A quick user guide to the SCINE-QCMaquis software suite for OpenMolcas

Leon Freitag, Sebastian Keller, Stefan Knecht, Yingjin Ma, Christopher Stein and Markus Reiher

November 21, 2018

ETH Zürich, Laboratorium für Physikalische Chemie,
Vladimir-Prelog-Weg 2,
CH-8093 Zürich

release version 2.1

We kindly request that, for reproducibility reasons, any use of the QCMaquis software suite for density matrix renormalization group (DMRG) calculations in OpenMolcas 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).**
> Freitag L.; Keller S.; Knecht S.; Lindh R.; Ma Y.; Stein C. J.; Reiher M. *in preparation*.

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

> Keller, S.; Dolfi, M.; Troyer, M.; Reiher, M. *J. Chem. Phys.* **2015**, *143*, 244118, doi:10.1063/1.4939000.

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 Timothée 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 computational cost of traditional full CI and full CI-based multiconfigurational calculations (such as CASSCF) scales exponentially with the number of active orbitals, thus calculations beyond a certain limit (e. g. 18 electrons in 18 orbitals for CASSCF) become prohibitively expensive. A possible approach to overcome the scaling limit is to employ methods such as density matrix renormalization group (DMRG) [1–4] to solve the full CI problem approximately.

In DMRG, usually the wave function is represented in form of a matrix product state (MPS). The QCMaquis software suite solves a full CI problem approximately by means of an efficient optimization of an MPS wave function based on a second-generation DMRG algorithm [5]. 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 [6], 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 [7]. 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 comes with a Fortran/Python-based interface [8] to the OpenMolcas [9] framework, 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, analytic gradients for state-specific calculations, state interaction in a nonorthogonal basis[10] and n-electron valence state perturbation theory employing the DMRG reference wave function.[11] These developments allow for, among others, structure optimizations, spin-orbit coupling calculations, evaluation of transition densities between non-orthogonal states for electronic and magnetic properties, as well as to account for dynamic correlation by means of the NEVPT2 approach.

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 OpenMolcas CAS approach.

1.2 What features are included in the release version 2.1?

1.2.1 QCMaquis standalone version

The release version 2.1 of the QCMaquis software suite includes:

- Optimization of spin-adapted SU(2) MPS wave functions with the DMRG algorithm (DMRG-CI calculations)
- Non-relativistic and scalar-relativistic quantum-chemical Hamiltonians
- Calculation of reduced density matrices of up to fourth order and transition reduced density matrices of up to third order
- Calculation of excited states
- A python tool set to analyze the MPS wave function and its quantum entanglement

1.2.2 QCMaquis in OpenMolcas

Additionally, the release version 2.1 of the QCMaquis software suite in OpenMolcas 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
- MPS state interaction (MPS-SI) for the calculation of spin-orbit coupling matrix elements, electronic and magnetic properties
- state-specific and quasi-degenerate (multi-state) DMRG-NEVPT2 calculations

1.3 Licenses

The QCMaquis-OpenMolcas interface and the NEVPT2 program are licensed under GNU LGPL, version 2.1. The QCMaquis software suite and the ALPS library are distributed under ALPS Application License version 1.0 and the ALPS Library License version 1.0.

2 Software Requirements & Download

2.1 Prerequisites

In order to install either QCMaquis or the quantum chemistry package OpenMolcas with DMRG support through the QCMaquis software suite [5, 8], requires the following libraries/programs:

- Git
- Python version 2.x with x ≥ 7, with SciPy and NumPy
- HDF5 (http://www.hdfgroup.org/HDF5) with Python2 bindings (h5py)
- SZIP
- GNU Scientific Library GSL (http://www.gnu.org/software/gsl/)
- CMake version ≥3.7 (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 OpenMolcas and QCMaquis may require *both* python 2 and 3 to be installed on the system. Most operating systems allow such setup without problems. Please make sure that Python 2 is available on the system using the `env python2` command.

2.2 Downloading QCMaquis

Advice: We recommend to install QCMaquis with the OpenMolcas-QCMaquis interface. QCMaquis installed with the OpenMolcas-QCMaquis interface is also fully usable as a standalone version.

If you want to install QCMaquis with the OpenMolcas-QCMaquis interface, please proceed to Section 3.1. No manual download is necessary: QCMaquis and OpenMolcas-QCMaquis interface will be automatically downloaded installed during the compilation and installation procedure of OpenMolcas.

If you want to download and install a standalone version of QCMaquis, it may be obtained from https://scine.ethz.ch/download/qcmaquis. After entering your

name, surname and e-mail address, you will be able to download the QCMaquis install script. After downloading, please proceed to Section 7.1 for installation instructions.

3 Installation

3.1 Installation of the OpenMolcas-QCMaquis interface

The installation of QCMaquis has been tested for different operating systems and compiler/math libraries environments found in Section 3.2. While other combinations might work equally well they are *not* officially supported.
The installation of the QCMaquis software suite within OpenMolcas will comprise several libraries which are *automatically downloaded and installed* during the OpenMolcas build process provided that DMRG support within OpenMolcas is requested by the user. The list of external libraries comprises:

- QCMaquis
- QCMaquis-driver-utils
- Boost
- ALPS
- Optionally: the NEVPT2 program

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

3.1.1 Download and build folder setup

Download OpenMolcas from the official OpenMolcas GitLab repository:

```
git clone git@gitlab.com:Molcas/OpenMolcas.git /path/to/my-Molcas-src
```

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
```

3.1.2 Configuration

To install QCMaquis and the QCMaquis-OpenMolcas interface, one may follow the standard OpenMolcas installation procedure and enable the CMake option `DMRG` (and `NEVPT2` for the NEVPT2 program).

3.1.2.1 Configuration with the GNU compiler suite

To configure OpenMolcas and QCMaquis with the GNU compiler suite type

```
FC=gfortran CC=gcc CXX=g++ cmake -DDMRG=ON -DLINALG=MKL \
  -DQCMaquis_NAME="Your Name" -DQCMaquis_EMAIL="your@email.com" \
  /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 OpenMolcas can be requested with `-DLINALG=Internal` (default option for `LINALG` in OpenMolcas) or `-DLINALG=Runtime` (to use a system-wide BLAS or LAPACK installation). Please also specify your name and e-mail address in the `-DQCMaquis_NAME` and `-DQCMaquis_EMAIL` CMake options, otherwise the download of QCMaquis will not work.

Note: We strongly recommend to configure QCMaquis (and OpenMolcas) with the Intel Math Kernel Library (MKL) to ensure the best numerical performance. See Section 3.2 for further details.

Note: Intel compiler is currently not supported. Running a binary compiled with Intel compilers may result in segmentation faults. We are currently investigating the issue.

3.1.2.2 Upgrading an existing OpenMolcas installation

If you're upgrading from an OpenMolcas installation compiled with an older release of QCMaquis, you need to add an additional command line flag when invoking `cmake`, namely

```
-DQCMaquis_UPDATE=ON
```

This flag activates the QCMaquis update for the current and all the following `cmake` runs. To switch off QCMaquis updates, `cmake` may be run with

```
-DQCMaquis_UPDATE=OFF
```

3.1.2.3 Additional options

Following additional options can be switched on by passing additional flags to `cmake`:

- `-DNEVPT2=ON`: Downloads and compiles the NEVPT2 program and its interface with OpenMolcas and QCMaquis.

- `-DOPENMP=ON`: Enable OpenMP parallelization for MOLCAS and NEVPT2 (see Section 3.1.2.4). Note that QCMaquis is built with OpenMP support *regardless* of this flag.

Other options to enable/disable various options in OpenMolcas may be passed to `cmake` at this step. Please consult the OpenMolcas manual for details.

> Note: it is currently discouraged to install the QCMaquis-OpenMolcas interface together with the Block or CheMPS2 interfaces available in OpenMolcas. This setup has not been tested and may lead to an unusable installation. A fix allowing different interfaces to be installed at the same time is currently under development.

3.1.2.4 MPI parallel and shared-memory OMP parallel configurations

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

```
FC=mpif90 CC=mpicc CXX=mpiCXX cmake -DDMRG=ON -DLINALG=MKL \
  -DQCMaquis_NAME="Your Name" -DQCMaquis_EMAIL="your@email.com" \
  /path/to/my-Molcas-src
```

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

```
FC=... CC=... CXX=... cmake -DDMRG=ON -DLINALG=MKL \
  -DQCMaquis_NAME="Your Name" -DQCMaquis_EMAIL="your@email.com" \
  /path/to/my-Molcas-src
```

It should work with either compiler suite, GNU or Intel, but the user may want to consult the OpenMolcas 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` or `export OMP_NUM_THREADS=XX` where `XX` specifies the number of shared-memory cores to be used. The default is to use a single core.

3.1.3 Building and installation

After a successful configuration, type

```
make
```

or

```
make -j8
```

to compile OpenMolcas (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.

3.1.4 Setting up the runtime environment

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

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

set up your OpenMolcas environment by setting `MOLCAS` environment variable

```
export MOLCAS=/path/to/my-MOLCAS-build
```

and then set up the QCMaquis runtime environment by running the following shell script

```
source $MOLCAS/qcmaquis/bin/qcmaquis.sh
```

You might want to include the above lines in your local `$HOME/.bashrc` to set up your environment automatically every time your session is started.

3.2 Supported operating systems and compiler environments

QCMaquis in OpenMolcas is tested and officially supported for Linux (x86_64) and MacOS X (10.12.x and 10.13.x) operating systems. We support GCC compilers starting from 5.x up to 7.x along with the MKL (Linux, x86_64) and MKL/Accelerate (Mac OS X) math libraries, respectively, to successfully build the QCMaquis software suite within OpenMolcas. Note that other combinations might work equally well but they are not officially supported at present.
See the OpenMolcas manual at www.molcas.org for details on supported operating systems and compiler environments of a “plain" OpenMolcas installation.

3.3 Tests and verification of the installation

To verify that your installation of OpenMolcas and QCMaquis was successful, run the following command

```
pymolcas verify qcmaquis
```

The message

```
Verification has been completed
```

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

Test inputs are distributed with OpenMolcas and can be found in the `qcmaquis` subdirectory of the OpenMolcas test directory. They may also serve as sample inputs for your actual calculation. Table 1 comprises a short summary of the tests for QCMaquis covering different computational aspects within the OpenMolcas framework.

Table 1: List of QCMaquis test inputs in OpenMolcas.

file	type	features
001.input	State-specific	Ground-state DMRG-SCF optimization of N_2
002.input	State-average	Two-state (S_0/S_1) DMRG-SCF of N_2
003.input	DMRG-CI	DMRG-CI with a 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	MPS-SI	Calculation of singlet-triplet spin-orbit matrix elements
008.input	State-specific	DMRG-SCF with Fiedler ordering and CI-DEAS start guess
009.input	NEVPT2	DMRG-NEVPT2 energy calculation of the CH_2 triplet state

3.4 Maintenance

3.4.1 Updates and patches

New versions of the QCMaquis software suite as well as possible bug fixes will be announced on the SCINE QCMaquis homepage. They can be then downloaded from the SCINE website or by rerunning CMake in the OpenMolcas directory.

3.4.2 Reporting bugs and user support

The QCMaquis program suite is distributed to the OpenMolcas 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 OpenMolcas.

Bugs/suggestions for improvements are to be reported as follows:

- For issues regarding usage of QCMaquis in OpenMolcas, open an issue in the OpenMolcas bug tracker at `https://gitlab.com/Molcas/OpenMolcas/issues`
- For issues regarding the usage of QCMaquis standalone version, please send an e-mail to dmrg@phys.chem.ethz.ch.

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 via the SCINE QCMaquis homepage.

4 Running a QCMaquis DMRG Calculation

Running a QCMaquis DMRG calculation — either DMRG-CI or DMRG-SCF — is most easily achieved with the QCMaquis-OpenMolcas interface [8], which integrates the DMRG calculation into the OpenMolcas workflow. Usage of the QCMaquis-OpenMolcas interface described in Section 4.1. In a few advanced cases the user might want to run QCMaquis directly, which will be described in Section 8.2.

4.1 Keywords and Options for OpenMolcas

4.1.1 The DMRGSCF module

Cite the following papers when performing DMRG-SCF calculations within the OpenMolcas software suite:

1. Freitag, L.; Keller, S.; Knecht, S.; Lindh, R.; Ma, Y.; Stein, C. J.; Reiher, M. *in preparation* **Check for a preprint on arXiv.org (to appear soon).**
2. Keller, S.; Reiher, M., *J. Chem. Phys.* **2016**, *144*, 134101 doi:10.1063/1.4944921
3. Keller, S.; Dolfi, M.; Troyer, M.; Reiher, M., *J. Chem. Phys.* **2015**, *143*, 244118 doi:10.1063/1.4939000
4. Knecht S.; Hedegård E. D.; Keller S.; Kovyrshin A.; Ma Y.; Muolo A.; Stein C. J.; Reiher M., *Chimia* **2016**, *70* 244–251, doi:10.2533/chimia.2016.244.

The QCMaquis-OpenMolcas interface introduces a new OpenMolcas module, `DMRGSCF` which may be used to perform DMRG-CI or DMRG-SCF wave function optimizations. A `DMRGSCF` input (see example in Listing 1) consists of the mandatory keyword `ActiveSpaceOptimizer` and the two input blocks `DMRGSettings...EndDMRGSettings` and `OOptimizationSettings...EndOOptimizationSettings`. Optional keywords within the `DMRGSCF` input section are `Fiedler` and `CIDEAS`, respectively.

4.1.1.1 The `ActiveSpaceOptimizer` keyword

The `ActiveSpaceOptimizer` keyword specifies the CI solver program to be used in the `DMRGSCF` module. Currently, only one value, namely
`ActiveSpaceOptimizer=QCMaquis`
is supported, which, obviously, calls the QCMaquis program. In the future, this option will be used to enable interfaces to other DMRG programs such as Block or CheMPS2.

4.1.1.2 The `Fiedler` keyword

The `Fiedler` keyword, i.e.,

```
Fiedler=on
```

enables a state-specific orbital ordering for the MPS optimization by exploiting concepts from graph theory. The ordering follows from the elements of the Fiedler vector [12, 13] which is the eigenvector corresponding to the second lowest eigenvalue of the so-called graph Laplacian. Further details concerning our implementation of the Fiedler-vector ordering within QCMaquis can be found in Ref.[8].

4.1.1.3 The `CIDEAS` keyword

The `CIDEAS` keyword, i.e.,

```
CIDEAS=on
```

enables a more advanced algorithm to construct a suitable initial MPS (see the keyword `init_state` in Table 7 for other options) provided by the configuration interaction dynamically extended active space (CI-DEAS) approach [14, 15]. The CI-DEAS protocol can be interpreted as an orbital entanglement entropy guided configuration selection and the quality of this initial guess depends on the quality of the initial CAS vector. Further details concerning our implementation of the CI-DEAS approach within QCMaquis can be found in Ref.[8].

> **Warning!** The CI-DEAS functionality is currently restricted to calculations performed with C_1 symmetry. Support for other point group symmetries will be available in due time.
> **Note**: The `CIDEAS` option requires to set the "HF occupation" for each state in the `OOptimizationSettings` input section by means of the `SOCC` keyword (see Table 4).

4.1.1.4 The `DMRGSettings` block

The `DMRGSettings` block contains DMRG-specific options, which will be passed on to the QCMaquis program. The mandatory keywords, which must be present in each calculation, are summarized in Table 2. In addition to these keywords, any QCMaquis keyword (see Table 8 may be used in this section.

4.1.1.5 The `OOptimizationSettings` block

The `OOptimizationSettings` block contains general, non DMRG-specific options required for the wavefunction optimisation (such as number of the active electrons, active orbital specification etc.) and resemble the input for an analogous CASCI or CASSCF calculation with the OpenMolcas `RASSCF` module. Most of the `RASSCF` keywords are accepted, with the exception of the keywords in Table 3. Please consult the `RASSCF` entry of the OpenMolcas manual for a detailed description of the input.

In addition to the standard `RASSCF` keywords, several optional keywords are available, listed in Table 4.

Table 2: Mandatory and special QCMaquis keywords to be specified in the DMRGSettings block of the DMRGSCF input.

keyword	description
max_bond_dimension	Maximum number of renormalized states (commonly referred to as m-value or maximum bond dimension) kept during each microiteration step of a 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 $2 \times$ nsweeps.

Table 3: Inoperative RASSCF keywords in the OOptimizationSettings block.

keyword	description
RASScf GASScf	Keywords are unavailable as excitation level restrictions between subspaces are unsupported 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 7.2) in the DMRGSettings section.

4.1.2 DMRG calculations with the RASSCF module

Note: The input format for DMRG/DMRGSCF calculations with the RASSCF module of OpenMolcas described in this section is **deprecated** and will be **unavailable** in future versions of OpenMolcas!
Instead, please use the DMRGSCF module as described in Section 4.1.1.

Instead of using the new DMRGSCF module, DMRG calculations with QCMaquis are also supported from within the RASSCF module directly. Compared to the DMRGSCF input, the RASSCF input changes as follows:

- The DMRG keyword must be specified in the RASSCF input.
- The DMRGSettings block from DMRGSCF input is delimited with RGInput...EndRG instead of DMRGSettings...EndDMRGSettings
- The OOptimizationSettings block is just the normal RASSCF input, i. e. it is not delimited with OOptimizationSettings...EndOOptimizationSettings

An input example is provided in Listing 2.

Table 4: Optional non-standard keywords in the `OOptimizationSettings` block.

keyword	description
`FCIDUMP`	Skip the optimization and write out the transformed active MO integrals to a `FCIDUMP` file in `$WorkDir` (see Ref. 16 for a detailed description of the format) which can be used in subsequent QCMaquis DMRG calculations.
`SOCCupy`	Initial electronic configuration for the calculated state(s). This keyword is equivalent to the `hf_occ` card in the QCMaquis input (see Section 7.2), but allows input for multiple states. The occupation is inserted as a string (strings) of aliases of occupations of the active (RAS2) orbitals with the aliases 2 = full, u = up, d = down, 0 = empty. For several states, the occupation strings for each state are separated by newlines.
Options required for DMRG-NEVPT2 calculations	
`NEVPT2prep`	Prepare for a subsequent DMRG-NEVPT2 calculation. If this keyword is followed by a parameter `EvRDM`, then the four- and transition three-particle density matrices (4- and t-3RDMs), required for the NEVPT2 calculation, will be evaluated and stored on disk in `$WorkDir`. Otherwise, QCMaquis input files for the 4- and t-3RDMs evaluation are prepared and the RDM evaluation may be performed externally. More about external RDM evaluation in Section 6.3.

4.1.3 Molcas environment variables

As described in Section 3.1.2.4 QCMaquis is built by default with a shared-memory OMP parallelization. To speedup calculations the user can thus set at runtime the environment variable `QCMaquis_CPUS` or `OMP_NUM_THREADS` to the number of shared-memory cores to be used. The default is to use a single core.

4.1.4 Input Examples

Listing 1: (File `001.input` in the test directory) Input example for a DMRG-SCF calculation with the `DMRGSCF` module (N_2, 6 orbitals, with symmetry).

```
&GATEWAY
 coord
 2
Angstrom
 N  0.000000 0.000000 -0.54880
 N  0.000000 0.000000  0.54880
 basis=cc-pvdz
&SEWARD
&SCF
&DMRGSCF
ActiveSpaceOptimizer=QCMaquis
DMRGSettings
  nsweeps           = 4
  max_bond_dimension = 100
EndDMRGSettings
OOptimizationSettings
  inactive = 2 0 0 0 2 0 0 0
  RAS2     = 1 1 1 0 1 1 1 0
  LINEAR
EndOOptimizationSettings
```

Listing 2: Input example for a DMRG-SCF calculation with the `RASSCF` module (N_2, 6 orbitals, with symmetry)

```
&GATEWAY
 coord
 2
Angstrom
 N 0.000000 0.000000 -0.54880
 N 0.000000 0.000000  0.54880
 basis=cc-pvdz
&SEWARD
&SCF
&RASSCF
DMRG
RGINPUT
  nsweeps           = 4
  max_bond_dimension = 100
EndRG
inactive = 2 0 0 0 2 0 0 0
RAS2     = 1 1 1 0 1 1 1 0
LINEAR
```

5 State interaction with DMRG (MPS-SI)

Cite the following papers when performing MPS-SI calculations within the OpenMolcas software suite:

1. Knecht, S.; Keller S.; Autschbach J.; Reiher M., *J. Chem. Theory Comput.* **2016**, *12* 5881–5894, doi:10.1021/acs.jctc.6b00889
2. Malmqvist, P.-Å.; Roos, B. O., *Chem. Phys. Lett.* **1989**, *155*, 189–194 doi:10.1016/0009-2614(89)85347-3

The state-interaction module for MPS wave functions (MPS-SI) introduced in Ref. [17] is based on the `CASSI/RASSI` framework [18, 19] of OpenMolcas. MPS-SI is a generalized state-interaction approach for both *nonorthogonal* and *orthonormal* spin-free MPS wave functions which enables the evaluation of arbitrary one- and two-particle transition matrix elements as well as, for example, matrix elements of the spin-orbit coupling operator. For instance, diagonalization of the spin-orbit Hamiltonian matrix yields spin-orbit coupled wave functions as linear combinations of the uncoupled, spin-pure MPS states. The latter can (but do not have to) be obtained as results from one or several DMRG-SCF orbital optimization calculations as described in Section 4.1.1.

Following the work of Malmqvist [20], the central element of the MPS-SI algorithm is the transformation of the bra and ket MPS wave functions to a biorthonormal basis representation. It is important to note that the latter transformation is not needed if the MPS wave functions considered for state interaction share a common MO basis. In this particular case, the MPS-SI program directly proceeds with the calculation of the reduced (transition) one- and two-particle density matrices. We emphasize that our approach is applicable to the general case with MPS wave functions built from mutually nonorthogonal molecular orbital bases. It therefore provides the desired flexibility to find the best individual molecular orbital basis to represent wave functions of different spin and/or spatial symmetry. After solving a generalized eigenvalue equation of the form

$$\mathbf{Hc} = E\mathbf{Sc}\,, \tag{1}$$

with the Hamiltonian matrix **H** expressed in the basis of the DMRG-SCF MPS wave functions and the wave function overlap matrix **S**, a set of fully orthogonal and non-interacting states are obtained as linear combinations of the DMRG-SCF MPS wave functions with the expansion coefficients given by **c**.

Moreover, it is possible to “dress" the diagonal elements of the Hamiltonian in Eq. (1) for `MPS-SI` by adding a correlation-correction term obtained, for example, from a preceding NEVPT2 calculation (see Section 6), by either using the `HDIAG` keyword within the `RASSI` module or provide the `nevpt2.h5` wave function file as input

option to the &MPSSI module (see the description of the keyword FILE in the OpenMolcas manual for the MPSSI module for details on the correct syntax).

5.1 Running MPS-SI

An MPS-SI calculation requires reference DMRG-SCF or NEVPT2 wave functions as obtained with the DMRGSCF (see Section 4.1.1) or NEVPT2 modules (see Section 6), respectively.

> Note: Currently, the MPS-SI implementation supports all point groups implemented in OpenMolcas if there is only one set of input states (i.e., all states have the same spatial and spin symmetry). By contrast, if there is more than one set of input states of different spin symmetry, the MPS-SI module supports only calculations in C_1 symmetry. The latter restriction will be removed in a forthcoming release.

Listing 3 shows an input example for an MPS-SI calculation (a calculation of spin-orbit coupling matrix elements for the CH_2 molecule) where the important keywords are highlighted in blue. A detailed explanation of the input is given below.

Listing 3: (File 007.input in the QCMaquis test subdirectory) Input example for a one-shot MPS-SI spin-orbit calculation (Singlet and triplet state, CH_2 molecule, 6 electrons in 6 orbitals, C1 symmetry).

```
&GATEWAY
  coord
  3
  CH2 Triplet coordinates in Angstrom
  C 0.000000 0.000000 0.000000
  H 0.000000 0.000000 1.077500
  H 0.784304 0.000000 -0.738832
  basis=ANO-RCC-MB
  Group=Nosym
  AMFI
  ANGMOM
  0.0 0.0 0.0
&SEWARD
&DMRGSCF
  ActiveSpaceOptimizer=QCMaquis
  DMRGSettings
    max_bond_dimension=1024
    nsweeps=10
  EndDMRGSettings
  OOptimizationSettings
    Spin=3
```

```
    Inactive=1
    Ras2=6
    NActEl=6,0,0
  EndOOptimizationSettings
  >> COPY $Project.JobIph JOBOLD
  >> COPY $Project.dmrgscf.h5 $Project.trip.h5
  >> EXEC mv $CurrDir/$Project.checkpoint_state.0.h5 $CurrDir/$Project.trip.
    checkpoint_state.0.h5
  >> EXEC $MOLCAS/pytools/qcm_checkpoint_rename.py $Project.trip.h5 -q
&DMRGSCF
ActiveSpaceOptimizer=QCMaquis
  DMRGSettings
    max_bond_dimension=1024
    nsweeps=10
  EndDMRGSettings
  OOptimizationSettings
    Spin=1
    Inactive=1
    Ras2=6
    NActEl=6,0,0
    JobIph
  EndOOptimizationSettings
>> COPY $Project.dmrgscf.h5 $Project.sing.h5
>> EXEC mv $CurrDir/$Project.checkpoint_state.0.h5 $CurrDir/$Project.sing.
    checkpoint_state.0.h5
>> EXEC $MOLCAS/pytools/qcm_checkpoint_rename.py $Project.sing.h5 -q
&MPSSI
  Nrof
  2 1 1
  1
  1
  FILE
  2
  $Project.trip.h5
  $Project.sing.h5
  EJOB
  SOCOupling
  0.0001
  SPIN
```

In order to calculate spin-orbit matrix elements within the MPS-SI framework, one needs to add the keyword AMFI (and optionally Angmom) in the &GATEWAY input section (This is no different from a standard RASSI calculation). Then, we perform two DMRG-SCF calculations with the &DMRGSCF module (see Section 4.1.1) to calculate the lowest-lying triplet and singlet state of CH_2. At the end of each DMRG-SCF step, we save the dmrgscf.h5 files for each spin state for later usage in &MPSSI.

The largest differences to standard RASSI calculations are found in lines 28 and 29 in Listing 3. Line 28, in addition to the `dmrgscf.h5` file, also saves the QCMaquis checkpoint folder `$Project.checkpoint_state.0.h5` under a different name, which would otherwise be overwritten in the subsequent DMRG-SCF calculation. The information about the QCMaquis checkpoint name is saved in the `dmrgscf.h5` file, hence the reference to the new name must be updated in the `dmrgscf.h5` file with an utility script named `qcm_checkpoint_rename.py` in line 29. In the second DMRG-SCF calculation, these steps are repeated in lines 44 and 45.

The MPS-SI framework is driven by the `&MPSSI` module, which derives from the OpenMolcas `&RASSI` module and has the same input syntax. The spin-orbit matrix-element calculation is triggered with `SPIN`. Note, that we use the keyword `EJOB` which prevents the MPS-SI program to calculate any two-particle reduced (transition) density matrices which are not only rather expensive to calculate compared to the one-electron reduced (transition) density matrices but also are of no further usage for the calculation of spin-orbit matrix-elements. The diagonal values for the Hamiltonian in Eq.(1) are taken in this case from the respective `$Project.xxx.h5` files (with `xxx=sing,trip`).

> Note: For MPS-SI, the wavefunctions from the RASSCF/DMRG-SCF calculations *MUST* be provided as `dmrgscf.h5` files. Old-style `JobIph` files are not supported.

5.2 Input options for MPS-SI

Table 5: `&MPSSI` module keywords not present in the OpenMolcas `&RASSI` module.

keyword	description
`QDSC`	Read the Hermitian effective Hamiltonian from a preceding QD-SC-NEVPT2 calculation for state mixing in MPS-SI.
`QDPC`	Same as above, but for the QD-PC-NEVPT2 effective Hamiltonian.

6 The NEVPT2 module

Cite the following papers when performing NEVPT2 calculations within the OpenMolcas software suite:

1. Freitag, L.; Knecht, S.; Angeli, C.; Reiher, M. *J. Chem. Theory Comput.* **2017**, *13*, 451–459
   doi:10.1021/acs.jctc.6b00778

2. Angeli, C.; Cimiraglia, R.; Malrieu, J.-P. *J. Chem. Phys.* **2002**, *117*, 9138
   doi:10.1063/1.1515317

NEVPT2 is a second-order perturbation theory with a CAS (or a CAS-like) reference wavefunction originally developed by Angeli et al. [21–24] In contrast to CASPT2, it uses a Dyall Hamiltonian[25] as the zeroth-order Hamiltonian and is therefore inherently free of intruder states and parameters such as the IPEA shift. NEVPT2 exists in two formulations – the strongly- (SC-) and the partially-contracted NEVPT2 (PC-NEVPT2), which differ in the basis of the first-order wavefunction expansion. The implementation in the NEVPT2 module is based on the original NEVPT2 implementation by Angeli et al. [23, 24], with the implementation of the QCMaquis DMRG reference wave function and Cholesky decomposition for the two-electron integrals[11, 26]. For excited states both single-state and multi-state calculations with the QD-NEVPT2 approach[24] are supported.

6.1 Running NEVPT2

Prior to running a NEVPT2 calculation, one must obtain a reference wavefunction with the :program:‘DMRGSCF‘ program and perform an integral transformation with the :program:‘MOTRA‘ program.

Currently, the implementation supports *only* QCMaquis DMRG reference wavefunctions (support for CASSCF reference wavefunctions will be added in the near future). It is nevertheless possible to run NEVPT2 with a CASSCF reference wavefunction by performing a DMRG-CI calculation with a sufficiently large m value using the CASSCF converged orbitals. For example, an m value of 2000 recovers the exact CASCI energy up to 5×10^{-8} a. u. for active spaces of up to 14 orbitals. Listing 4 shows an input example for a NEVPT2 calculation, the important keywords are highlighted in blue, and below we will explain the input example in more detail.

Listing 4: (`009.input` in the QCMaquis test subdirectory) Input example for a one-shot NEVPT2 calculation (Singlet CH_2, 6 electrons in 6 orbitals, RICD)

```
1 &GATEWAY
2   coord
```

```
  3
  CH2 Triplet coordinates in Angstrom
  C      0.000000 0.000000 0.000000
  H      0.000000 0.000000 1.077500
  H      0.784304 0.000000 -0.738832
  basis=cc-pVTZ
  Group=Nosym
  RICD
  CDTH=1.0E-7
&SEWARD
&DMRGSCF
  ActiveSpaceOptimizer=QCMaquis
  DMRGSettings
      max_bond_dimension=128
      nsweeps=5
  EndDMRGSettings
  OOptimizationSettings
      Spin=3
      Inactive=1
      Ras2=6
      NActEl=6,0,0
      NEVPT2Prep
      EvRDM
  EndOOptimizationSettings
&MOTRA
  Frozen=0
  CTOnly
  Kpq
  HDF5
&NEVPT2
```

First, one performs a DMRG-SCF calculation with the keywords `NEVPT2Prep` and (optionally) `EvRDM`, which enable the evaluation of the 4-RDMs (and, in case of multiple states, also t-3RDMs) required by NEVPT2. If `EvRDM` is specified, the RDMs are evaluated immediately, otherwise an input file is prepared for separate RDM calculations with QCMaquis (see also Table 6). The latter option is particularly useful for large active spaces, where offloaded evaluation can be performed in a massively parallel manner, which will be explained in detail in Section 6.3.

> Note: The computational cost of the RDM evaluation grows as N^8 with the number of active orbitals, therefore we strongly recommend to use the distributed RDM evaluation for active spaces larger than 10-11 orbitals described in Section 6.3.

Second, one must perform an integral transformation with the `MOTRA` module. If no Cholesky decomposition or RICD is used in the calculation, the only mandatory

keyword is HDF5, which enables the write-out of the transformed integrals in the HDF5 format required by the NEVPT2 module. If Cholesky decomposition is used, one additionally needs to add the keys CTOnly and Kpq.

> Note: The Cholesky decomposition currently does not support symmetry, and the support for frozen orbitals in MOTRA with Cholesky is untested, hence also the keyword Frozen=0 is recommended!

Finally, one calls the NEVPT2 module with &NEVPT2. It has no mandatory options, but options described in Section 6.2 can be specified. If a distributed RDM evaluation is performed, the NEVPT2 module must be called in a separate calculation.

6.2 Options to the NEVPT2 module

Table 6: NEVPT2 keywords.

keyword	description
STATes	Number of electronic states to calculate. Default: 1
NOMS	Omit the QD-NEVPT2 calculation and perform single-state NEVPT2 calculations instead.
MULT	Select specific states to perform QD-NEVPT2 calculation. Followed by a list of whitespace-separated state numbers, preceded by their total amount. Example: MULT=3 1 2 4 for states 1, 2, 4 of a preceeding DMRG-SCF calculation of 4 roots (or more). MULT=ALL includes all states and is the default.
FILE	Specify the path to a JobIph or .h5 file with the reference wavefunction. By default, the reference wavefunction is read from JOBIPH.
FROZen	Specify the number of frozen orbitals. The number of frozen orbitals may be specified in two ways: if only one number n is specified, then all orbitals from 1 to n are frozen. Otherwise, it is possible to freeze specific orbitals with the SELECT keyword which follows the FROZEN keyword. In this case, the total number of frozen orbitals followed by the space-separated list of frozen orbitals must be entered. Note that if symmetry is used, the orbital numbering for all symmetries is still consecutive, e. g. the 1^{st} orbital of symmetry 2 is has the number $m+1$ if there are m orbitals in symmetry 1.
NOPC	Disable the PC-NEVPT2 calculation. If the option is not present (default), both SC-NEVPT2 and PC-NEVPT2 calculations are performed.

Table 6 – continued from previous page

keyword	description
`SKIPk`	Skip the calculation of Koopmans' matrices. Requires a file named `nevpt.h5` obtained from a previous calculation in the scratch directory. May be useful to restart a previous crashed calculation if it crashed past the calculation of Koopmans' matrices, and may save some computational time, especially for large active spaces.

6.3 Massively parallel distributed 4-RDM calculations (experimental)

The computational cost of the RDM evaluation grows as N^8 with the number of active orbitals, and becomes prohibitively expensive even for moderate active spaces. Therefore we provide an (experimental) python utility `jobmanager.py` for distributed massively parallel 4-RDM calculations. With distributed 4-RDM calculations, active spaces of up to 22 orbitals can be employed in DMRG-NEVPT2 calculations without any approximation to the 4-RDM.

`jobmanager.py` splits the evaluation of the 4-RDM $G_{ijklmnop}$ into four-index subblocks with indices i,j,k,l. Due to permutational symmetry and the properties of the creation and annihilation operators, $i >= j >= k >= l$ and no more than two indexes are equal (pairwise equality $i = j$ and $k = l$ is allowed). The script prepares input files and submits a separate job for each subblock, and merges the subblocks into the full matrix once the jobs are finished. The script is expected to be run on a head node of a distrubuted computing system with a batch system: LSF has been tested, but any batch system which supports the DRMAA library, such as Slurm or PBS, should work.

Note that the DMRG-SCF/NEVPT2 calculation need not be performed on the same system as the 4-RDM evaluation.

How to run NEVPT2 calculations with distributed 4-RDM evaluation

6.3.1 Prerequisites

- Python >= 2.7.9
- DRMAA library compatible with your batch submission system (e. g. LSF-DRMAA)
- Python DRMAA
- GNU Parallel (optional)

If your system administrator has not set up DRMAA and Python DRMAA, you might need to download and install these libraries yourself. After the installation, the environment variable `DRMAA_LIBRARY_PATH` must be set to the path to `libdrmaa.so` and,

if Python does not find the DRMAA Python binding, also PYTHONPATH to the path of the Python DRMAA library.

6.3.2 Workflow

1. Run DMRGSCF and MOTRA calculations as shown in e. g. Listing 4, but **omit** the EvRDM and &NEVPT2 keywords. Without the EvRDM keyword, the NEVPT2Prep only creates QCMaquis templates and the MPS checkpoint files required for a later 4-RDM and/or t-3RDM evaluation.

2. Copy the $MOLCAS/Tools/distributed-4rdm/prepare_rdm_template.sh script to the MOLCAS scratch directory and run it. The script will create subdirectories named 4rdm-scratch.⟨state⟩ for each state.

   If you wish to perform the 4-RDM evaluation on a different machine (e. g. a cluster), copy the subdirectory for each state to that machine. If you do not wish to evaluate the 4-RDM for all states, pass the list of desired states as parameters to the prepare_rdm_template.sh script. For example, ./prepare_rdm_template.sh {0..2} or ./prepare_rdm_template.sh 0 1 2 will create the scratch directories for states from 0 to 2 (note that QCMaquis starts counting states with 0).

3. For each state, change to the 4rdm-scratch.⟨state⟩ subdirectory and run
   nohup jobmanager.py &
   (Login to the machine where you evaluate the 4-RDM before if you wish to run the evaluation on a different machine.) This will create a subdirectory for each batch job (corresponding for each four-index 4-RDM subblock) and submit the jobs. The script will stay in the background until all the jobs have completed. The script also accepts the following job-specific options:
   - -t HH:MM:SS: set the maximum walltime per job. Default is 24h.
   - -n NCPU: run each job in an SMP parallelised fashion and set the number of CPU cores per job. Default is 1 core. For large active spaces, it is recommended to use several cores (e. g. 16 or 24, or as much as is available on a single node on your cluster).

4. Upon successful completion, check the output of the jobmanager.py script for any errors. It is also advisable to run the script named check_4rdm_size.sh to perform some error checks on the fourparticle.rdm.⟨state⟩.⟨state⟩ file.

5. If the checks are successful, copy the fourparticle.rdm.⟨state⟩.⟨state⟩ file back to the scratch directory of the DMRGSCF/MOTRA calculation.

6. Create an input file with the input to the &NEVPT2 module and run it.

6.3.3 Troubleshooting

The `jobmanager.py` script is experimental, and also batch jobs in queuing systems are prone to crash, therefore we provide a mechanism to identify and restart the crashed batch jobs. After `jobmanager.py` completion, it is advisable to run the script named `check_4rdm_size.sh` to check the 4-RDM file. If `check_4rdm_size.sh` yields errors, there are several options:

1. If the `jobmanager.py` finishes without errors, it will produce two files, `successlist` and `faillist` with the list of successful and failed batch jobs, respectively. In this case, the failed jobs may be restarted using the restart mode of `jobmanager.py`, which is invoked with
   `nohup jobmanager.py -r successlist faillist &`

2. If the `jobmanager.py` finishes with an error, the `successlist` and `faillist` will be either nonexistent or empty. Note that this does NOT necessarily mean that the jobs have failed: in our tests, certain configurations of the queuing system may lead to the crash of the `jobmanager.py` script after the successful completion of the jobs.

   In this case, the output of `check_4rdm_size.sh` provides the list of the failed jobs, which can be saved to the `faillist` file, and then `jobmanager.py` may be run in the restart mode (as described in p. 1) to restart the failed jobs.

6.3.4 Transition 3-RDM distributed calculations

`jobmanager.py` also supports distributed calculations of t-3RDMs (required for QD-NEVPT2). The split evaluation is similar to that of the 4-RDMs, and Section 6.3.2 can be followed with the following differences:

1. The t-3RDM evaluation requires two states instead of one. Run the `prepare_rdm_template.sh` script with the `-3` parameter.

2. Launch the `jobmanager.py` script with the `-3` parameter.

3. The files containing t-3RDMs are named `threeparticle.tdm.`⟨`state1`⟩`.`⟨`state2`⟩.

7 QCMaquis Standalone Version

Section 7.1 describes in details the installation process for a standalone version of QCMaquis.

> Note: If you have installed QCMaquis with the QCMaquis-OpenMolcas interface, you may skip this Section.

7.1 Installation

In the following steps 7.1.1-7.1.5 we describe how to successfully build and install the QCMaquis software suite in a standalone manner. The installation of QCMaquis has been tested for different operating systems and compiler/math libraries environments. Their list can be found in Section 3.2. 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`.

Additionally, the following libraries are required and need to be installed manually by the user (but are usually provided by the system package manager)

- GSL
- HDF5
- SZIP

7.1.1 Download QCMaquis

As mentioned in Section 2.2, QCMaquis standalone version may be downloaded from `https://scine.ethz.ch/download/qcmaquis`.

7.1.2 Setting up a build folder

Extract the downloaded archive `public.tar.bz2` into a new directory `my-QCMaquis-src`. 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
```

7.1.3 Configuration

QCMaquis supports operating systems, C++ compilers and math libraries as listed in Section 3.2. Below we will show the configuration steps for the most popular compiler suites GNU (Section 7.1.3.1) and Intel (Section **??**), respectively. How to setup and use a shared-memory OMP installation of QCMaquis is described in Section 7.1.3.2.

7.1.3.1 Configuration with the GNU compiler suite

To configure QCMaquis with the GNU compiler suite type

```
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.

> Note: Intel compiler is currently not supported. Running a binary compiled with Intel compilers may result in segmentation faults. We are currently investigating the issue.

7.1.3.2 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.

7.1.4 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.

7.1.5 Setting up the runtime environment

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

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

set up the runtime environment by running the following shell script

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

You might want to include the above line in your local `$HOME/.bashrc` to set up your environment automatically every time your session is started.

7.2 Keywords and Options

In the following we describe (i) compulsory keywords (Section 7.2.1), (ii) optional keywords (Section 7.2.2) for QCMaquis DMRG calculations as well as (iii) keywords for property calculations (Section 7.2.3). Most of the QCMaquis keywords have default settings that guarantee convergence in the general case and are inserted automatically by the host program (OpenMolcas in this case). A reasonable choice of default values for optional keywords is given in our example QCMaquis input file in Section 7.3.

> **Note:** Maquis has many features beyond quantum chemistry, e.g. related to solid state physics. Some keywords listed in the example file in Section 7.3 (e. g. `LATTICE`, `lattice_library` etc.) are therefore explained only briefly in the following and are not to be changed from the default values (see Listing 5), if a quantum chemical calculation is desired.

7.2.1 Compulsory keywords

The keywords in Tables 2 and 7 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 (i. e. whether 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.

Table 7: Compulsory QCMaquis keywords automatically set by the QCMaquis-OpenMolcas interface.

keyword	description
`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.
`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 [8, 14, 15] (as invoked by `dmrginit.py`, see Section 8.5) behaves like a restart from the newly generated CI-DEAS MPS.

`LATTICE`
`lattice_library`
`MODEL`
`model_library`
	These options set the Hamiltonian and the lattice type in QCMaquis calculations. For a quantum chemical calculation, these must be set to (see also Listing 5)
	`LATTICE = "orbitals"`
	`lattice_library = "coded"`
	`MODEL = "quantum_chemistry"`
	`model_library = "coded"`

Note: We strongly encourage any new user of QCMaquis to carefully read first Ref. 27 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. [8, 28–30].

7.2.2 Optional keywords

The keywords summarized in Table 8 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 8: 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 [8, 15] which can be obtained by means of the python script `fiedler.py` (see Section 8.5). **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 [8, 14, 15].
`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 8 – 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 7.2.1) 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.
`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 [8, 14, 15] 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.

Table 8 – continued from previous page

keyword	description
`storagedir`	Scratch directory for temporary files.

7.2.3 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 9 comprises a list of the available property keywords available in QCMaquis.

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]`.

If a property is available only in 2U1 symmetry, the SU2U1 checkpoint may be transformed into 2U1 group with the `mps_transform(_pg)` tool (see Section 8.4)

Table 9: 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. [31]) will be computed. 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]`
`MEASURE[3rdm]`
`MEASURE[4rdm]`
	Computes the two-particle, three-particle and four-particle reduced density matrix, respectively. In case of 4-RDM, `MEASURE[4rdm]="p4:p3:p1:p2@`l,k,i,j`"` allows to compute only a 4-index sub-block of the 4-RDM with the indices i,j,k,l. This functionality is used in the distributed RDM calculations described in Section 6.3. Analogously, for 3-RDM a two-index subblock may be calculated with `MEASURE[3rdm]="p1:p2@`j,i`"`. 3-RDM and 4-RDM calculations are available only in 2U1(pg) symmetry.

`MEASURE[trans2rdm]="`*chk*`"` `MEASURE[trans3rdm]="`*chk*`"`	Computes the transition two- or three-particle reduced density matrix with the current state and the state specified in the *chk* checkpoint file. For t-3RDM calculations, two-index subblocks can be calculated with `MEASURE[trans3rdm]="`*chk*`;p1:p2@`j,i`"`. Available only for 2U1(pg) symmetry.

Table 9 – continued from previous page

keyword	description
MEASURE_LOCAL[*name*]="*op*"	Computes $\langle\psi\|op_i\|\psi\rangle$, $i = 1 \ldots L$. 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 $\langle\psi\|op_{1i}op_{2j}\|\psi\rangle$, $i = 1 \ldots L, j = i+1 \ldots L$. 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.

7.3 QCMaquis input example

Listing 5 provides an input example for a DMRG-CASCI calculation. It assumes that an integral file FCIDUMP has been generated and placed in the same directory with the input file. The mandatory keywords are indicated in red, and the optional keywords in blue. The FCIDUMP for this particular example has been generated by the OpenMolcas-QCMaquis interface with the 001.input input file from the OpenMolcas-QCMaquis test directory.

Listing 5: Input example for a DMRG-CASCI(8,8) standalone QCMaquis calculation

```
 //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
//parameters 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]
```

8 Additional Considerations

8.1 Files required and written by QCMaquis

QCMaquis requires only two files to start a calculation:

1. an input file, see Listing 5 for example
2. an integral file in the `FCIDUMP` format as described in Ref. 16.

When used with the QCMaquis-OpenMolcas interface, both files are produced by the interface from the OpenMolcas input.

Besides standard output, QCMaquis produces two types of files:

1. a HDF5 *result file* in which all information on, e.g. the energies and expectation values is stored. The file name is specified as the `resultfile` option in QCMaquis input file. In QCMaquis-OpenMolcas calculations, the result files are named `$Project.result_state.`(state#)`.h5`, and are stored in the same directory as the OpenMolcas input file. Note that state numbering begins with 0.
2. a *checkpoint folder* which contains the matrix product state (MPS) wave function. The checkpoint folder name is controlled with the `chkpfile` option in QCMaquis input file, and In QCMaquis-OpenMolcas calculations, the checkpoint folders are named `$Project.checkpoint_state.`(state#)`.h5`. The checkpoint is required for a later restart of the calculation, while the tools and the analysis scripts described in Sections 8.4 and 8.5, respectively, require either a `result-file` or a `checkpoint-folder` or both.

8.2 Typical workflow for a standalone QCMaquis DMRG calculation

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

- prepare an FCIDUMP integral file using the OpenMolcas-QCMaquis host program driver [8]) 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 Listing 5 and adjust all molecular system and wave function specific parameters (for example `nelec, spin, irrep, L, ...`, see Sections 7.2.1-7.2.3 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 8.4 and 8.5, respectively, for a list of utility programs and scripts)

8.3 Memory management and memory requirements

Note that if you run a OpenMolcas-QCMaquis DMRG calculation, the memory consumption and memory allocation of QCMaquis is entirely *disconnected* from that of OpenMolcas. This means that the maximum memory specified within OpenMolcas (e. g. with the `MOLCAS_MEM` environment variable) has no influence on the QCMaquis memory usage. For DMRG-SCF calculations, the OpenMolcas memory usage may be kept low (i. e. at around 4 GB regardless of the size of the active space), since CI vector expansions (which grow roughly factorially with the number of electrons and N_{act}) need not be stored in memory anymore. On the other hand, the calculation of the reduced two-particle density matrix is quite memory-intensive with the current implementation: for a 20 orbital active space it currently requires ≈16 GB of memory, and an active space of 72 orbitals requires about 900 GB of memory. (Theoretically, the memory requirement scales as N_{act}^4, but with a large prefactor). Note that in a QCMaquis standalone run, the memory problem may be alleviated by running `dmrg_meas` on a different machine.

8.4 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 10: 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 [14, 15] 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. [8]. 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`.
`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 [32]. 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)`.

8.5 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 HDF5 result file as input. They are located in the folder `$MOLCAS/qcmaquis/lib/python/pyeval`.

Table 11: 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.
`dmrginit.py`	Starts a QCMaquis DMRG calculation with `max_bond_dimension = 200` and `nsweeps = 2`, measures the entropy information [31, 33] 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. 15. The current implementation is described in Ref. 8. 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 11 – continued from previous page

script	description
`mutinf.py`	Produces mutual information plots [31] 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`.

References

[1] Schollwöck, U. The Density-Matrix Renormalization Group in the Age of Matrix Product States. *Ann. Phys.* **2011**, *326*, 96–192.

[2] Marti, K. H.; Reiher, M. The Density Matrix Renormalization Group Algorithm in Quantum Chemistry. *Z. Phys. Chem.* **2010**, *224*, 583–599.

[3] Chan, G. K.-L.; Sharma, S. The Density Matrix Renormalization Group in Quantum Chemistry. *Annu. Rev. Phys. Chem.* **2011**, *62*, 465–481.

[4] Chan, G. K.-L.; Keselman, A.; Nakatani, N.; Li, Z.; White, S. R. Matrix Product Operators, Matrix Product States, and Ab Initio Density Matrix Renormalization Group Algorithms. *J. Chem. Phys.* **2016**, *145*, 014102.

[5] Keller, S.; Dolfi, M.; Troyer, M.; Reiher, M. An Efficient Matrix Product Operator Representation of the Quantum Chemical Hamiltonian. *J. Chem. Phys.* **2015**, *143*, 244118.

[6] Battaglia, S.; Keller, S.; Knecht, S. An Efficient Relativistic Density-Matrix Renormalization Group Implementation in a Matrix-Product Formulation. **2017**, arXiv:1710.08301.

[7] Keller, S.; Reiher, M. Spin-Adapted Matrix Product States and Operators. *J. Chem. Phys.* **2016**, *144*, 134101.

[8] Freitag, L.; Keller, S.; Knecht, S.; Lindh, R.; Ma, Y.; Stein, C. J.; Reiher, M. QC-Maquis in openMolcas: A Standardized Protocol for DMRG Calculations With a Low Barrier to Entry. in preparation.

[9] Aquilante, F. et al. Molcas 8: New Capabilities for Multiconfigurational Quantum Chemical Calculations across the Periodic Table. *J. Comput. Chem.* **2016**, *37*, 506–541.

[10] Knecht, S.; Keller, S.; Autschbach, J.; Reiher, M. A Nonorthogonal State-Interaction Approach for Matrix Product State Wave Functions. *J. Chem. Theory Comput.* **2016**, *12*, 5881–5894.

[11] Freitag, L.; Knecht, S.; Angeli, C.; Reiher, M. Multireference Perturbation Theory with Cholesky Decomposition for the Density Matrix Renormalization Group. *J. Chem. Theory Comput.* **2017**, *13*, 451–459.

[12] Fiedler, M. Algebraic Connectivity of Graphs. *Czech. Math. J.* **1973**, *23*, 298–305.

[13] Fiedler, M. A Property of Eigenvectors of Nonnegative Symmetric Matrices and its Application to Graph Theory. *Czech. Math. J.* **1975**, *25*, 619–633.

[14] Legeza, Ö.; Sólyom, J. Optimizing the Density-Matrix Renormalization Group Method Using Quantum Information Entropy. *Phys. Rev. B* **2003**, *68*, 195116.

[15] Barcza, G.; Legeza, Ö.; Marti, K. H.; Reiher, M. Quantum-Information Analysis of Electronic States of Different Molecular Structures. *Phys. Rev. A* **2011**, *83*, 012508.

[16] Knowles, P. J.; Handy, N. C. A Determinant Based Full Configuration Interaction Program. *Comput. Phys. Commun.* **1989**, *54*, 75–83.

[17] Knecht, S.; Keller, S.; Autschbach, J.; Reiher, M. A Nonorthogonal State-Interaction Approach for Matrix Product State Wave Functions. *J. Chem. Theory Comput.* **2016**, *12*, 5881–5894.

[18] Malmqvist, P.-A.; Roos, B. O. The CASSCF State Interaction Method. *Chem. Phys. Lett.* **1989**, *155*, 189–194.

[19] Malmqvist, P. A.; Roos, B. O.; Schimmelpfennig, B. The Restricted Active Space (RAS) State Interaction Approach with Spin–orbit Coupling. *Chem. Phys. Lett.* **2002**, *357*, 230–240.

[20] Malmqvist, P. Å. Calculation of Transition Density Matrices by Nonunitary Orbital Transformations. *Int. J. Quantum Chem.* **1986**, *30*, 479–494.

[21] Angeli, C.; Cimiraglia, R.; Evangelisti, S.; Leininger, T.; Malrieu, J.-P. Introduction of N-Electron Valence States for Multireference Perturbation Theory. *J. Chem. Phys.* **2001**, *114*, 10252.

[22] Angeli, C.; Cimiraglia, R.; Malrieu, J.-P. N-Electron Valence State Perturbation Theory: A Fast Implementation of the Strongly Contracted Variant. *Chem. Phys. Lett.* **2001**, *350*, 297–305.

[23] Angeli, C.; Cimiraglia, R.; Malrieu, J.-P. N-Electron Valence State Perturbation Theory: A Spinless Formulation and an Efficient Implementation of the Strongly Contracted and of the Partially Contracted Variants. *J. Chem. Phys.* **2002**, *117*, 9138–9153.

[24] Angeli, C.; Borini, S.; Cestari, M.; Cimiraglia, R. A Quasidegenerate Formulation of the Second Order N-Electron Valence State Perturbation Theory Approach. *J. Chem. Phys. 121*, 4043–4049.

[25] Dyall, K. G. The Choice of a Zeroth-order Hamiltonian for Second-order Perturbation Theory with a Complete Active Space Self-consistent-field Reference Function. *J. Chem. Phys.* **1995**, *102*, 4909–4918.

[26] Knecht, S.; Hedegård, E. D.; Keller, S.; Kovyrshin, A.; Ma, Y.; Muolo, A.; Stein, C. J.; Reiher, M. New Approaches for Ab Initio Calculations of Molecules with Strong Electron Correlation. *CHIMIA* **2016**, *70*, 244–251.

[27] Keller, S. F.; Reiher, M. Determining Factors for the Accuracy of DMRG in Chemistry. *CHIMIA* **2014**, *68*, 200–203.

[28] Freitag, L.; Knecht, S.; Keller, S. F.; Delcey, M. G.; Aquilante, F.; Pedersen, T. B.; Lindh, R.; Reiher, M.; González, L. Orbital entanglement and CASSCF analysis of the Ru–NO bond in a Ruthenium nitrosyl complex. *Phys. Chem. Chem. Phys.* **2015**, *17*, 14383–14392.

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

[30] Hedegård, E. D.; Knecht, S.; Kielberg, J. S.; Jensen, H. J. A.; Reiher, M. Density matrix renormalization group with efficient dynamical electron correlation through range separation. *J. Chem. Phys.* **2015**, *142*, 224108.

[31] Boguslawski, K.; Tecmer, P.; Legeza, Ö.; Reiher, M. Entanglement Measures for Single- and Multireference Correlation Effects. *J. Phys. Chem. Lett.* **2012**, *3*, 3129–3135.

[32] Moritz, G.; Reiher, M. Decomposition of Density Matrix Renormalization Group States into a Slater Determinant Basis. *J. Chem. Phys.* **2007**, *126*, 244109.

[33] Rissler, J.; Noack, R. M.; White, S. R. Measuring Orbital Interaction Using Quantum Information Theory. *Chem. Phys.* **2006**, *323*, 519–531.