DynareOBC: A toolkit for handling occasionally binding constraints with Dynare, by Tom Holden.

Background

Please read the paper here: https://github.com/tholden/dynareOBC/raw/master/paper.pdf

Or read the slides here: https://github.com/tholden/dynareOBC/raw/master/slides.pdf

Installation

First, either download or clone the toolkit, by pressing the download or clone button on:
https://github.com/tholden/dynareOBC

Then, add the toolkit to your MATLAB path. DynareOBC updates itself and its dependencies each time it is run.

Requirements

Requirements (to be installed and added to your Matlab path):

- Matlab version R2013a or later, or a fully compatible clone. Note that while DynareOBC should work on all platforms, it has
  been most heavily tested on 64-bit Windows, so if possible we suggest you use this platform.
- dynare, version 4.4 or later, from: http://www.dynare.org/download/dynare-stable

Recommended additional installations:

- MATLAB R2015a or later.
- The MATLAB Parallel toolbox, or a fully compatible clone.
- The MATLAB Optimization toolbox, or an alternative non-linear least squares routine, which is required for the experimental
  `global` option. (To use an alternative routine, you must set `dynareOBC.FSolveFunctor`.)
- A working compiler for MEX which is supported by MATLAB Coder, ideally supporting OpenMP. On Windows, a free compiler meeting
  these requirements is available from: https://www.visualstudio.com/en-us/news/vs2013-community-vs.aspx . Alternatively, on
  Windows, with MATLAB r2015b, another free compiler meeting these requirements (which uses much less disk space) is available

by clicking on "Add-Ons" in the MATLAB toolbar, then searching for MinGW. Be sure to untick the "check for updated files" in the installer that opens.

- MATLAB Coder, or a fully compatible clone (only used with MATLAB R2015a or later).
- A competitive mixed integer linear programming solver, such as one of the below, each of which is available for free to academics:
  - FICO Xpress, from https://community.fico.com/download.jspa
  - GUROBI Optimizer, from http://www.gurobi.com/academia/for-universities
  - MOSEK, from https://mosek.com/resources/academic-license and https://mosek.com/resources/downloads
  - IBM CPLEX, by following the instructions here: https://www.ibm.com/developerworks/community/blogs/jfp/entry/cplex_studio_in_ibm_academic_initiative?lang=en

Troubleshooting

If you have any strange errors, first try these steps:

1. Delete all `.mat` files from the DynareOBC directory.
2. Delete all `.mex???` files from the `dynareOBC` sub-directory of your DynareOBC install (but not from its sub-directories).
3. Follow the following steps to do a manual update:
   - Open a Git shell (e.g. by clicking on the gear icon from within the GitHub application).
   - Navigate to the directory in which you installed DynareOBC.
   - Type `git fetch --all --recurse-submodules` then press return.
   - Type `git reset --hard origin/master` then press return.
   - Type `git submodule foreach --recursive git deinit --force` then press return.
   - Type `git submodule foreach --recursive git submodule update --init` then press return.

If after doing this, you suspect a bug, report it here: https://github.com/tholden/dynareOBC/issues

Basic Usage

Usage: `dynareOBC FILENAME[.mod,.dyn] [OPTIONS]`

DynareOBC executes instruction included in the conventional dynare mod file, `FILENAME.mod`.

Unlike dynare, DynareOBC can handle simulation of models containing non-differentiable functions.
Note:

- DynareOBC approximates the non-differentiable function in levels. Thus if r and pi are the endogenous variables of interest:
  `r = max( 0, 0.005 + 1.5 * pi );` will be more accurate than: `exp(r) = max( 1, exp( 0.005 + 1.5 * pi ) );`.
- DynareOBC may produce strange results on models with an indeterminate steady-state, so caution should be taken when using the `STEADY_STATE` command. The initval or steady_state_model blocks should not be used to attempt to pin down a steady-state,
  since these will be ignored by DynareOBC in later steps of its solution procedure.
- DynareOBC defines the preprocessor constant `dynareOBC` during execution.

OPTIONS (NOT CASE SENSITIVE!) include:

- **For controlling the inner solution procedure**
  - `TimeToSolveParametrically=INTEGER` (default: `4`)
    If the simulation is at the bound for at most this number of periods, then a pre-computed solution will be used,
    increasing the speed of long simulations, or those involving integration.
  - `TimeToEscapeBounds=INTEGER` (default: `32`)
    The number of periods following a shock after which the model is expected to be away from any occasionally binding
    constraints. Note that when a global solution is requested, this value is ignored, and the maximum of the requested
    number of IRF periods, and the value of TimeToReturnToSteadyState, is used instead.
  - `TimeToReturnToSteadyState=INTEGER` (default: `64`)
    The number of periods in which to verify that the constraints are not being violated. If this is lower than
    TimeToEscapeBounds, or the requested number of IRF periods, or PeriodsOfUncertainty + 1, then that value will be used
    instead.
  - `Omega=FLOAT` (default: `1000`)
    The tightness of the constraint on the news shocks. If this is large, solutions with news shocks close to zero will be
    returned when there are multiple solutions.
  - `MILPSolver=STRING` (default: automatically selected based on the detected solvers)
    DynareOBC uses YALMIP internally for solving a mixed integer linear programming problem. This option sets YALMIP's
    solver. To find out what solvers are available to you, run `dynareOBC TestSolvers`, and examine the list displayed by
    YALMIP.

  - `FullHorizon`
    By default, DynareOBC finds a solution for which the last period at the bound is as soon as possible. This option
    makes DynareOBC just solve the bounds problem at the longest horizon (i.e. `TimeToEscapeBounds`).
  - `IgnoreBoundFailures`
    Makes DynareOBC atttempt to continue even after it has failed to solve the bounds problem due to e.g. infeasability.
    This will severely compromise accuracy.
- **For controlling cubature**
  - `PeriodsOfUncertainty=INTEGER` (default: `16`)
    Controls the number of periods of uncertainty over which we integrate. Since a cosine windowing function is used, the
    effective number of periods of uncertainty is roughly half this number.
  - `FastCubature`
    Causes DynareOBC to ignore the value specified in `MaxCubatureDegree` and `QuasiMonteCarlo`, and to instead use a degree 3
    monomial cubature rule without negative weights, but involving evaluations further from the origin.
  - `QuasiMonteCarloLevel=INTEGER` (default: `0`)
    If this is non-zero, then Gaussian cubature is not used (so the MaxCubatureDegree option is ignored). Instead, quasi-
    Monte Carlo integration with at most `2^(1+INTEGER) - 1` samples is used.
  - `MaxCubatureDegree=INTEGER` (default: `7`)
    Specifies the degree of polynomial which will be integrated exactly in the highest degree, cubature performed. Values
    above `51` are treated as equal to `51`. Note that setting `CubatureSmoothing>0` will reduce the effective maximum accuracy
    by 2 degrees. Setting `CubatureTolerance>0` may also mean that the result does not integrate the stated degree polynomials
    exactly as well.
    - `CubatureSmoothing=FLOAT_IN_UNIT_INTERVAL` (default: `0`)
      When this is larger than 0, and less than 1, DynareOBC takes a weighted combination of the results of cubature rules
      of adjacent degrees. Large numbers imply larger weights on lower degree rules. A good setting is often 0.01 - 0.3.
  - `CubaturePruningCutOff=FLOAT` (default: `0.01`)
    Eigenvalues of the covariance matrix of the distribution from which we integrate that are below `FLOAT` times the maximum
    eigenvalue are "pruned" to zero, in order to increase integration speed.
  - `MaxCubatureDimension=INTEGER` (default: `128`)
    The maximum dimension over which to integrate. If the algorithm needs to integrate over a larger space, it will "prune" all
    but the INTEGER largest eigenvalues of the covariance matrix to

zero.

- `CubatureTolerance=FLOAT` (default: `1e-6`)
  Specifies that the maximum acceptable change in the integrals is the given value, for quasi Monte Carlo or cubature.
  Setting this to zero disables adaptive cubature.
- `NoCubature`
  Speeds up DynareOBC by assuming that agents are "surprised" by the existence of the bound. At `order=1`, this is equivalent to a perfect foresight solution to the model.

- **For controlling accuracy**
  - `FirstOrderAroundRSS`
    Takes a linear approximation around the risky steady state of the non-linear model. If specifying this option, you should set `order=2` or `order=3` in your mod file.
  - `FirstOrderAroundMean`
    Takes a linear approximation around the ergodic mean of the non-linear model. If specifying this option, you should set `order=2` or `order=3` in your mod file.
  - `FirstOrderConditionalCovariance`
    When `order>1` (possibly with `FirstOrderAroundRSS` or `FirstOrderAroundMean`), by default, DynareOBC uses a second order approximation of the conditional covariance. This option specifies that a first order approximation should be used instead.
  - `MLVSimulationMode=0|1|2|3` (default: `0`)
    If `MLVSimulationMode=0`, DynareOBC does not attempt to simulate the path of model local variables.
    If `MLVSimulationMode>0`, DynareOBC generates simulated paths and average impulse responses for each model local variable (MLV) which is used in the model, non-constant, non-forward looking, and not purely backwards looking. Note that to generate impulse responses, you must enable the SlowIRFs option.
    If `MLVSimulationMode>1`, DynareOBC additionally generates simulated paths and average impulse responses for each non-constant MLV, used in the model, containing forward looking terms.
    If `MLVSimulationMode=2`, then DynareOBC takes the expectation of each forward looking MLV using sparse cubature.
    If `MLVSimulationMode=3`, then DynareOBC takes the expectation of each forward looking MLV using quasi-Monte Carlo integration.
    - `MLVSimulationAccuracy=INTEGER` (default: `9`)
      When `MLVSimulationMode=2`, this specifies the degree of polynomial which should be integrated exactly. In this case, values above 51 are treated as equal to 51. When `MLVSimulationMode=3`, `2^(1+INTEGER) - 1` is the number

of points
used for quasi-Monte Carlo integration.
    * `MLVSimulationSubSample=INTEGER` (default: `1`)
    Causes DynareOBC to only calculate the value of MLVs every `INTEGER` samples. Setting this greater than 1 is useful
    when calculating MLVs is expensive, and you want to reduce the standard error of simulated moments.
  - `Sparse`
  Causes DynareOBC to replace all of the elements of the decision rules by sparse matrices, which may speed up DynareOBC,
  at the cost of some slight reduction in accuracy for certain models with small coefficients.
- **For controlling and performing model diagnostics**
  - `FeasibilityTestGridSize=INTEGER` (default: `0`)
  Specifies the number of points in each of the two axes of the grid on which a test of a sufficient condition for feasibility
  is performed. Setting a larger number increases the chance of finding feasibility, but may be slow.
  If `FeasibilityTestGridSize=0` then the test is disabled.
  - `FullTest=INTEGER` (default: `0`)
  Runs very slow tests to see if the top `INTEGERxINTEGER` submatrix of M is a P and/or (strictly) semi-monotone matrix.
  - `PTest=INTEGER` (default: `0`)
  Tests if the top `INTEGERxINTEGER` submatirx of M is a P matrix. Set this to 0 to disable these tests.
  - `LPSolver=STRING` (default: automatically selected based on the detected solvers)
  Specifies the solver to use for the linear programming problem that is solved when checking whether matrices are S/S_0.
  To find out what solvers are available to you, run `dynareOBC TestSolvers`, and examine the list displayed by YALMIP.
- **For controlling IRFs**
  - `SlowIRFs`
  Calculates a more accurate approximation to expected IRFs using Monte-Carlo simulation. Without this option, DynareOBC
  calculates expected IRFs via cubature (unless this is also disabled).
  - `IRFsAroundZero`
  By default, IRFs are centered around the risky steady state with the fastirfs option, or around the approximate mean
  without it. This option instead centers IRFs around 0.
  - `ShockScale=FLOAT` (default: `1`)
  Scale of shocks for IRFs. This allows the calculation of IRFs to shocks larger or smaller than one standard deviation.
- **EXPERIMENTAL settings for controlling accuracy**
  - `Global`
  Without this, DynareOBC assumes agents realise that shocks may

arrive in the near future which push them towards the bound, but they do not take into account the risk of hitting the bound in the far future. With the global option, DynareOBC assumes agents take into account the risk of hitting the bound at all horizons. Note that under the global solution algorithm, dotted lines give the responses with the polynomial approximation to the bound. They are not the response ignoring the bound entirely.
Requires the MATLAB Optimization toolbox, or an alternative non-linear least squares routine, see above for details.

  - `GlobalConstraintStrength=FLOAT` (default: `1`)
    Specifies the weight on the squared deviation of the equality constraint in the global objective problem. Larger values imply that the long-run path is very close to the short-run one, but can produce excess volatility in simulations.
  - `GlobalViolationStrength=FLOAT` (default: `1`)
    Specifies the weight on the squared violations of the inequality constraints in the global objective problem. Larger values reduce violations, but can produce bias or excess volatility in simulations.
  - `Resume`
    Resumes an interrupted solution iteration, when using global.
  - `QPSolver=STRING` (default: automatically selected based on the detected solvers)
    Specifies the solver to use for the quadratic programming problem that is solved by the global algorithm. To find out what solvers are available to you, run `dynareOBC TestSolvers`, and examine the list displayed by YALMIP.

- **EXPERIMENTAL settings for controlling estimation**
  - `Estimation`
    Enables estimation of the model's parameters
    - `EstimationDataFile=STRING` (default: `MOD-FILE-NAME.xlsx`)
      Specifies the spreadsheet containing the data to estimate. This spreadsheet should contain two worksheets. The first sheet should have a title row containing the names of the MLVs being observed, followed by one row per observation. There should not be a column with dates. The second sheet should contain a title row with the names of the parameters being estimated, followed by one row for their minima (with empty cells being interpreted as minus infinity), then by one row for their maxima (with empty cells being interpreted as plus infinity).
    - `EstimationFixedPointMaxIterations=INTEGER` (default: `100`)
      The maximum number of iterations used to evaluate the stationary distribution of the non-linear filter.
    - `EstimationAlternativeCubature`

Uses an alternative cubature rule for integrating over the states and shocks of the model, which includes an
additional central point. While this requires solving the model less far from the steady-state, it also requires
a negative weight, which may cause numerical issues with the positive definiteness of the state covariance matrix.

- **Advanced options**
  - `CompileSimulationCode`
    Compiles the code used for simulating the base model, without the bound. May speed up long simulations.
  - `OrderOverride=1|2|3`
    Overrides the order of approximation set within the call to stoch_simul.
  - `Bypass`
    Ignores all non-differentiabilities in the model. Useful for debugging.
  - `NoCleanup`
    Prevents the deletion of DynareOBC's temporary files. Useful for debugging.

See the Dynare reference manual for other available options.

Supported options inside the .mod file

Note that DynareOBC only supports some of the options of `stoch_simul`, and no warning is generated if it is used with an
unsupported option. Currently supported options for `stoch_simul` are:

- `irf=INTEGER`
- `periods=INTEGER`
- `drop=INTEGER`
- `order=1|2|3`
- `replic=INTEGER`
- `loglinear`
- `irf_shocks`
- `nograph`
- `nodisplay`
- `nomoments`
- `nocorr`

DynareOBC also supports a list of variables for simulation after the call to `stoch_simul`. When `MLVSimulationMode>0`, this list
can include the names of model local variables. Any MLV included in this list will be simulated even if it does not meet the
previous criteria.

Advanced Usage

Alternative usage: `dynareOBC [AddPath|RmPath|TestSolvers] [OPTIONS]`

- `AddPath`
  Adds all folders used by DynareOBC to the path, useful for debugging and testing.
- `RmPath`
  Removes all folders used by DynareOBC from the path, useful for cleanup following a crash.
- `TestSolvers`
  Tests the installed solvers.

Acknowledgements and copyright information

DynareOBC incorporates code:

- for calculating risky first order approximations that is copyright Meyer-Gohde, 2014.
  More information is contained in his paper describing the algorithm, available here:
  http://enim.wiwi.hu-berlin.de/vwl/wtm2/mitarbeiter/meyer-gohde/stochss_main.pdf
- from the nonlinear moving average toolkit that is copyright Lan and Alexander Meyer-Gohde, 2014,
- for nested Gaussian cubature that is copyright Genz and Keister, 1996,
- for LDL decompositions that is copyright Borchers, 2002,
- for displaying a progress bar that is copyright Cacho, "Stefan" and Scheff, 2014,
- for calculating relative paths that is copyright Lenz and Chatfield, 2013,
- for (mixed-integer) linear programming, from GLPKMEX, copyright Makhorin, Legat and others, 2015,
- for quadratic programming, from qpOASES, copyright Ferreau, Kirches, Potschka, Bock, Diehl, 2014,
- for semi-definite programming, from the SeDuMi solver, copyright Sturm, Terlaky, Polik and Pomanko, 2014,
- for calculating pseudo-spectral radii, from EigTool, copyright Wright, Mengi, Overton and colleagues, 2014,
- for interacting with Git, from JGit4MATLAB, copyright Mikofski and Glauche, 2014.

Additionally, DynareOBC automatically downloads:

- YALMIP, copyright Lofberg, 2015,
- the Opti Toolbox, copyright Currie, and others, 2015,
- QPC, copyright Wills, 2009,
- and MPT, with its dependencies, copyright Herceg and others, 2015.