquis software suite which can be used to analyze and visualize the resulting DMRG wave function(s).

- Section 7, Examples of MOLCAS DMRG and QCMAQUIS DMRG Input Files, then shows two example input files for QCMAQUIS DMRG calculations in MOLCAS and with the QCMAQUIS standalone suite, respectively.

2 Software Requirements & Registration

2.1 Prerequisites

In order to install either QCMaquis or the developers version of the quantum chemistry package Molcas with DMRG support through the QCMaquis software suite [1, 4], requires the following libraries/programs:

- Git
- Python version 2.x with $x \geq 5$
- HDF5 (http://www.hdfgroup.org/HDF5)
- GNU Scientific Library GSL (http://www.gnu.org/software/gsl/)
- CMake version 3.x with $x \geq 0$ (https://cmake.org/)

Please make sure that these libraries/programs are available and their location is visible in your `$PATH` and `$LD_LIBRARY_PATH`, respectively. Note that these libraries are NOT part of the installation package of the QCMaquis software suite (see Section 3 for further details).

Warning! The combined installation of Molcas and QCMaquis requires to follow the CMake installation of Molcas, whereas a "configure"-based installation is NOT possible. A detailed step-by-step installation guide is provided in Section 3.2.

2.2 Registration

QCMaquis can be installed either as standalone version (Section 3.1) or as external module of the quantum chemistry software Molcas (Section 3.2). In the former case proceed with the registration as described in Section 2.2.1 else with Section 2.2.2.

2.2.1 QCMaquis standalone version

The use of the standalone version of the QCMaquis software suite requires a registration at http://tc-gitlab.ethz.ch. To sign up at http://tc-gitlab.ethz.ch please send an e-mail to dmrg@phys.chem.ethz.ch. After confirmation of your registration, login to http:

//tc-gitlab.ethz.ch and upload your local public ssh key to your gitlab profile. Verify that you have been granted access to the following projects, for example, they should be listed on the right as projects you are enrolled at:

- QCMaquis/QCMaquis-public
- QCMaquis/QCMaquis-src .

If this is not the case, please send a notification to dmrg@phys.chem.ethz.ch.

2.2.2 QCMaquis in Molcas

The use of the QCMaquis software suite in Molcas requires (apart from a valid Molcas developers license) a registration at http://tc-gitlab.ethz.ch. To sign up at http://tc-gitlab.ethz.ch please send an e-mail to dmrg@phys.chem.ethz.ch. After confirmation of your registration, login to http://tc-gitlab.ethz.ch and upload your local public ssh key to your gitlab profile. Verify that you have been granted access to the following projects, for example, they should be listed on the right as projects you are enrolled at:

- QCMaquis/QCMaquis-public
- QCMaquis/QCMaquis-src
- molcas-dev/hdf5-utils
- molcas-dev/dmrg-interface-utils .

If this is not the case, please send a notification to dmrg@phys.chem.ethz.ch.

3 Installation

Section 3.1 describes in details the installation process for a standalone version of QCMaquis. Ton install QCMaquis as external module of the quantum chemistry software Molcas proceed to Section 3.2.

3.1 QCMaquis standalone version

In the following steps 3.1.1-3.1.5 we describe how to successfully build and install the QCMaquis software suite. The installation of QCMaquis has been tested for different operating systems and compiler/math libraries environments. Their list can be found in Section 3.3. While other combinations might work equally well they are not officially supported.
The installation of the QCMaquis software suite will comprise several libraries which are automatically downloaded and installed during the QCMaquis build process

- QCMaquis
- Boost
- ALPS

All of the above libraries will be installed locally in the user-defined build folder `my-QCMaquis-build`.

3.1.1 Download QCMaquis

Type

```
git clone git@tc-gitlab.ethz.ch:qcmaquis/qcmaquis-public.git my-QCMaquis-src
```

to download a new local QCMaquis repository in the source folder `my-QCMaquis-src`.

3.1.2 Setting up a build folder

Create a build folder `my-QCMaquis-build` – note that this folder does not necessarily have to be a subfolder of `my-QCMaquis-src` – and change to this new folder:

```
mkdir /path/to/my-QCMaquis-build && cd /path/to/my-QCMaquis-build
```

3.1.3 Configuration

Table 1 in Section 3.3 summarizes the list of tested and supported operating system and compiler combinations for the installation of QCMaquis. Below we will show the configuration steps for the most popular compiler suites GNU (Section 3.1.3.1) and Intel (Section 3.1.3.2), respectively. How to setup and use a shared-memory OMP installation of QCMaquis is described in Section 3.1.3.3.

3.1.3.1 Configuration with the GNU compiler suite

To configure QCMaquis with the GNU compiler suite type

```
FC=gfortran CC=gcc CXX=g++ cmake -DQCM_standalone=ON /path/to/my-QCMaquis-src
```

where we assumed that the Intel Math Kernel Library (MKL) is available (recommended option). If the MKL libraries are not available QCMaquis will search for other suitable math libraries installed on the operating system. If none are found the configuration step will stop with an appropriate message.

3.1.3.2 Configuration with the Intel compiler suite

To configure QCMaquis with the Intel compiler suite including MKL type

```
FC=ifort CC=icc CXX=icpc cmake -DQCM_standalone=ON /path/to/my-QCMaquis-src
```

3.1.3.3 Shared-memory OMP parallel configuration

By default QCMaquis is built as shared-memory OMP parallelized version which should work with either compiler suite, GNU or Intel.

In order to exploit the shared-memory OMP parallelization of QCMaquis the user is strongly encouraged to set at runtime the environment variable

```
export OMP_NUM_THREADS=XX
```

where `XX` specifies the number of shared-memory cores to be used. The default (depending on the operating system!!!) is to use a single core.

3.1.4 Building and installation

After a successful configuration, type

```
make
```

or

```
make -j8
```

to compile QCMaquis (in parallel on 8 cores) and install all its components in the build folder `my-QCMaquis-build`. In the same folder Boost and Alps will be downloaded and installed, respectively. The installation process thus requires a working internet connection.

3.1.5 Setting up the runtime environment

After having successfully passed the QCMaquis installation step as indicated by CMake messages like

```
[xxx%] Built target alps
...
[xxx%] Built target qcmaquis
```

and

```
[xxx%] Installation of QCMaquis, ALPS and Boost was successful!
```

adjust your runtime environment variables (assuming a bash environment) as follows:

```
export PATH=/path/to/my-QCMaquis-build/alps/bin:$PATH
export PATH=/path/to/my-QCMaquis-build/qcmaquis/bin:$PATH
export PYTHONPATH=/path/to/my-QCMaquis-build/alps/lib:$PYTHONPATH
export PYTHONPATH=/path/to/my-QCMaquis-build/qcmaquis/lib/python/pyeval:$PYTHONPATH
export PYTHONPATH=/path/to/my-QCMaquis-build/qcmaquis/lib/python:$PYTHONPATH
```

On Linux systems (x86_64) set in addition

```
export LIBRARY_PATH=/path/to/my-QCMaquis-build/alps/lib:$LIBRARY_PATH
export LD_LIBRARY_PATH=/path/to/my-QCMaquis-build/alps/lib:$LD_LIBRARY_PATH
```

whereas on Mac OS X set

```
export LIBRARY_PATH=/path/to/my-QCMaquis-build/alps/lib:$LIBRARY_PATH
export DYLD_LIBRARY_PATH=$LIBRARY_PATH:$DYLD_LIBRARY_PATH
```

Full instructions ready for copy+paste will also be printed on the screen and shall be copied to either your local `$HOME/.bashrc` file, or be executed in the QCMaquis bash session.

Alternative: use the "sourceable" configuration file `qcmaquis.sh` that is created after the build/install step succeeded. To do so type

```
source /path/to/my-QCMaquis-build/qcmaquis/bin/qcmaquis.sh
```

3.2 QCMaquis as external module of Molcas

In the following steps 3.2.1-3.2.7 we describe how to successfully build and install the Molcas program together with the QCMaquis software suite. The installation of QCMaquis has been tested for different operating systems and compiler/math libraries environments. Their list can be found in Section 3.3. While other combinations might work equally well they are not officially supported.
The installation of the QCMaquis software suite within Molcas will comprise several libraries which are automatically downloaded and installed during the Molcas build process provided that DMRG support within Molcas is requested by the user. The list of external libraries comprises:

- QCMaquis
- QCMaquis-driver-utils
- Boost
- ALPS

All of the above libraries will be installed locally in the user-defined build folder `my-Molcas-build` of Molcas.

3.2.1 Download Molcas

If you already have a local Molcas repository in `my-Molcas-src` that points to the remote repository `git@molcas:molcas` skip this section, otherwise choose either to download a new local Molcas repository (Section 3.2.1.1) or add a new remote repository to an existing local

MOLCAS repository (in `my-Molcas-src`) that does NOT yet have a pointer to the remote repository `git@molcas:molcas` (Section 3.2.1.2).

3.2.1.1 Cloning a new local Molcas repository

Type

```
git clone --recursive git@molcas:molcas my-Molcas-src
```

to download a new local MOLCAS repository in the source folder `my-Molcas-src`.

3.2.1.2 Adding a new remote repository to an existing local Molcas repository

Type

```
git remote add gitsrc_molcas git@molcas:molcas
```

to add the additional remote repository `gitsrc_molcas` that hosts the MOLCAS-DMRG branch to your local MOLCAS repository (which we assume in the following to be located in `my-Molcas-src`). To obtain all remote informations from `gitsrc_molcas` type next

```
git fetch gitsrc_molcas
```

3.2.2 Switch to the Molcas-DMRG branch

QCMAQUIS is available on the `master` branch of MOLCAS which is the default branch that you currently reference if you "`git cloned`" a new local MOLCAS repository in the previous step. Typing

```
git branch
```

should then yield something like

```
* master
```

The "$*$" in front of the branch name indicates which local branch you are currently tracking. In this particular case you are tracking the branch `master`.

If you added a new remote repository ("`gitsrc_molcas`") to your existing local MOLCAS repository (see Section 3.2.1.2) type

```
git checkout -b dmrg-master gitsrc_molcas/master
```

Typing next

```
git branch
```

should then yield something like

```
master
*dmrg-master
```

The "∗" in front of the branch name indicates which local branch you are currently tracking. In this particular case you successfully switched to track the development branch `dmrg-master`.

3.2.3 Initialization of git submodules

Type

```
git submodule update --init --recursive
```

to initialize (and possibly download) the necessary git submodules of MOLCAS.

3.2.4 Setting up a build folder

Create a build folder `my-Molcas-build` – note that this folder does not necessarily have to be a subfolder of `my-Molcas-src` – and change to this new folder:

```
mkdir /path/to/my-Molcas-build && cd /path/to/my-Molcas-build
```

3.2.5 Configuration

Table 1 in Section 3.3 summarizes the list of tested and supported operating system and compiler combinations for the simultaenous installation of MOLCAS and QCMAQUIS. Below we will show the configuration steps for the most popular compiler suites GNU (Section 3.2.5.1) and Intel (Section 3.2.5.2), respectively. How to setup an MPI-parallel MOLCAS installation is described in Section 3.2.5.3.

> Note! We strongly recommend to configure QCMAQUIS (and MOLCAS) with the Intel Math Kernel Library (MKL) to ensure the best numerical performance.

3.2.5.1 Configuration with the GNU compiler suite

To configure MOLCAS and QCMAQUIS with the GNU compiler suite type

```
FC=gfortran CC=gcc CXX=g++ cmake -DDMRG=ON -DLINALG=MKL /path/to/my-Molcas-src
```

where we assumed (`-DLINALG=MKL`) that the MKL libraries are available (recommended option). If the MKL libraries are not available internal math libraries of MOLCAS can be requested with `-DLINALG=Internal` (default option for `LINALG` in MOLCAS).

3.2.5.2 Configuration with the Intel compiler suite

To configure MOLCAS and QCMAQUIS with the Intel compiler suite including MKL type

```
FC=ifort CC=icc CXX=icpc cmake -DDMRG=ON -DLINALG=MKL /path/to/my-Molcas-src
```

3.2.5.3 MPI-parallel and shared-memory OMP parallel configurations

Installing an MPI-parallel version of MOLCAS with DMRG support is possible although QCMAQUIS itself is by default shared-memory OMP but not yet MPI-parallelized. To configure MOLCAS for an MPI-installation type

```
FC=mpif90 CC=mpicc CXX=mpiCXX cmake -DDMRG=ON /path/to/my-Molcas-src
```

A shared-memory OMP parallelized version of MOLCAS can be activated with the option `-DOPENMP=ON` passed to `cmake` during the configuration step, for example

```
FC=... CC=... CXX=... cmake -DDMRG=ON -DOPENMP=ON /path/to/my-Molcas-src
```

It should work with either compiler suite, GNU or Intel, but the user may want to consult the MOLCAS user manual for further information.

In order to exploit the shared-memory OMP parallelization of QCMAQUIS which is enabled by default the user is strongly encouraged to set at runtime the environment variable

```
export QCMaquis_CPUS=XX
```

where `XX` specifies the number of shared-memory cores to be used. The default is to use a single core.

3.2.6 Building and installation

After a successful configuration, type

```
make
```

or

```
make -j8
```

to compile MOLCAS (in parallel on 8 cores) and install all its components in the build folder `my-Molcas-build`. In the same folder QCMAQUIS as well as the required BOOST and ALPS libraries will be downloaded and installed, respectively. The installation process thus requires a working internet connection.

3.2.7 Setting up the runtime environment

After having successfully passed the QCMAQUIS installation step as indicated by CMake messages like

```
[xxx%] Built target alps
...
[xxx%] Built target qcmaquis
```

and

```
[xxx%] Installation of QCMaquis, ALPS and Boost was successful!
```

adjust your runtime environment variables (assuming a bash environment) as follows:

```
export PATH=/path/to/my-Molcas-build/alps/bin:$PATH
export PATH=/path/to/my-Molcas-build/qcmaquis/bin:$PATH
export PYTHONPATH=/path/to/my-Molcas-build/alps/lib:$PYTHONPATH
export PYTHONPATH=/path/to/my-Molcas-build/qcmaquis/lib/python/pyeval:$PYTHONPATH
export PYTHONPATH=/path/to/my-Molcas-build/qcmaquis/lib/python:$PYTHONPATH
```

On Linux systems (x86_64) set in addition

```
export LIBRARY_PATH=/path/to/my-Molcas-build/alps/lib:$LIBRARY_PATH
export LD_LIBRARY_PATH=/path/to/my-Molcas-build/alps/lib:$LD_LIBRARY_PATH
```

whereas on Mac OS X set

```
export LIBRARY_PATH=/path/to/my-Molcas-build/alps/lib:$LIBRARY_PATH
export DYLD_LIBRARY_PATH=$LIBRARY_PATH:$DYLD_LIBRARY_PATH
```

Full instructions ready for copy+paste will also be printed on the screen and shall be copied to either your local `$HOME/.bashrc` file, or be executed in the MOLCAS bash session.

Alternative: use the "sourceable" configuration file `qcmaquis.sh` that is created after the build/install step succeeded. To do so type

```
source /path/to/my-Molcas-build/qcmaquis/bin/qcmaquis.sh
```

3.3 Supported operating systems and compiler environments

Table 1 summarizes tested and officially supported operating system/compiler/math library combinations for building MOLCAS together with the QCMAQUIS software suite. Note that other combinations might work equally well but they are not officially supported at present. See the MOLCAS manual at www.molcas.org for details on supported operating systems and compiler environments of a "plain" MOLCAS installation.

Table 1: Supported operating system/compiler/math library combinations for the combined installation of MOLCAS and QCMAQUIS.

operating system	compiler	math library
Mac OS X (10.11 "El Capitan")	GNU 5.2	"Accelerate" (Xcode)
Mac OS X (10.11 "El Capitan")	Intel XE 15 (patch 0)	MKL 11.2
x86_64	Intel XE 15 (patch 0)	MKL 11.2
x86_64	GNU 4.8	MKL 11.1
x86_64	GNU 4.7.2	MKL 11.1
x86_64	Intel XE 13	MKL 11.1
x86_64	Intel XE 13 (sp 1)	MKL 11.1
x86_64	OpenMPI 1.6.5 + Intel XE 13 (sp 1)	MKL 11.1

3.4 First tests and verification of the installation

To verify that your installation of MOLCAS and QCMAQUIS was successful, type in your build folder `my-Molcas-build`

```
molcas verify qcmaquis
```

A message like

```
Verification has been completed
```

indicates that all tests passed correctly and your installation is good to go for production work.

Note that all test inputs can be found in `my-Molcas-build/test/qcmaquis`. They may also serve as sample inputs for your actual calculation. Table 2 comprises a short summary of the seven tests for QCMaquis covering different computational aspects within the Molcas framework.

Table 2: List of test inputs that are stored in `my-Molcas-build/test/qcmaquis`.

file	type	details
001.input	State-specific	introduces the DMRG keyword under the RASSCF card
002.input	State-average	two-state (S_0/S_1) DMRG-SCF of N_2
003.input	DMRG-CI	dynamic list of renormalized states m
004.input	State-specific	structure optimization of the ground state of dioxetanone
005.input	State-specific	structure optimization of the S_1 state of N_2
006.input	PCM model	DMRG-SCF for a non equilibrium state
007.input	State-specific	DMRG-SCF with manual orbital ordering

4 Maintenance

4.1 New versions and patches

New versions of the QCMaquis and/or QCMaquis-driver software suite as well as possible bug fixes will be announced on the QCMaquis homepage. The latter patches will be made available as git commit(s) on the QCMaquis repository at `http://tc-gitlab.ethz.ch` that can be directly applied ("`git merge`"d) to the user's release version of QCMaquis.

4.2 Reporting bugs and user support

The QCMaquis program suite is distributed to the Molcas community with no obligations on the side of the authors. The authors thus take no responsibility for the performance of the code nor for the correctness of the results. This distribution policy gives the authors no responsibility for problems experienced by the users when using the QCMaquis program in Molcas.

Bugs/suggestions for improvements are to be reported as issues on the gitlab server. Please open an issue concerning:

- QCMaquis on the project page `QCMaquis/QCMaquis-public`
- QCMaquis in Molcas on the project page `molcas-dev/dmrg-interface-utils`

Any issue will be dealt with by one of the authors, although no responsibility on the promptness of response is given. In general, serious bugs that have been reported and fixed will lead to a new patch of the QCMaquis program, announced and distributed from the QCMaquis homepage.

5 General Considerations for Running a QCMaquis DMRG Calculation

Running a QCMaquis DMRG calculation — either DMRG-CI or DMRG-SCF — within the framework of Molcas involves the QCMaquis host program driver described in Ref. [4]. In doing so, Molcas-DMRG calculations are easily accessible since they are integrated in the RASSCF module of Molcas. Hence, if the user considers to run a Molcas-DMRG calculation rather than doing a standalone QCMaquis DMRG calculation section 5.3 may be skipped.

5.1 Memory management and memory requirements

The memory layout of QCMaquis is designed for large multinode architectures and therefore the memory consumption on a single node is not consequently minimized.

Note that if you run a Molcas-QCMaquis DMRG calculation, the memory consumption and memory allocation of QCMaquis is entirely disconnected from the host program Molcas. This in turn means that the amount of core memory assigned to the host program should be carefully chosen. For example, DMRG-SCF calculations for large active spaces will not require a considerably larger amount of memory in the RASSCF module (apart from memory for reduced one- and two-particle density matrices that scale approximately with the number of active orbitals N_{act} as N_{act}^2 and N_{act}^4, respectively) than regular CASSCF calculations with modest-sized active spaces. The reason being that in the former case we do not need to reserve memory for CI vector(s) expansions which grow roughly factorily with the number of electrons and N_{act}.

5.2 Files required and written by QCMaquis

QCMaquis requires only two files to start a calculation, (i) an input file, named for example `dmrg-input` that includes all keywords and options as specified in Section 6.2 and (ii) an integral file in the `FCIDUMP` format as described in Ref. [6]. The latter can be produced for example with the Molcas/QCMaquis-host program driver described in Ref. [4]. Different types of reference orbitals may be used as starting orbitals for a QCMaquis DMRG calculation.

QCMaquis produces two types of files, (i) a `result-file` in which all information on, e.g. the energies and expectation values, is stored and (ii) a `checkpoint-folder` which contains the matrix product state (MPS) wave function. The `checkpoint-folder` is required for a later restart of the calculation, while the tools and the analysis scripts described in Sections 6.3 and 6.4, respectively, require either a `result-file` or a `checkpoint-folder` or both. All data is stored in the `hdf5` format.

5.3 Typical workflow for a DMRG calculation including a post-processing analysis

The usual workflow to set up, run and analyze a DMRG calculation proceeds as follows:

- prepare an FCIDUMP integral file [6] using the Molcas-QCMaquis host program driver [4]) starting from a set of previously computed reference orbitals
- prepare an input file `dmrg-input` starting for example from the template input file provided in Section 7.2 and adjust all molecular system and wave function specific parameters (for example `nelec, spin, irrep, L, ...`, see Sections 6.2.1-6.2.3 for a list of all compulsory and optional input arguments)
- run the DMRG calculation with
  `dmrg dmrg-input (| tee out)`
- compute expectation values with
  `dmrg_meas dmrg-input`
- analyze the results using the Python tools provided with the QCMaquis package (see Sections 6.3 and 6.4, respectively, for a list of utility programs and scripts)

6 Input Keywords

Caution! Be aware of underscores "_" in some of the QCMaquis or Molcas keywords. They might get lost when you copy+paste keywords from this pdf to your input file.

6.1 Keywords and Options for Molcas

6.1.1 Compulsory keywords

This section describes the QCMaquis input to the RASSCF program within the Molcas program system. The minimal input to run a DMRG-SCF calculation with QCMaquis in the framework of the RASSCF program is

```
&RASSCF
DMRG
```

where the keyword `DMRG` is compulsory to activate the DMRG functionality in Molcas.

6.1.2 Optional keywords

Table 3 summarizes optional keywords under the RASSCF input card that allow to control the type of DMRG calculation, that is, DMRG-SCF, DMRG-CI or just a dump of information (input, one- and two-electron integrals) to perform a DMRG calculation outside of Molcas. The detailed input control for QCMaquis in Molcas is designed such that parameters between the RGINput and ENDRG section of the RASSCF input card only affect QCMaquis that is, they have no effect within the RASSCF module of Molcas. These parameters are described in Section 6.2.

6.1.3 Inoperative keywords

Table 4 comprises a list of keywords under the RASSCF input card which are have no meaning when a DMRG calculation is requested.

Table 3: Optional keywords under the RASSCF card to control the DMRG calculation.

keyword	description

`RGINput`
`XXX`
`ENDRG`
	`RGInput` marks the beginning of the QCMaquis parameter control section while `ENDRG` marks its end. Any QCMaquis-internal keyword can be forwarded directly to QCMaquis by replacing the `XXX` with a list of keywords, each keyword in a new line. A list of compulsory, optional, and property calculation keywords for QCMaquis are described in Sections 6.2.1-6.2.3.
	Notable exceptions are the active-space and wave function symmetry dependent parameters `L, nelec, spin, irrep` that will be forwarded automatically by Molcas.

`CIONLY`	This keyword disables orbital optimization. A single DMRG-CI calculation is performed instead.
`FCIDUMP`	This keyword disables orbital and wave function optimization, that is, input information and transformed integrals are written to a formatted integral file named "FCIDUMP" (see Ref. [6] for a detailed description of the format) which can be used in subsequent QCMaquis DMRG calculations.
`SOCCupy`	Initial electronic configuration for the calculated state(s). For the calculation of a single state this is equivalent to the `hf_occ` card in the QCMaquis input (see Section 6.2). The occupation is inserted as a string of blank space separated aliases of occupations of the active (RAS2) orbitals with the aliases 2 = full, u = up, d = down, 0 = empty.

Table 4: Inoperative keywords in the RASSCF section when DMRG is active.

keyword	description
`RASScf`	This keyword is not available because it is not possible to restrict the excitation level between subspaces in DMRG calculations.
`TIGHt`	This keyword is not available because the Jacobi-Davidson diagonalization is independent and can be controlled with the `ietl_jcd_tol` and `ietl_jcd_maxiter` parameters (see Section 6.2) in the RGINPUT section.

6.1.4 Molcas environment variables

As described in Section 3.2.5.3 QCMaquis is built by default with a shared-memory OMP parallelization. To speedup calculations the user can thus set at runtime the environment variable

```
export QCMaquis_CPUS=XX
```

where `XX` specifies the number of shared-memory cores to be used. The default is to use a single core.

6.2 Keywords and Options for QCMaquis

In the following we describe (i) compulsory keywords (Section 6.2.1), (ii) optional keywords (Section 6.2.2) for QCMaquis DMRG calculations as well as (iii) keywords for property calculations (Section 6.2.3). Most of the QCMaquis keywords have default settings that guarantee convergence in the general case and are inserted automatically by the host program (Molcas in this case). A reasonable choice of default values for optional keywords is given in our example QCMaquis input file in Section 7.2.

Caution! Maquis has many features beyond quantum chemistry, e.g. related to solid state physics. Some keywords listed in the example file in Section 7.2 are therefore not explained in the following and are not to be changed, if a quantum chemical calculation is desired.

6.2.1 Compulsory keywords

The keywords in Table 5 have to be set for every DMRG calculation since they may crucially affect the accuracy of the final result. Their choice depends for example on the molecule under consideration (do you expect strong static electron correlation and/or dynamical correlation to play a major role), the nature of the reference orbitals (Hartree-Fock orbitals, natural orbitals of some kind, ...), the size of the active space, and many other aspects.

Note! We strongly encourage any new user of QCMaquis or QCMaquis-in-Molcas to carefully read first Ref. [7] which gives a detailed introduction to DMRG calculations in quantum chemistry. Further quantum chemical DMRG sample calculations starting from different computational setups are discussed for example in Refs. [4, 8, 9, 10].

Some of the compulsory keywords listed in Table 5 are indeed automatically set if you run a QCMaquis DMRG calculation through the QCMaquis host program driver which is the case for Molcas-QCMaquis DMRG calculations. In this case skip the upper part of Table 5 and proceed immediately to the lower part marked by "Keywords NOT set by the host program Molcas".

Table 5: Compulsory keywords to be set in all QCMaquis DMRG calculations.

keyword	description
	Keywords set by the host program Molcas
`nelec`	Total number of electrons.
`irrep`	Irreducible representation of the point group symmetry of the target state. **Note:** Counting starts with 0 which has to be the totally symmetric representation.
`spin`	Total spin ($2 \times S$) of the target state, for example: `spin=0` (singlet), `spin=1` (doublet), `spin=2` (triplet), ...
`L`	Length of lattice = number of orbitals in the active space.
`integral_file`	Path and filename of the integral file, for example `integral_file = /path/to/file/FCIDUMP`
`chkpfile`	Path and name of the folder in which the MPS is stored.
`resultfile`	Path and filename of the result file.
`n_ortho_states`	If an excited state calculation is desired, the number of states the current wave function is to be orthogonalized against shall be specified here.
`ortho_states`	Path(s) and filename(s) of the MPS checkpoint file(s) that store the lower lying states to which the current MPS shall be orthogonal to.

Table 5 – continued from previous page

keyword	description
`init_state`	Possible options are `default`, `thin` and `hf`. The `default` and `thin` initializations fill the initial MPS with random numbers, the difference being that a singular value decomposition reduces the bond dimension to `init_bond_dimension` in the case of `thin`. Usage of `hf` generates an MPS consisting of only the determinant defined on the `hf_occ` card. Note that the CI-DEAS procedure [11, 4] (as invoked by `dmrginit.py`, see Section 6.4) behaves like a restart from the newly generated CI-DEAS MPS. **default in Molcas:** `init_state = "default"`
	Keywords NOT set by the host program Molcas
`max_bond_dimension`	Maximum number of renormalized states (commonly referred to as m-value or virtual bond dimension) kept during each microiteration step of a forward- or backward sweep.
`nsweeps`	Maximum number of DMRG sweeps. Please be aware that `nsweeps` sets the number of combined forward and backward sweeps. Thus, the actual number of sweeps is 2 × `nsweeps`.

6.2.2 Optional keywords

The keywords summarized in Table 6 may be exploited by the more experienced user but can be safely ignored by those who just want to get started. They may affect the convergence and accuracy of the final result, though. For the inexperienced user however, we advise to not change these settings and accept the default values provided by QCMaquis.

Table 6: Optional keywords for QCMaquis calculations.

keyword	description
`orbital_order`	Manual ordering of the orbitals along the one dimensional lattice. The order has to be entered as a string of comma separated orbital numbers. We recommend the Fiedler ordering based on the mutual information [4, 12] which can be obtained by means of the python script `fiedler.py` (see Section 6.4). **default**: `orbital_order = "1,2,3,4,5,6,..."`
`sweep_bond_dimensions`	Sets `max_bond_dimension` for each sweep separately. Note: Replaces `max_bond_dimension` which does NOT need to be specified in this case. Example (`nsweeps=3`): `sweep_bond_dimensions`="300,400,500"
`init_bond_dimension`	Adjusts the maximal bond dimension of the MPS produced by the CI-DEAS procedure [11, 4].
`conv_thresh`	Sets the energy convergence threshold (in Hartree). If the lowest energy from the previous sweep differs from the lowest energy of the current sweep by less than `conv_thresh`, the DMRG calculation stops. Note: Requires to set also `truncation_final` and `ietl_jcd_tol`. Numerical format: `xe-y` with `x` and `y` being integers. Example: `conv_thresh = 1e-6`.
`ietl_jcd_tol`	Convergence threshold for the Jacobi-Davidson diagonalization. Numerical format: `xe-y` with `x` and `y` being integers. Example: `ietl_jcd_tol = 1e-6`.
`ietl_jcd_maxiter`	Maximum number of iterations in the Jacobi-Davidson diagonalization.

Table 6 – continued from previous page

keyword	description
`truncation_initial`	If during the `ngrowsweeps`, the sum of the discarded singular values for m retained states is lower than the value defined here, more block states will be discarded until the discarded sum increases to `truncation_initial`. Numerical format: `xe-y` with `x` and `y` being integers. Example: `truncation_initial = 1e-6`.
`truncation_final`	If during the `nmainsweeps`, the sum of the discarded singular values for m retained states is lower than the value defined here, more block states will be discarded until the discarded sum increases to `truncation_final`. Numerical format: `xe-y` with `x` and `y` being integers. Example: `truncation_final = 1e-6`.
`measure_each`	Tells the program to compute the expectation values every $2 \times$ `measure_each` sweeps.
`symmetry`	Defines the total symmetry group. Default is the combined spin- (SU2) and point symmetry group (PG) `su2u1pg`, where the `pg`-suffix should be omitted for better performance if the molecule is C1-symmetric. For test purposes, it is possible to switch off spin-adaptation, again with or without point group symmetry: `2u1(pg)` . In the latter case the keywords `spin` and `nelec` (see Section 6.2.1) have no meaning. Instead `u1_total_charge1` and `u1_total_charge2` corresponding to the number of up and down electrons have to be specified.
`chkp_each`	Tells the program to update the checkpoint file every $2 \times$ `chkp_each` sweeps.

Table 6 – continued from previous page

keyword	description
`hf_occ`	Occupation of the starting orbitals (e.g. Hartree-Fock occupation) to be entered as a comma separated string of occupation aliases. The aliases are defined as follows: 4 = full, 3 = up, 2 = down, 1 = empty. This information has to be entered in case of `hf` as `init_state` and for the CI-DEAS procedure [11, 4] as invoked by `dmrginit.py`. example: `hf_occ = "4,4,4,2,2,1"` for a CAS(8,6) triplet state setup where the unpaired electrons are "located" in orbital # 4 and # 5.
`nmainsweeps`	Number of sweeps in which `truncation_final` is used in the singular value decomposition.
`ngrowsweeps`	Number of sweeps in which `truncation_initial` is used in the singular value decomposition.

6.2.3 Keywords for expectation value calculations

QCMaquis can (in principle) compute expectation values for any one- or two-particle operator that can be formulated in second quantization. Table 7 comprises a list of the available property keywords in the release version of QCMaquis. For further updates/other properties please contact dmrg@phys.chem.ethz.ch. The one-particle reduced density matrix as well as the one-particle spin-density matrix are implicitly computed from the expectation values of some of the operators contained in `MEASURE[ChemEntropy]`.

Table 7: Expectation value calculations available in the release version of QCMaquis.

keyword	description
`MEASURE[ChemEntropy]`	All expectation values over the operators required to calculate the mutual information (as specified in Ref. [13]) will be computed. Please note that this is available only for SU2 symmetry.
`MEASURE[1rdm]`	Computes the one-particle reduced density matrix, without the additional correlators contained in the ChemEntropy measurement.
`MEASURE[2rdm]`	Computes the two-particle reduced density matrix.
`MEASURE_LOCAL[`*name*`] = "`*op*`"`	Computes $\langle\psi\|op_i\|\psi\rangle$, $i = 1 \dots L$. `Nup`, `Ndown` and `Nup*Ndown` are meaningful choices for *op*. Available for `2u1(pg) only.` Note: *name* defines the name under which the expectation values are stored on the `resultfile`.
`MEASURE_HALF_CORRELATIONS [`*name*`] = "`op_1`:`op_2`"`	Computes $\langle\psi\|op_{1i}op_{2j}\|\psi\rangle$, $i = 1 \dots L, j = i+1 \dots L$. `Nup`, `Ndown`, `Nup*Ndown`, `cdag_up`, `cdag_down`, `c_up`, `c_down`, `cdag_up*Ndown`, `c_up*Ndown`, `cdag_down*Nup`, `c_down*Nup`, `cdag_up*cdag_down`, `c_up*c_down`, `cdag_up*c_down`, and `cdag_down*c_up`, as op_1 and op_2 are recognized by the program. Available for `2u1(pg) only.` Note: *name* defines the name under which the expectation values are stored on the `resultfile`.

6.3 QCMaquis tools

QCMaquis comes with several tools that allow for example further manipulation of the MPS or to acquire additional wave function analysis information. The tools `det2mps_"symmetry"`,

`mps2ci` and `mps_transform(_"pg")` will be briefly described in the following. These tools are provided in the `my-Molcas-build/qcmaquis/bin` folder.

Table 8: Overview of QCMaquis tools.

tool	description
`mps_transform(_pg)`	This tool allows for a transformation of an `su2u1(pg)` MPS wave function to `2u1(pg)` symmetry. Command line: `mps_transform(_pg) chkpfile` Note: for an `su2u1(pg)` MPS with S $>$ 0 `2u1(pg)` MPSs for all S_z components are generated.
`det2mps_symmetry`	This tool generates determinants based on the CI-DEAS procedure [11] and inserts them in an MPS from which a new DMRG calculation can be started. Starting from such an MPS is likely to improve convergence behaviour and is less prone to get stuck in local minima. The current implementation is described in Ref. [4]. The number of determinants will be chosen such that the maximal bond order at any site does not exceed the value set in `init_bond_dimension`. The new MPS will be stored in the checkpoint folder specified in `chkpfile`. Command line: `det2mps_"symmetry" dmrg-input` Note: "`symmetry`" must equal the total symmetry specified in the input file `dmrg-input`.
`mps_overlap_symmetry(_pg)`	This tool calculates the overlap between two MPS Θ_1 and Θ_2, respectively, according to $\langle\Theta_1\|\Theta_2\rangle$. Command line: `mps_overlap_"symmetry"(_pg) chkpfile_1 chkpfile_2` Note: "`symmetry`" must equal the full symmetry specified in the input file `dmrg-input`.

Table 8 – continued from previous page

tool	description
`mps2ci_2u1(pg)`	Given a text file `determant_list.txt` containing a list of determinant strings, this tool calculates the CI-coefficients of the respective determinants [14]. The determinant strings have to encode the occupation of the orbitals as described for the `hf_occ` keyword (4 = full, 3 = up, 2 = down, 1 = empty). Command line: `mps2ci_2u1(pg) chkpfile determinant_list.txt` Note: with the conversion tool `mps_transform(_pg)` this analysis becomes possible also for MPS of the full symmetry `su2u1(pg)`.

6.4 QCMaquis PYTHON scripts for wave function analysis and visualization

The python scripts of QCMaquis are helpful to analyze and visualize the results of a DMRG calculation. Their usage should be evident from the documentation strings in the Python files. However, those that are most frequently used will be briefly explained here. All scripts except `dmrginit.py` take the QCMaquis output file `resultfile` as input. They are located in the folder `my-Molcas-build/qcmaquis/lib/python/pyeval`.

Table 9: Overview of QCMaquis Python analysis and visualization scripts.

script	description
`sweeps.py`	Plots the energy for each microiteration. Command line: `sweeps.py resultfile` Note: use this tool to check the convergence wrt the number of sweeps.

Table 9 – continued from previous page

script	description
`dmrginit.py`	Starts a QCMaquis DMRG calculation with `max_bond_dimension = 200` and `nsweeps = 2`, measures the entropy information [15, 13] from this unconverged calculation and based on this, generates a new MPS with the CI-DEAS procedure according to all settings specified in the input file `dmrg-input`. We recommend to use this script for the preparation of calculations for active spaces that are larger than those that can be handled with traditional CAS methods. Command line: `dmrginit.py dmrg-input`
`fiedler.py`	Optimizes the ordering based on entropy information as proposed in Ref. [12]. The current implementation is described in Ref. [4]. The ordering ensures that highly entangled orbitals are close to each other in the one dimensional lattice. The first ordering in the output ignores the point group symmetry of the orbitals, while the second version orders the orbitals within each irreducible representation and then reorders these symmetry blocks. Command line: `fiedler.py resultfile` Note: we recommend to use the second option.
`input.py`	This script recovers the complete QCMaquis input file `dmrg-input` from a given `resultfile`. Command line: `input.py resultfile`

Table 9 – continued from previous page

script	description
`mutinf.py`	Produces mutual information plots [13] given that an expectation value calculation for `MEASURE[ChemEntropy]` has been performed. Command line: `mutinf.py resultfile` Note: if orbital images (in .png format) named 1.png, 2.png, ..., L.png (with L being the number of active orbitals) are present in the same folder where `mutinf.py` is executed, they can be added to the mutual information plot by providing the optional argument `-i` to `mutinf.py`.

7 Examples of Molcas DMRG and QCMaquis DMRG Input Files

7.1 Example file for Molcas – DMRG-SCF(14,10)

```
&GATEWAY
  2
Angstrom
  N 0.000000 0.000000 -0.54880
  N 0.000000 0.000000  0.54880
  basis=cc-pvdz
&SEWARD
&SCF
&RASSCF
  DMRG
  RGinput                    !!!! QCMaquis Keywords input
    conv_thresh = 1.0E-07
    nsweeps = 4
    max_bond_dimension = 100
  endRG                      !!!! End of QCMaquis Keywords input
  inactive= 0 0 0 0 0 0 0 0
  RAS2= 3 1 1 0 3 1 1 0
  ITER= 10,100
&Grid_it
  all
```

7.2 Example file for QCMaquis– DMRG-CASCI(8,8)

```
//active-space dependent parameters
L = 8
nelec = 8
// target symmetry (spin and spatial) of the wave function:
// 2*spin = 0 (singlet) + totally symmetric point group irrep
spin = 0
irrep = 0
//parameter to control the actual DMRG calculation
nsweeps = 8
max_bond_dimension = 256
conv_thresh = 1e-6
// initialization procedure
init_state = 'default'
// technical parameters
symmetry = 'su2u1pg'
integral_cutoff = 1e-40
truncation_initial = 1e-50
truncation_final = 1e-7
chkpfile = 'chkp.h5'
resultfile = 'result.h5'
integral_file = "FCIDUMP"
storagedir = '/scratch/$USER/boundaries'
LATTICE = "orbitals"
lattice_library = "coded"
MODEL = "quantum_chemistry"
model_library = "coded"
//all expectation value calculations required for entropy measures
MEASURE[ChemEntropy]
```

References

[1] S. Keller, M. Dolfi, M. Troyer, M. Reiher, arXiv:1510.02026.
"An efficient matrix product operator representation of the quantum-chemical Hamiltonian"

[2] S. Battaglia, A. Muolo, S. Keller, S. Knecht, and M. Reiher, *in preparation.*

[3] S. Keller and M. Reiher, *in preparation.*

[4] Y. Ma, S. Keller, C. Stein, S. Knecht, R. Lindh, M. Reiher, *in preparation.*

[5] www.molcas.org

[6] P. J. Knowles, N. C. Handy, Comp. Phys. Commun. **1989**, 54, 75.
"A determinant based full configuration interaction program"

[7] S. F. Keller and M. Reiher, Chimia **2014**, 68, 200–203, arXiv:1401.5497.
"Determining Factors for the Accuracy of DMRG in Chemistry."

[8] L. Freitag, S. Knecht, S. F. Keller, M. G. Delcey, F. Aquilante, T. Bondo Pedersen, R. Lindh, M. Reiher, and L. González, Phys. Chem. Chem. Phys. **2015**, DOI:10.1039/C4CP05278A.
"Orbital entanglement and CASSCF analysis of the RuNO bond in a Ruthenium nitrosyl complex."

[9] T. Dresselhaus, J. Neugebauer, S. Knecht, S. Keller, Y. Ma, and M. Reiher, J. Chem. Phys. **2015**, 142, 044111. arXiv:1409.1953.
"Self-consistent embedding of density-matrix renormalization group wavefunctions in a density functional environment."

[10] E. D. Hedegaard, S. Knecht, J. S. Kielberg, H. J. Aa. Jensen, and M. Reiher, J. Chem. Phys. **2015**, 142, 224108. arXiv:1502.06157.
"Density matrix renormalization group with efficient dynamical electron correlation through range separation."

[11] Ö. Legeza, J. Sólyom, Phys. Rev. B **2003**, 68, 195116.
"Optimizing the density-matrix renormalization group method using quantum information entropy"

[12] G. Barcza, Ö. Legeza, K. H. Marti, M. Reiher, Phys. Rev. A **2011**, 83, 012508.
"Quantum-information analysis of electronic states of different molecular structures"

[13] K. Boguslawski, P. Tecmer, Ö. Legeza, M. Reiher, J. Phys. Chem. Lett. **2012**, 3, 3129.
"Entanglement measures for single- and multireference correlation effects"

[14] G. Moritz, M. Reiher, J. Chem. Phys. **2007**, 126, 244109.
"Decomposition of density matrix renormalization group states into Slater determinant basis"

[15] J. Rissler, R. M. Noack, S. R. White, Chem. Phys. **2006**, 323, 519.
"Measuring orbital interaction using quantum information theory"