BOXMOX extensions to KPP

Christoph Knote (christoph.knote@med.uni-augsburg.de)

Model-Based Environmental Exposure Science, Faculty of Medicine, University of Augsburg, Germany

Version 1.8

03/2022

Contents

Version history

1.8

not released yet

New features

- Move to GNU autotools for distribution

1.8a

not released

New features

- Implemented NO_x fixing method for steady state calculations

1.7

released December 2017

New features

- Method to include heteorogeneous reactions on aerosol
- Revamped mixing options (by constant K, volume, or tracer concentration)
- Simplified photoysis rates labeling
- New and renamed namelist options (`lverbose`, `iturb`)
- Restructured and updated documentation

Minor updates

- New command `list_BOXMOX_mechanisms`
- `new_BOXMOX_experiment` now copies default namelist to work directory

Internal changes

- Code cleanup
- *setup* file removed
- Creation of *VERSION* file to indicate BOXMOX version used for simulation

1.6

released February 2017

New features

- Ability to create adjoint mechanism using `prepare_BOXMOX_mechanism`

1.5

released December 2016

New features

- Simplified tooling and use scripts

- Bug fix in MOZART-4 mechanism

1.4

released September 2016

New features

- MCM v3.3, as well as several other mechanisms used in Knote et al., Atm. Env., 2015, are included
- Preparations to use adjoint models

1.3

released November 2015

New features

- Namelist input (runtime definitions, verbosity)
- Example cases for an urban plume, PBL diurnal cycle, and a chamber experiment (*boxmox/data* subdirectory)
- Diagnostic messages: notify user of unusable species in input files

Internal changes

- Code refactoring
- Preparations for using MCM in BOXMOX

1.2

released July 2015

Initial public release

1 The BOXMOX extension

A bunch of code that makes it possible to use KPP mechanisms in a box model fashion with the ability to read in parameters from simple text files.

1.1 Essentials of KPP

The Kinetic Preprocessor (KPP; Sandu and Sander, 2006) is a code generator that creates a numerical solver for a set of predefined (chemical) differential equations in a number of programming languages (Fortran, Matlab, C).

1.2 Essentials of BOXMOX

While KPP is already a box model, it is missing a "wrapper" that allows convenient input of initial and boundary conditions as well as environmental parameters, and it lacks processes typically associated with box model simulations of atmospheric chemistry. This is what the BOXMOX extensions provide.

Through a set of text files the user provides input for initial conditions, as well as for (time-varying) photolysis rates, environmental parameters, turbulent mixing, emissions and deposition.

Output of such a simulation are text files containing time series of the concentrations of all species in the mechanism, as well as all rate constants and the Jacobian.

BOXMOX was already used in a multi-mechanism comparison within the AQMEII Phase 2 project (Knote et al., 2015), please use this citation when referring to BOXMOX. A data assimilation toy model ('BEAT-BOX') has been built using BOXMOX in Knote et al. (2017).

1.3 References

Knote, C., Barré, J., and Eckl, M.: **BEATBOX: Background Error Analysis Testbed with Box Models**. *Geoscientific Model Development Discussions*, in review, doi:10.5194/gmd-2017-188, 2017.

Knote, C., Tuccella, P., Curci, G., Emmons, L., Orlando, J., Madronich, S., Baró, R., Jiménez-Guerrero, P., Luecken, D., Hogrefe, C., Forkel, R., Werhahn, J., Hirtl, M., Pérez, J.-L., San José, R., Giordano, L., Brunner, D., Yahya, K., and Zhang, Y.: **Influence of the choice of gas-phase mechanism on predictions of key gaseous pollutants during the AQMEII phase-2 intercomparison**. *Atmospheric Environment*, 115:553–568, doi:10.1016/j.atmosenv.2014.11.066, 2015.

Sandu, A., and Sander, R.: **Technical note: Simulating chemical systems in Fortran90 and Matlab with the Kinetic PreProcessor KPP-2.1**. *Atmospheric Chemistry and Physics*, 6(1):187–195, doi:10.5194/acp-6-187-2006, 2006.

2   Installation

BOXMOX is a Linux program, we assume you are working on a reasonably recent Linux distribution.

Requirements for installing and running BOXMOX:

- C compiler (usually `gcc` or `cc`)

- Fortran compiler (`gfortran` is assumed)

- flex (test with `which flex`)

- flex library (libfl.a, try `type -a libfl.a`)

- yacc or bison

BOXMOX installation is then straightforward:

```
./configure --prefix=<absolue path to where BOXMOX should be installed>
make
make install
```

Once you see the message "YOU ARE GOOD TO GO", everything should have worked out fine. Remember to set your environment variables (required) by copying the two lines starting with `export` and pasting in the terminal, or by adding them to your *.bashrc* / *.profile* files in your home directory (new login required to make it work). Contact me if you run into problems installing BOXMOX.

3   Preparing a BOXMOX simulation

This section assumes you have sucessfully installed BOXMOX, and intend to use a KPP mechanism description that has already been adapted for use with BOXMOX. Several such adapted mechanisms were installed with BOXMOX, please refer to section 7 on how to adapt other mechanisms.

3.1   Utility programs

Several commands have been installed with BOXMOX that make preparing box model simulations easy:

3.1.1   `list_BOXMOX_mechanisms`

Lists mechanisms ready for simulation, or (with `-a` switch) all mechanisms available including the ones not yet prepared.

3.1.2   `prepare_BOXMOX_mechanism`

To use a chemical mechanism in BOXMOX, you need to prepare (compile) it first. This has to be done only once for each mechanism. To prepare a mechanism, just call `prepare_BOXMOX_mechanism`:

```
me@mymachine> prepare_BOXMOX_mechanism MOZART_4
All necessary files and programs found.
Running KPP.
'Make'ing the model.
Successfully created MOZART_4.
me@mymachine>
```

This creates a box model executable for the chosen mechanism (in this case, MOZART-4) which you can now use for your experiments. The executables can be found in *$KPP_HOME/compiled_mechs/<mech>*.

3.1.3   `new_BOXMOX_experiment`

To make a box model simulation, you would copy the box model executable created in the last step into a newly created directory (and execute it). This is what the command `new_BOXMOX_experiment` is for:

```
me@mymachine> new_BOXMOX_experiment MOZART_4 new_experiment_folder
me@mymachine> ls new_experiment_folder/
BOXMOX.nml MOZART_4.exe VERSION
me@mymachine>
```

3.1.4   `new_BOXMOX_experiment_from_example` (alternatively)

A couple of ready-to-run experiments are distributed with BOXMOX, which can serve as starting point for further experiments. `new_BOXMOX_experiment_from_example` creates a new experiment based on these examples.

```
me@mymachine> new_BOXMOX_experiment_from_example chamber_experiment new_experiment_folder
me@mymachine> ls new_experiment_folder/
BOXMOX.nml InitialConditions.csv PhotolysisRates.csv VERSION
Environment.csv MOZART_4.exe README plot.R
me@mymachine>
```

3.2   Setting up a BOXMOX namelist

Parameters like runtime and verbosity can be controlled using a Namelist, a simple text file that resides in the same directory as the BOXMOX executable. It is a standard FORTRAN namelist, has to have the name *BOXMOX.nml*, and can contain the following variables:

`TSTART` is the start time of the run in seconds.

`TEND` is the end time of the run in seconds.

`DT` is the time step in seconds.

`lverbose` triggers (a lot) of additional screen output from a BOXMOX simulation.

`iturb` defines the type of (turbulent) mixing.

`lfix_nox` defines whether total NO_x should be fixed to its initial values.

A typical *BOXMOX.nml* file looks like this:

```
&BOXMOX
  tstart    =       0.0, ! start time in seconds
  tend      =    3600.0, ! end time in seconds
  dt        =      10.0, ! time step in seconds
  lverbose  =   .FALSE., ! verbose output?
  iturb     =         0, ! turbulent mixing option
  lfix_nox  =   .FALSE.  ! keep total NOx fixed (for steady state calculations)
/
```

3.3   Input file format

All input files described above are simple (ASCII) text files that need to be found in the same directory where the compiled mechanism (*<mech>.exe*) is executed. Their format needs to be readable through FORTRAN list-directed formatting. Exports from Excel as ".csv" (character separated values) as well as fixed-width formats should work. The general formatting rules are:

- values are separated by blanks or a comma
- values cannot contain blanks
- UNIX line endings (LF)

If an input file does not exist, BOXMOX will notify the user and continue. This is equal to not considering the process the file is associated with.

The file format convention is as follows:

```
I
I
I
(A) (A+) A+
(F) (F+) F+
```

with `I` integer, `A` string, `F` a real variable. `(...)` indicates optional values, and `...+` indicates possibility to have multiple values.

line 1: number of variables

line 2: number of ancillary variables (e.g. Kturb)

line 3: units of time in file

line 4: names of (time, ancillary and main) variables

line 5 ff: values

In case values provided are time-dependent (**line 3** is not 0), the first column (in **lines 4** and **5 ff**) must contain the time. In case the values do not depend on time, the time column must not exist. Ancillary variables are to be provided in columns before the main variables.

An example, time-dependent *PhotolysisRates.csv* file without ancillary variables could look like this (note that fixed-width is not necessary, comma-separated works as well):

```
3
0
2
        time          o2         o31d          o33p
  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
  1.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
  2.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
  3.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
  4.0000E+00  0.0000E+00  5.4040E-08  7.8720E-05
  5.0000E+00  0.0000E+00  1.0490E-06  3.2440E-04
  6.0000E+00  0.0000E+00  5.5420E-06  4.2150E-04
[... more ...]
```

3.3.1 Units of time

In **line 3**, input files define the type of time dependence the values have:

0: without time dependency, one line of values expected

1: seconds since simulation start

2: hour of day for an idealized diurnal cycle

3.3.2 Variable names

In **line 4**, variable names (labels) have to be defined. Different sets are used:

PhotolysisRates.csv: labels have to match the name used in the KPP rate constant (see 6.3).

Environment.csv: the variables described in 6.1 are available.

all other input files: labels have to match whatever species names you used in the KPP mechanism definition (*<mech>.spc* file).

The name of the time variable can be freely chosen, we suggest "`time`".

3.3.3 Variable units

The underlying KPP can work with whatever units the rate constants are defined in. In BOXMOX we assume molecules cm^{-3} for concentrations, and cm^3 molecules^{-1} s^{-1} for rate constants. This constraint is made to ensure that the model is consistent with input values for the boundary layer height and emissions. Input values of concentrations are given in parts per million per volume (ppmv). The other potential input variables are to be provided in the following units:

- concentrations: parts per million per volume (ppmv)

- emissions: molecules cm^{-2} s^{-1}

- Eddy diffusivity: m^3 s−1

- volume: m^3

- planetary boundary layer height: m

- temperature: K

- deposition velocities: cm s^{-1}

- photolysis rates: s^{-1}

4   Running a simulation

Once you have created a new experiment, adapted the namelist, added / adapted the input files needed for the processes you want to include, you can then run the box model:

```
me@mymachine>cd <path to experiment>
me@mymachine>./<mech>.exe


 ---------------------------------------------------------------
 BOXMOX (1.7) driver
 ---------------------------------------------------------------


 BOXMOX configuration file BOXMOX.nml found.


           TSTART             0.000
             TEND          3600.000
               DT            10.000


         lverbose                 F


            iturb                 0


 ---------------------------------------------------------------

  * Environment.csv
      useable:TEMP,PRES

  * InitialConditions.csv
      useable:CH2O,C2H6,CH4,H2O,CO,NO,NO2,O3,HNO3,SO2,NH3,PAN,H2O2,CH3OOH,CH3COOH,
[...]
      unknown: CO2,HCOOH,NROG,NOASN,XYLENES,ALKNIT,MBO,BENZENE,APIN,BPIN,LIMON,
[...]

  * Environment.csv
      useable:TEMP,PRES

  * PhotolysisRates.csv
      known:1_c4h9ono2,2_c4h9ono2,acrolein,biacet,c2cho,c2h5cho,c2h5ono2,
[...]

  10.0%. T=0.350E+03
[...]
 100.0%. T=0.360E+04


 ---------------------------------------------------------------
 BOXMOX successfully completed.
 ---------------------------------------------------------------


me@mymachine>
```

That is it.

5 Results and output

The model run creates a number of output files, all of them simple text files:

- $<mech>.conc$ - the concentration matrix

  In the first line the names of each column are given, then follows beginning at line two a data matrix with one more columns than your .spc file has species (first column is time in hours), and one row per time step.

- $<mech>.rates$ - the rate constant matrix

  The number of rate constants (nrates) in your mechanism in the first line, followed by nrates lines with the name of each reaction, followed by a data matrix with one more columns than your mechanism has rate constants (first column is time in hours), and one row per time step.

- $<mech>.jacobian$ - the Jacobian (dx / dy) matrix

- $<mech>.hessian$ - the Hessian (d^2x / (dy dz)) matrix, if `HESSIAN` is set to ON in $<mech>.kpp$

6 Processes and their representation in BOXMOX

The following section describes the processes that are added to a KPP box model by the BOXMOX extensions. Each process is connected to an input file, and their file names are mentioned in each section. Omission of an input file is equal to ignoring the associated process.

6.1 Environmental parameters

Environmental parameters define properties of the box and is characteristics. These are provided through file *Environment.csv*.

Available variables are:

- `TEMP`: temperature in Kelvin
- `PRES`: pressure in Pa
- `VOL`: box volume in m^3
- (`PBLH`: planetary boundary layer height in m)

The variable `PBLH` is an alternative to describe a change in the vertical extent (hence, the volume) of the box.

When temperature and pressure are provided as time-dependent variables, this will cause assumed air density to vary accordingly. Values provided in any input file that require conversion using density (e.g., initial concentrations, background values) will be converted using the time-varying air density accordingly.

6.2 Initial conditions

Values for initial conditions of all species used in the mechanism can be provided through the file *Initial-Conditions.csv*. All species not defined in *InitialConditions.csv* are set to a tiny, positive value for numerical stability (variable `ALL_SPEC` in *boxmox/wrapper*).

6.3 Photolysis rates

Photolysis rates can either be provided in the .eqn file as fixed values or values that depend on the KPP variable SUN (see KPP documentation).

Additionally, photolysis rates can be provided through the file *PhotolysisRates.csv*. In this case, use J(''label'') as reaction rate function, with `label` corresponding to a column in the file *PhotolysisRates.csv*.

Example:

In the .eqn file of your mechanism you add the following photolysis rates:

```
[...]
A + B = C : J("AB");
C + D = E : J("CD_ch1");
C + D = F : J("CD_ch2");
[...]
```

Then the corresponding contents of *PhotolysisRates.csv* would be:

```
3
0
0
AB CD_ch1 CD_ch2
```

```
1e-3 5e-4 2e-4
```

6.4   Heterogeneous surface reactions

A combination of input file and rate constant variable allows to consider heterogeneous reactions that depend on aerosol. The variables `AER_SAD` (aerosol surface area in $m^3\ m^{-2}$) and `AER_R` (aerosol effective radius in m) are provided for rate constants in the *<mech>.spc* file. Their values can be set by creating a new input file *Aerosol.csv*, for which labels `SAD` and `R` are known. The BOXMOX example het_chem and the mechanism HETCHEM shows how to use this feature.

6.5   Turbulent mixing

The box can exchange with its surroundings via mixing (due to turbulent motions), a process also referred to as dilution or en-/detrainment. The convective term $(\vec{v}\nabla c)$ of the continuity equation for the concentration of a substance c

$$\frac{\partial c}{\partial t} + \vec{v}\nabla c = 0 \tag{1}$$

is simplified to an exchange between the box and a background concentration c_{bg}, scaled by a coefficient:

$$K_{turb}(c - c_{bg}) := \vec{v}\nabla c \tag{2}$$

The coefficient K_{turb} is equal to the Eddy diffusivity, or turbulent diffusion coefficient, if only turbulent exchange is considered. The new concentration of c at the end of time step Δt is then calculated as

$$c(t = t_0 + \Delta t) = c(t = t_0) + K_{turb}(c(t = t_0) - c_{bg}(t = t_0)) \tag{3}$$

Three different options are implemented to describe K_{turb}, and thereby mixing. They are specified using the namelist variable `iturb` and through the provision of background concentrations (and `Kturb`) in *Background.csv*.

6.5.1   Specified K_{turb} (iturb=1)

Using values for the eddy diffusivity K_{turb} specified as ancillary variable `Kturb` in *Background.csv*.

6.5.2   Box volume changes (iturb=2)

Using the changes in the volume of the box as defined through the `VOL` variable in *Environment.csv*. Increases in box volume are matched by mixing assuming the change in box volume is due to the influx of background air, decreases in box volume are ignored.

Hint: simulating *top de-/entrainment* due to a time-varying planetary boundary layer height is a special case of mixing due to volume changes of a box with a constant horizontal extent.

6.5.3   Following a tracer concentration (iturb=3)

Using the change in concentration of a tracer known to the chemical mechanism. The model attempts to match the box concentration of a species (tracer) with the tracer concentrations specified as ancillary variable in *Background.csv* through mixing with the background.

6.6 Emissions

For all species emission fluxes can be provided in *Emissions.csv*. It is assumed that these mix instantaneously within the box. Realistic emission fluxes (e.g. from 3-D models) require a realistic mixing volume, hence it is only reasonable to provide emissions in combination with a realistic `PBLH` / `VOL` setting in *Environment.csv*. Details on how to set up a correct input file can be found in section 3.3.

6.7 Deposition

A first-order loss process, analogous to "dry deposition", can be included. Values are provided as deposition velocities $v_{dep,i}$ for a species i. The change in concentration C_i over time t is calculated as

$$\frac{\partial C}{\partial t} = -v_{dep,i} \cdot C_i(t = t_0)/dz \tag{4}$$

with dz the vertical extent of the box (`PBLH`). Values for deposition are provided through file *Deposition.csv*. Also for deposition, a realistic value for the boundary layer height `PBLH` should be set.

7 Adding and modifying mechanisms

Making a BOXMOX simulation itself consists of only one step - running the BOXMOX executable in a directory in which you provided all input files you needed. Change/add/remove input files as you like and simply re-run the executable to make a new simulation. Output files will be overwritten.

It might, however, be of interest to make changes to the underlying mechanism, or even create a new mechanism. In the following it is shown how to modify an existing mechanism and even create a new one.

7.1 Modifying an existing mechanism

The set of equations, species and further definitions for an existing mechanism <mech> can be found in your KPP/BOXMOX installation directory:

`${KPP_HOME}/models/<mech>.(eqn|spc|def)`

The KPP definitions (integration scheme, creation of Jacobian, . . .) for a mechanism are in:

`${KPP_HOME}/examples/<mech>.kpp`

To make changes to this mechanism, edit the files as needed, and then recreate the mechanism for use in BOXMOX using `prepare_BOXMOX_mechanism -f <mech>`. Note usage of the `-f` flag to force recreation of the mechanism even though it has already been prepared. You can then make simulations with the new mechanism as it has been described above.

7.2 Adding new mechanism

To add a new KPP mechanism for use in BOXMOX, you need to create the files mentioned in the previous subsection (7.1). It might be the easiest to use one of the simple existing mechanisms as blueprint. BOXMOX mechanisms are KPP mechanisms (see KPP documentation) with slight adaptations to make them useable in BOXMOX:

7.2.1 Edit the *$KPP_HOME/examples/<mech>.kpp* file

Make sure the following variables are set:

```
#MODEL <mech>
#LANGUAGE Fortran90
#DOUBLE ON
#INTEGRATOR rosenbrock
#DRIVER boxmox
#JACOBIAN SPARSE_LU_ROW
#HESSIAN OFF
```

Additional lines might exist, and can be used with BOXMOX unchanged most of time. To get the Hessian matrix (e.g. for adjoint runs), set `HESSIAN` to 'ON'.

7.2.2 Edit the *$KPP_HOME/models/<mech>.def* file

The following has to be included in the .def file, at the top of the file before any `#INCLUDE` directives:

```
#include ../boxmox/wrapper

#LOOKATALL
```

7.2.3 Preparing the new mechanism

Once you created the 4 files (*$KPP\_HOME/examples/<mech>.kpp*, *$KPP_HOME/models/<mech>.(def|spc|eqn)*) you can prepare the mechanism with `prepare_BOXMOX_mechanism <mech>`.