QDYN

Quasi-DYNamic earthquake simulator

User's Manual

V 1.3.0

code.google.com/p/qdyn/

Developers

QuakeID Group, Caltech Seismo Lab:
Yingdi Luo (http://www.seismolab.caltech.edu/luo_y.html)
Jean-Paul Ampuero (http://www.seismolab.caltech.edu/ampuero_jp.html)
Bryan Riel (http://www.seismolab.caltech.edu/riel_b.html)

Last modified on 11/27/2014

[Click here](#) to access the most recent version

Table of contents

1. Introduction

1.1. Summary

QDYN is a boundary element software to simulate earthquake cycles (seismic and aseismic slip on tectonic faults) under the quasi-dynamic approximation (quasi-static elasticity combined with radiation damping) on faults governed by rate-and-state friction and embedded in elastic media.

QDYN includes various forms of rate-and-state friction and state evolution laws, and handles non-planar fault geometry in 3D and 2D media, as well as spring-block simulations. Loading is controlled by remote displacement, steady creep or oscillatory load. In 3D it handles free surface effects in a half-space, including normal stress coupling. The medium surrounding the fault is uniform, linear, isotropic and elastic.

QDYN implements adaptive time stepping, shared-memory parallelization, and can deal with multi-scale earthquake cycle simulations with fine details in both time and space. It is equipped with a user-friendly Matlab interface and graphical output utilities.

1.2. Main features

- Rate-and-state friction, with velocity cut-offs, aging and slip laws
- Arbitrarily heterogeneous frictional properties
- Slow and fast, aseismic and seismic slip transients (adaptive timestep)
- Non-planar faults (currently limited to variable dip, rectangular elements)
- 3D, 2D and 1D (spring-block)
- Steady and oscillatory loads
- Matlab wrapper and graphic output display utilities
- Parallelized for shared memory systems (OpenMP)
- Normal stress coupling

1.3. Documentation

Documentation for QDYN is available through the following resources:
- This User's Manual. Click here to access the most recent version.
- The `examples` directory contains several examples, some have a `README` file
- The Matlab tools provided with the QDYN package are documented

through Matlab's help. For instance `help qdyn` provides an overview of the usage of the `qdyn` Matlab interface.
- The `ToDo` file contains a list of known issues and features that we plan to implement in the future.

1.4. Support

Support for QDYN users (reporting bugs, installation problems, documentation issues, feature requests, etc) is available at:

https://code.google.com/p/qdyn/issues/list

Before submitting an issue please make sure that:
- you have read the QDYN documentation (see section 1.3)
- you are running the most recent version of QDYN (see sections 2.2 and 2.4)
- your problem has not been treated in previous issues. You can browse or search the issues database at https://code.google.com/p/qdyn/issues/list

New issues are submitted via https://code.google.com/p/qdyn/issues/entry. Please include all information needed to reproduce your problem (input files, operating system, compiler, etc).

1.5. License

This software is freely available for academic research purposes. If you use QDYN please include proper attributions to its authors and cite one of the references in section 1.7 in your scientific papers and reports.

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see http://www.gnu.org/licenses/.

1.6. Acknowledgements

1.6.1. Code contributions

QDYN outgrew from a 2D code written by Allan Rubin (Princeton University) in the early 2000s.

The subroutines implementing Okada's formulas were provided by Shinichi Miyazaki (Kyoto University).

The FFT subroutines are based on the [General Purpose FFT Package](#) written by Takuya Ooura (Kyoto University).

Martijn van den Ende (Utrecht University) contributed code fixes for Octave.

1.6.2. Funding

1.7. Suggested references

Y. Luo, J. P. Ampuero (2011), *Numerical Simulation of Tremor Migration Triggered by Slow Slip and Rapid Tremor Reversals*, AGU Fall Meeting 2011 Abstract S33C-02

Y. Luo, J. P. Ampuero (2012), *Simulation of Complex Tremor Migration Patterns*, AGU Fall Meeting 2012 Abstract S44B-02

2. Installation

2.1. Requirements

- [Make](#) and [Subversion](#) facilitate code installation and updates. Both are standard Linux tools.
- A Fortran compiler.
- Matlab or Octave.

We mostly develop and use QDYN with Linux, the Intel Fortran compiler (ifort) and Matlab. The code has been successfully compiled and used on Windows or Mac, with gfortran and Octave.

2.2. Download QDYN

QDYN is managed under the [Subversion](#) (SVN) version control system. To get the most current version of QDYN execute the following command on Linux (first time only):

```
svn checkout http://qdyn.googlecode.com/svn/trunk/  qdyn-read-only
```

This creates a directory `qdyn-read-only` which contains the whole QDYN package. You can create a directory with a different name.

You can also manually download the QDYN source files and pre-compiled executables for Windows or Mac OS directly from [this link](#). However, note that these may not be most up-to-date versions, and that pre-compiled executables may run slower than if you compile the code by yourself.

2.3. Install QDYN

1. Move to the `qdyn-read-only/src` directory
2. Modify the section "User Settings" of file `Makefile` following the instructions and examples therein:
    a. In section 1, set the executable file name and its path: set the variable `EXEC = [your executable file name, with its full path]`. By default, the executable file is named `~/bin/qdyn`. Make sure the path points to an existing directory. For instance, if the `~/bin/` directory does not exist, create it.
    b. In section 2, adjust your Fortran compiler settings: set the variables `F90 = [your compiler name]` and `OPT = [your compiler`

optimisation flags]. Intel Fortran (ifort) is the default compiler, but settings for several commonly used compilers are provided. Note that specific optimisation flags need to be set to enable parallelization through OpenMP (see section 4.2.1).

3. Modify the section "User Settings" in `constants.f90` following the instructions therein:
   a. Set the name of the kernel file including its full path (see section 3.2), a pre-calculated kernel file with `N=32768` is included at `qdyn-read-only/src/kernel_I_32768.tab` and should be sufficient for most simulations (this kernel file is only required while running 2D finite fault simulations)
   b. If you plan to use the option `FINITE=1` with `N>32768` (see section 3.2) update the name of the file `kernel*tab`
   c. If you plan to use Octave instead of Matlab set `OCTAVE_OUTPUT=.true.`
4. Run `make`
5. If in step 2 you changed the path or name of the executable file, modify accordingly the line `status = system('~/bin/qdyn')` of file `qdyn.m`

2.4. Update QDYN

After the first-time checkout, you can update the package by executing the following command in your `qdyn-read-only` directory:

```
svn update
```

Any source file that you have modified will be flagged as "conflicted" by SVN, and you will be prompted to select a conflict resolution method. You can preserve your modifications by selecting the option `mc` ("mine-conflict"). This is particularly useful to preserve your user settings in `Makefile` and `constants.f90` (otherwise you would need to repeat the steps in the "Install QDYN" section). For that purpose, the following command eliminates the interactive prompt:

```
svn update --accept mc
```

3. Running a simulation

3.1. The Matlab wrapper

The core of QDYN is a Fortran code. While the format of its input and output files is well defined, we find it more convenient to set up the input parameters,

perform simulations and read the output data within the Matlab environment through the wrapper function `qdyn.m`. You first need to set in Matlab the full path to the `src` directory, for instance:

```
addpath ~/qdyn-read-only/src
```

Tip for Mac users: if you get an error message related to gfortran libraries (e.g. `libgfortran.3.dylib`) when running `qdyn` in Matlab do:
```
setenv('DYLD_LIBRARY_PATH', '/usr/local/bin/')
```
The second argument should be the path to your gfortran libraries (sometimes `/opt/local/lib/libgcc`).

The general usage syntax is:

```
[pars,ot,ox] = qdyn(mode,[parsin],['Property',Value,...])
```

The default input values can be listed by executing:
```
pars = qdyn('set')
```

The input parameters are:

mode	One of the following execution modes:
	'set' Outputs the default parameter structure (*pars*) or overrides it with fields present in the structure *parsin* or with *Property/Value* pairs
	'write' Sets parameters and writes the qdyn input file
	'run' Sets parameters, writes the input file and runs a simulation
	'read' Reads parameters and outputs from a previous simulation
parsin	Parameter structure to override the default parameters (see section 3.2 for details)
'Property'	Name of a field to be set in the parameter structure (see section 3.2)
Value	Value to override the default value and the value in *parsin*

The output variables are:

pars	Structure containing the parameters (see section 3.2)
ot	Structure containing time series outputs, global or at selected points (see section 3.3)
ox	Structure containing snapshot outputs, i.e. quantities over the whole fault, output at selected times (see section 3.3)

3.2. Simulation parameters structure (pars)

The parameters in the structure *pars*, that can be set through 'parsin' or 'Prop/Value' pairs are:

Parameters defining the geometry of the problem and loading:

MESHDIM	Dimension of the problem:
	0 = Spring-block model
	1 = 1D fault in a 2D elastic medium
	2 = 2D fault in a 3D elastic medium
	4 = Same as 2 but fault stresses computed via 2D-FFT (works only if the grid spacings and dip angle are uniform)
MU	Shear modulus (Pa)
LAM	Elastic modulus lambda for 3D simulations (Pa)
VS	Shear wave speed (m/s). If *VS=0*, radiation damping is turned off
L	If *MESHDIM=1*, L is the fault length (or spatial period) If *MESHDIM=0*, *MU/L* is the spring stiffness
FINITE	Boundary conditions when *MESHDIM=1*
	0 = The fault is infinitely long but slip is spatially periodic with period L, loaded by steady displacement at distance W from the fault
	1 = The fault is infinitely long but only a segment of length

	L has rate-and-state friction, the rest has steady slip. If running the code with this option gives the error message "finite kernel is too small", you should create a larger "kernel file" with the matlab function *TabKernelFiniteFlt.m*, update the file name in `src/constants.f90`, and re-compile the code
W	Distance between displacement loading and fault, only if MESHDIM=1 and FINITE=0
DIP_W	Fault dip angle (degree) if MESHDIM=2 or 4. If depth-dependent, values must be given from deeper to shallower depth.
Z_CORNER	Fault bottom depth (m, negative down) if MESHDIM=2 or 4
SIGMA_CPL	Normal stress coupling: 0 = disable, 1 = enable
APER	Amplitude of additional time-dependent oscillatory shear stress loading (Pa)
TPER	Period of oscillatory loading (s)

Rate-and-state friction parameters:

A	Direct effect coefficient
B	Evolution effect coefficient
DC	Characteristic slip distance (m)
MU_SS	Reference steady-state friction coefficient
V_SS	Reference steady-state slip velocity (m/s)
TH_SS	Reference steady-state state (s). The default is *TH_SS=DC/V_SS*.
RNS_LAW	Type of rate-and-state friction law:
	0 = original
	1 = with cut-off velocities V1 and V2
V1	Cut-off velocity of direct effect (m/s)

V2	Cut-off velocity of evolution effect (m/s), controls the transition from weakening to strengthening when a<b. V2 should be ≤ V1.
THETA_LAW	Type of evolution law for the state variable:
	0 = ageing law in the "no-healing" approximation
	1 = ageing law
	2 = slip law

Initial conditions:

SIGMA	Initial effective normal stress (Pa). Remains constant unless *SIGMA_CPL =1*
V_0	Initial slip velocity (m/s)
TH_0	Initial state (s)

Discretization and accuracy parameters:

N	Number of fault elements if MESHDIM=1
NX	Number of fault elements along-strike, in 3D
NW	Number of fault elements along-dip, in 3D
DW	Along-dip length (m) of each element, from deep to shallow
TMAX	Total simulation time (s)
NSTOP	Stopping criterion
	0 = Stop at t=TMAX
	1 = Stop at end of slip localization phase
	2 = Stop at first slip rate peak
DTTRY	First trial timestep (s)
DTMAX	Maximum timestep (0=unrestricted)

ACC	Solver accuracy

Output control parameters:

OX_SEQ	Type of snapshot outputs
	
	      0 = All snapshots in a single output file (fort.19)
	
	      1 = One output file per snapshot (fort.1001, ...)

NXOUT	Spatial interval for snapshot outputs (in number of elements)
NTOUT	Temporal interval (in number of time steps) for snapshot outputs

OX_DYN	Output specific snapshots of dynamic events defined by thresholds on peak slip velocity DYN_TH_ON and DYN_TH_OFF (see below)
	
	      0 = Disable
	
	      1 = Enable outputs for event #i:
	          Event start: fort.19998+3i
	          Event end: fort.19999+3i
	          Rupture time: fort.20000+3i

NXOUT_DYN	Spatial interval (in number of elements) for dynamic snapshot outputs
DYN_TH_ON	Peak slip rate threshold (m/s) to define the beginning of a dynamic event
DYN_TH_OFF	Peak slip rate threshold (m/s) to define the end of a dynamic event
IC	Index of selected element for time series outputs
IOT	Indices of elements for additional time series outputs: set `IOT(i)=1` to enable time series outputs at the i-th element. By default, `IOT(i)=0` and this output is not done. Each element has a separate output file named `fort.xxxxx`, where `xxxxx` is an index (different than i) that starts at `10001` and is incremented by 1 for each selected element. For instance, if `IOT=[0 0 1 1]`, the output of elements `i=3` and

	`i=4` are in files `fort.10001` and `fort.10002`, respectively.
IASP	Auxiliary flags for elements (will not affect outputs, identification purpose only. e.g you can set elements of the VS part to -1 and VW to 0 and particular points of interests like asperities to numbers you want to use)

Parameters for integration with dynamic code:

DYN_FLAG	Integration with dynamic code
	
	       0 = Disable
	
	       1 = Enable: stop QDYN at the DYN_SKIP+1-th event with seismic moment > DYN_M

DYN_M	Target seismic moment of a dynamic event
DYN_SKIP	Number of dynamic events to skip (warm up cycles)

3.3. Output structures (ot, ox)

The outputs are:

pars	Structure containing the same fields as *parsin* (see above) plus the positions of the fault elements (X,Y,Z)

ot	Structure of time series outputs, with the following fields:
	
	    t Output times (s)
	
	    locl Localization length (distance between stressing rate maxima)
	
	    cl Crack length (distance between slip rate maxima)
	
	    p Seismic potency
	
	    pdot Seismic potency rate
	
	Outputs at the fault location with maximum slip rate:
	
	    xm Location of maximum slip rate

	v	Maximum slip rate
	th	State variable theta
	om	(slip rate)*theta/DC
	tau	Shear stress
	d	Slip
	Outputs at selected fault element with index IC:	
	vc	slip rate
	thc	state variable
	omc	(slip rate)*theta/DC
	tauc	shear stress
	dc	slip
ox	Structure of snapshot outputs, with the following fields:	
	x	fault coordinates
	t	output times
	v	slip rate
	th	state variable theta
	vd	slip acceleration
	dtau	shear stress relative to initial stress
	dtaud	shear stress rate
	d	slip
	sigma	effective normal stress

3.4. Examples

3.4.1. A simple 2D example

This example is in directory `examples/uniform_slip`. It's a 2D run with uniform slip and initial velocity slightly above steady state. In Matlab:

```
% get default parameters:
p = qdyn('set');
% reset some parameters:
p.N = 16; p.TMAX = 6e9; p.V_0=1.01*p.V_SS;
% run:
[p,ot,ox] = qdyn('run',p);
```

The estimated simulation time is shorter than 10 s on a single thread machine. Let's plot some outputs. Slip velocity as a function of time:
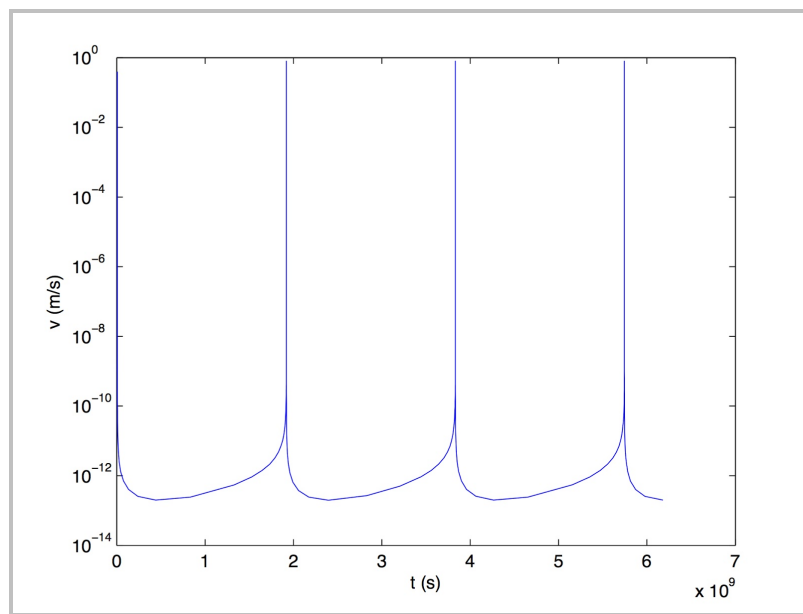
```
semilogy(ot.t,ot.v); xlabel('t (s)'); ylabel('v (m/s)')
```
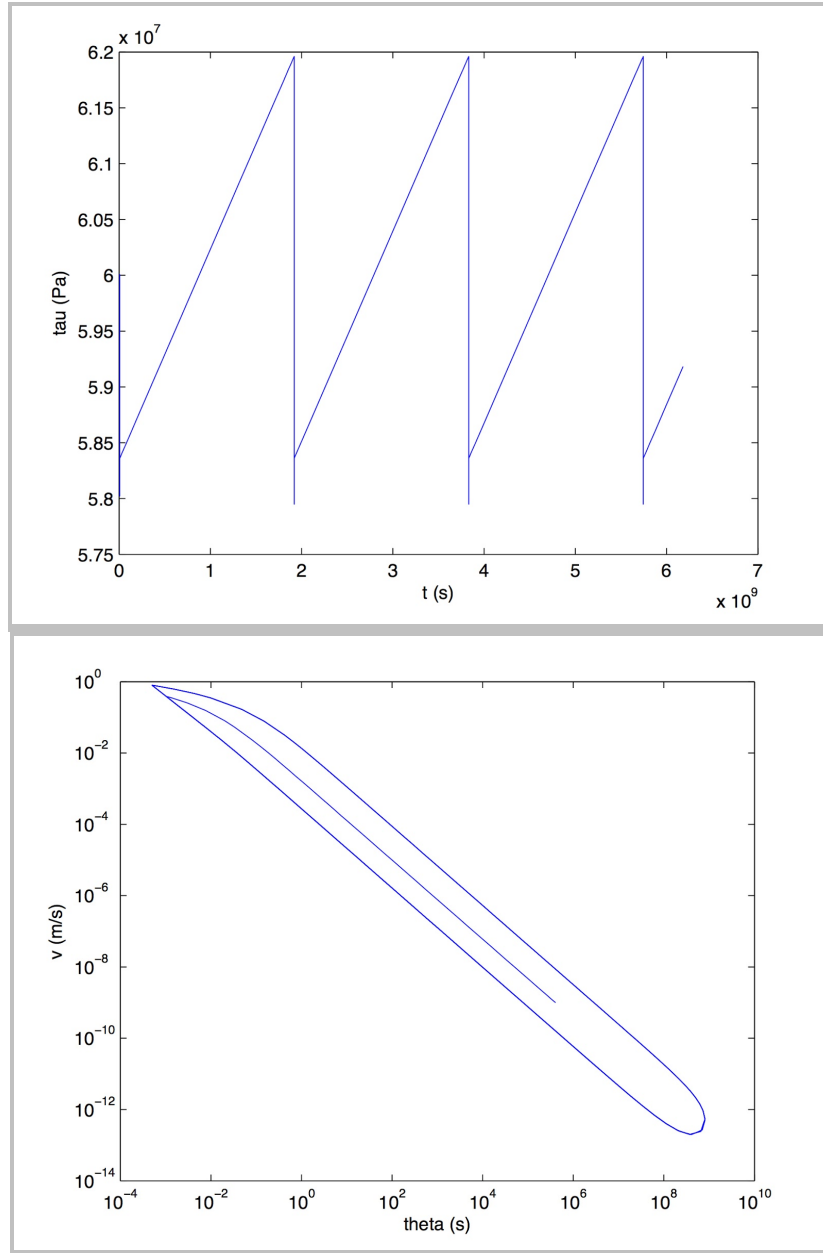
Plot shear stress as a function of time:

```
plot(ot.t,ot.tau); xlabel('t (s)'); ylabel('tau (Pa)')
```

Visualize the convergence to a limit cycle in a state-velocity plot:

```
loglog(ot.th,ot.v); xlabel('theta (s)'); ylabel('v (m/s)')
```

3.4.2. Two asperities interacting

This example is in directory `examples/double_asperities`. A velocity-weakening asperity interacts with a smaller asperity. Both are embedded in a velocity-strengthening (creeping) fault. When the large asperity breaks, its post-seismic slip propagates bi-laterally and triggers rupture of the small asperity. During the interseismic period of the large asperity, the smaller asperity breaks twice with a decreasing recurrence interval. The estimated simulation time is about 6 mins on a single thread machine.
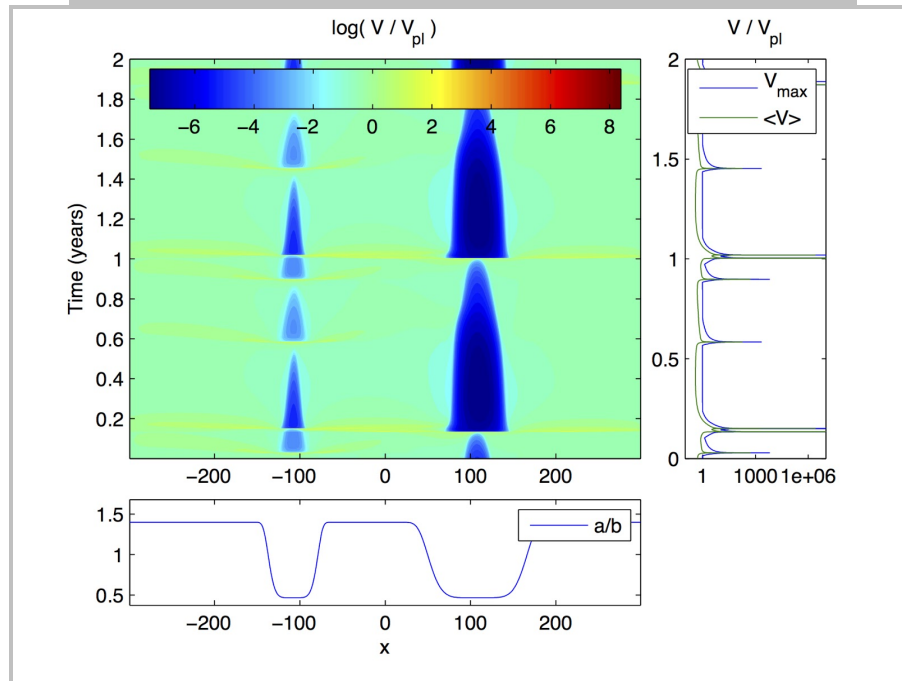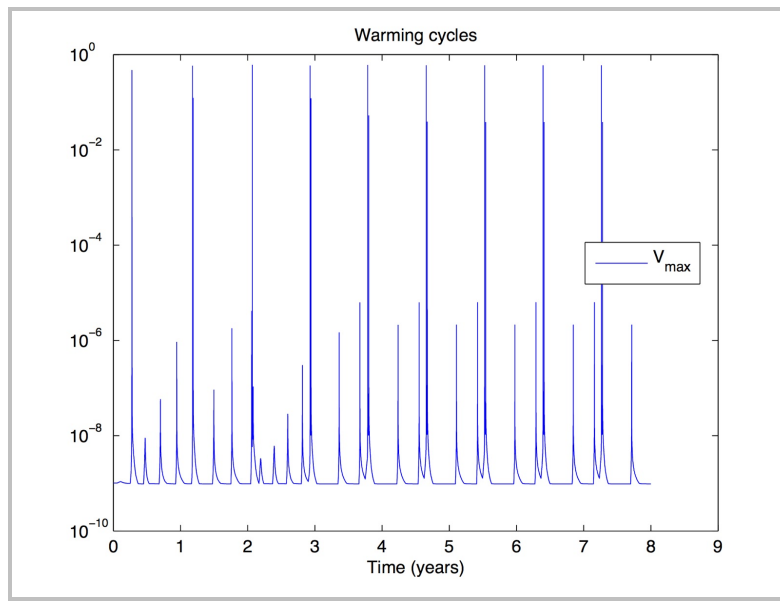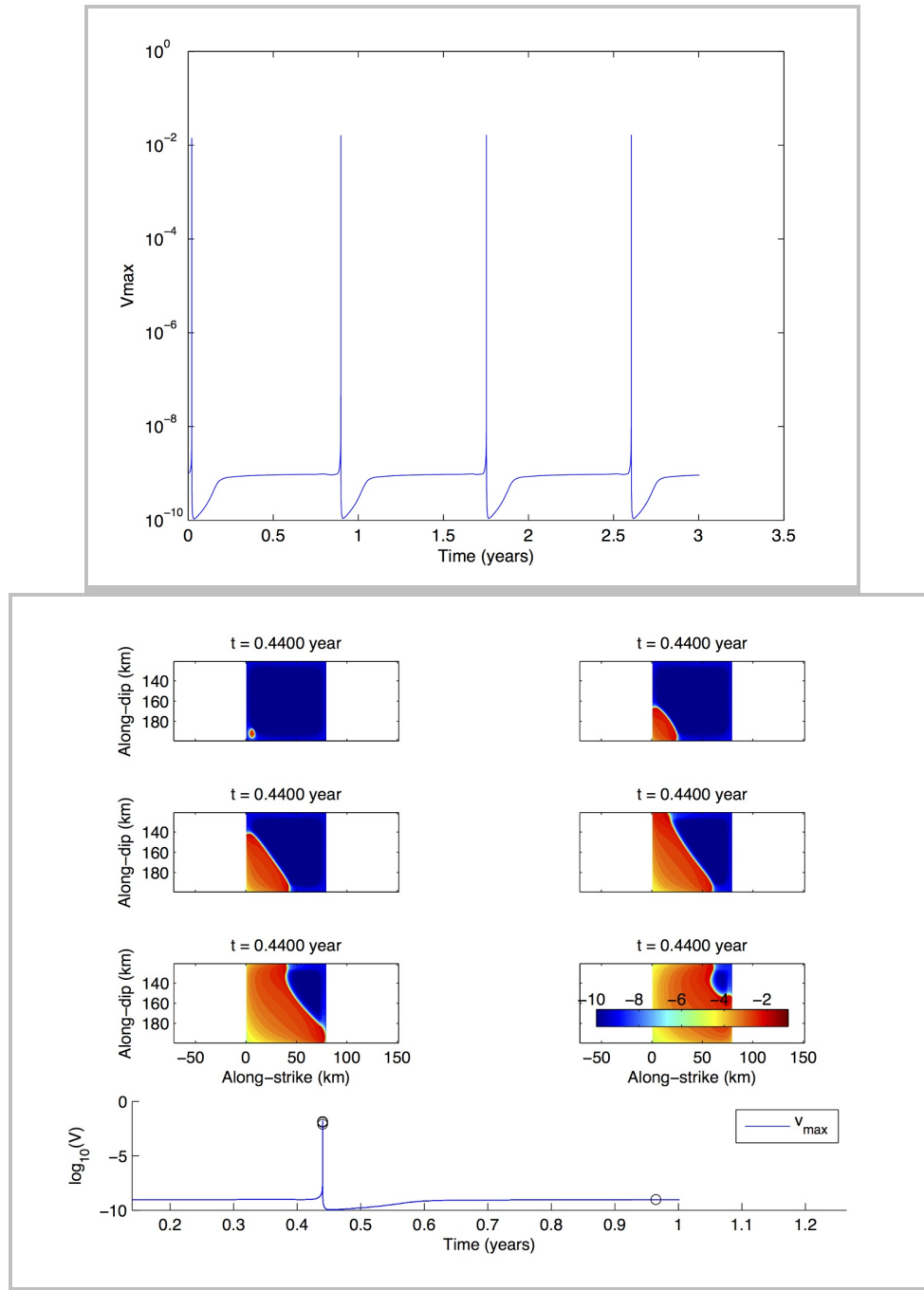
3.4.3. 3D simulations

An upper-layer Matlab wrapper for the base-layer Matlab wrapper qdyn.m is recommended for complicated simulations. We have included some examples for reference:

a) A 3D simulation: `src/test3dfft.m`
The estimated simulation time is about 10 mins on a single thread machine for the first earthquake cycle. The figures shown below are for a complete

multi-cycle simulation, comprising 4 warm-up cycles and one cycle output.

b) A simplified model for the Tohoku earthquake (2D along-dip and 3D simulations): `examples/Tohoku`

For more examples and real-world applications please refer to the wiki pages on the QDYN website.

4. Optimizing Performance

4.1. Running simulations outside the Matlab environment

Under certain circumstances, you may want to perform simulations outside the Matlab environment (e.g. while computing on a HPC). In this case, use Matlab wrapper separately first to generate an input file (qdyn.in), then run the qdyn executable directly.

4.2. Managing parallel computing

4.2.1. OpenMP

For 3D simulations on 2D faults (`MESHDIM=2` or `4`), QDYN is efficiently parallelized with OpenMP (shared memory parallelization). Before compiling the code, you should set the specific compiler optimisation flags that enable OpenMP, as described in the `Makefile` (see step 2.b in section 2.3). Before performing parallel simulations, you should set the following environment variable:

```
setenv OMP_NUM_THREADS 8
```

This command allows QDYN to run on 8 threads, which will roughly speed up calculations by a factor of 8. The number of threads should be set according to demand. In general, set this value to the maximum number of threads available on your shared memory system.

4.2.2. MPI
Features will be available in the next major update.

4.3. Managing outputs of large simulations

QDYN by default outputs results as a single ASCII file (fort.19) that may be too cumbersome to handle in large simulations. In practice, that is the case for most multi-cycle 3D simulations. It is then helpful to set `OX_SEQ = 1` when calling `qdyn.m`, to generate separate "ox" files outputs for each snapshot (fort.1001, ...). Also, setting `OX_DYN = 1` will automatically detect seismic events (according to parameters `DYN_TH_ON` and `DYN_TH_OFF`) and generate 3 snapshots for each event.

5. Visualizing simulation results

The QDYN software package includes several Matlab scripts to visualize simulation results in directory `src/`. These scripts are all self-documented:

plot_default.m	Plots slip rate of a 2D problem (along-strike)
plot2d_slip.m	Plots slip of a 2D problem (along-dip)
plot3d_m.m	Plots a sequence of snapshots of slip rate of a 3D simulation
plot3d_faultview3.m	Plots several snapshots of slip rate for 3D simulation in a single figure