Contents

1 EST-NLSS: A toolkit for performing non-linear state space estimation with an Extended Skew T (EST) approximation to the state distribution.

1.1 Background

A paper documenting the estimation procedure is available from here: https://github.com/tholden/EST-NLSS/raw/master/EstimationPaper.pdf
(Or here for citations: http://dx.doi.org/10.5281/zenodo.50127)

1.2 Installation

1. Download the source ZIP from: https://github.com/tholden/EST-NLSS/releases
2. Extract the code to a directory on a local drive.
3. Add just the folder containing `RunEstimation.m` to your MATLAB path. Do not click with add-with sub-folders!

1.3 Requirements

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

- MATLAB version R2016b or later, or a fully compatible clone. MATLAB version R2017a or later is strongly recommended.
- MATLAB Coder, or a fully compatible clone.
- The MATLAB Statistics and Machine Learning Toolox, or a fully compatible clone.
- A working compiler for MEX which is supported by MATLAB Coder is required. 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, 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.
- For estimation with any of the included optimisation wrappers, the MATLAB Parallel Toolbox (or a fully compatible clone) is required.
- For estimation using the included `fmincon` wrapper, the MATLAB Optimization Toolbox (or a fully compatible clone) is required.

1.4 Basic Usage

Usage:

```
[ EstimatedParameters, EstimatedParameterCovarianceMatrix, PersistentState
] = RunEstimation( Parameters, Options, PersistentState );
```

Inputs:

- `Parameters`
  A column vector of initial parameters.
- `Options`
  A struct including some or all of the following fields:
  - `AllowTailEvaluations` (default: `false`)
    Determines whether it is safe to use a potentially more efficient integration method which evaluates further into the tails of the distribution.
  - `CompileLikelihood` (default: `false`)
    Specifies whether to attempt to compile the likelihood. If true, this requires MATLAB version R2017a or later and MATLAB Coder.
  - `Data` (default: `[]`)
    A matrix with the data on which to estimate. Columns are observations, rows are variables.
  - `DynamicNu` (default: `false`)
    Causes the estimation procedure to calibrate the degrees of freedom parameter, nu, at each time step. We recommend that `FilterCubatureDegree` is at least 9 if this option is specified.
  - `ExoCovariance` (default: `[]`)
    The fixed covariance matrix of the source exogenous shocks. Usually setting this to the identity matrix is fine, and the shocks can be scaled (and correlations can be introduced) within your `Simulate` functor.
  - `FilterCubatureDegree` (default: `0`)
    If this is greater than zero, then EST-NLSS uses an alternative sparse cubature rule including additional points for integrating over the states and shocks of the model in the filter. While this requires solving the model less far from the steady-state when the state dimension is large, it also requires negative weights, which may cause numerical issues e.g. with the positive definiteness of the state covariance matrix. this cubature method exactly integrates a polynomial of degree `FilterCubatureDegree`. Values above `51` are treated as equal to `51`.
    If this is less than zero, then EST-NLSS takes `2.^(-FilterCubatureDegree)` points from a high order Sobol sequence.
  - `InitialMEStd` (default: `0.0001`)
    Either a scalar or a vector giving the initial standard deviation of the measurement error. If this is a scalar, the same standard deviation will be used for all measurement errors.
  - `InitialNu` (default: `20`)
    The initial value of `nu`, if it is being estimated (i.e., if `DynamicNu` is `false`).
  - `LB` (default: `[]`)
    The lower bound on the parameter vector.
  - `UB` (default: `[]`)
    The upper bound on the parameter vector.
  - `MaximisationFunctions` (default: `{ @CMAESWrapper, @FMinConWrapper }`)
    A cell array of maximisation function handles or strings containing maximisation function names, which will be invoked in order. EST-NLSS includes the following:
    * `CMAESWrapper` an evolutionary global search algorithm,
    * `CMAESResumeWrapper` an evolutionary global search algorithm, resuming an interrupted CMAES run,
    * `ACDWrapper` an adaptive coordinate descent algorithm,
    * `ACDResumeWrapper` an adaptive coordinate descent algorithm, resuming an interrupted ACD run,
    * `PACDWrapper` an alternative adaptive coordinate descent algorithm,
    * `PACDResumeWrapper` an alternative adaptive coordinate descent algorithm, resuming an interrupted ACD run,
    * `FMinConWrapper` MATLAB's local search, which requires a license for the MATLAB Optimisation Toolbox.
      If you choose to build your own maximisation function wrapper, it should have the signature:
      `[ x, f, PersistentState ] = Wrapper( OptiFunction, x, lb, ub,`

    PersistentState, varargin),
    where:
    * OptiFunction is the objective to be maximised, which has the signature f = OptiFunction(x, PersistentState).
    * x passes in the initial x and passes out the final (optimal) x.
    * f passes out the value of OptiFunction at the final point.
    * PersistentState passes in the initial persistent state, and passes out the persistent state at the final point.
    * lb is the lower bound on x.
    * ub is the upper bound on x.
    * varargin is a cell array of other inputs which may be used for e.g. setting additional options.
- NoSkewLikelihood (default: false)
  If set to true, this disables the skewing of the distribution used to approximate the likelihood.
- NoTLikelihood (default: false)
  If set to true, this disables the use of a (extended skew) t-distribution to approximate the likelihood. Instead a (extended skew) normal distribution will be used.
- ParameterNames (default: {})
  A cell array of names for the parameters, for clearer output.
- MeasurementVariableNames (default: {})
  A cell array of names for the measurement variables, for clearer output.
- Prior (default: @FlatPrior)
  A function handle or string containing a function name that specifies the prior to use for maximum a posteriori estimation. The inbuilt @FlatPrior gives a flat prior, so results in maximum likelihood estimation.
  If you choose to specify your own prior, it should have the signature LogPriorDensity = Prior(FullParameters) where FullParameters is a vector containing the model's parameters, followed by the measurement error standard deviations, and then possibly by nu-bar (if DynamicNu is not specified), and where LogPriorDensity should return the log prior density at that point (up to a constant).
- Simulate (default: @(Parameters, PersistentState, InitialStates, ShockSequence, t) [])
  A function handle or string containing a function name that specifies the function to use for simulating the model. The specified function should have the signature:
  [PersistentState, EndoSimulation, MeasurementSimulation] = Simulate(Parameters, PersistentState, InitialStates, ShockSequence, ~)
  where:
    * Parameters is a vector containing the model's current parameters.
    * PersistentState is the current PersistentState which can be a MATLAB variable of any kind. PersistentState may be used for e.g. storing the solution, or storing the solution of some slowly changing internal optimisation problem.
    * InitialStates is either an empty array, or a matrix of state vectors from which simulation steps should be performed. Rows of InitialStates give variables, and columns give observations. If InitialStates is empty, then the simulation call should return in EndoSimulation a large number of draws from the distribution of the endogenous variables at time 0 (where time 1 is the first observation), of length given by the number of columns of ShockSequence. For stationary models, the easiest way to do this is to return a time-series simulation, starting from some arbitrary point (ideally, either a draw from the ergodic distribution, or the final period of a previous simulation, saved to PersistentState).
    * ShockSequence is a matrix of pseudo-draws from a Gaussian distribution with covariance given by the set ExoCovariance. Rows of ShockSequence give particular shocks, and columns give observations. The simulation function should not use any other source of randomness.
    * t is a non-negative integer giving the current time period, ranging from 0 to the number of columns of the input Data.
    * EndoSimulation returns the generated simulation of the model's endogenous variables

    (i.e. all of the model's states, and any controls of interest). Rows of `EndoSimulation` give variables and columns give observations. `EndoSimulation` should have as many columns as `ShockSequence`.
    * `MeasurementSimulation` returns a matrix of perfect measurements given the simulation. Rows of `MeasurementSimulation` give variables and columns give observations. Note that this output will not always be requested, so it is a good idea to only generate it if `nargin > 2`.
  - `Solve` (default: `@( Parameters, PersistentState ) []`)
    A function handle or string containing a function name that specifies the function to use for solving the model. The specified function should have the signature:
    `[ PersistentState, StateSteadyState, StateVariableIndices ] = Solve( Parameters, PersistentState )`
    where:
    * `Parameters` is a vector containing the model's current parameters.
    * `PersistentState` is the current `PersistentState` which can be a MATLAB variable of any kind. `PersistentState` may be used for storing the solution.
    * `StateSteadyState` should contain a vector containing the steady-state of the state variables.
    * `StateVariableIndices` is a vector of indices specifying which of the variables returned by the `Simulate` functor in `EndoSimulation` are state variables. It should have the same length as `StateSteadyState`.
  - `SkipStandardErrors` (default: `false`)
    Makes EST-NLSS skip calculation of standard errors for the estimated parameters.
  - `StationaryDistAccuracy` (default: `10`)
    `2 ^ StationaryDistAccuracy - 1` periods will be used to evaluate the stationary distribution of the model.
  - `StationaryDisDrop` (default: `0`)
    Specifies the number of points dropped from the start of the simulation in computing the stationary distribution of the model.
  - `StdDevThreshold` (default: `1e-6`)
    Specifies the threshold below which the standard deviation of the state is set to zero, for dimension reduction.
- `PersistentState`
  A MATLAB variable of any kind that should be passed to the solution and simulation steps. New evaluations of the likelihood will be passed the `PersistentState` from the recent best evaluation.

Outputs:

- `EstimatedParameters`
  The full vector of estimated parameters, containing the model's parameters, followed by the measurement error standard deviations, and then possibly by `nu-bar` (if `DynamicNu` is not specified).
- `EstimatedParameterCovarianceMatrix`
  The covariance matrix of the parameter estimates, if it was calculated (i.e. if `SkipStandardErrors` was set to `false`).
- `PersistentState`
  The persistent state at the optimal point.

1.5 Acknowledgements and copyright information

EST-NLSS incorporates code:

- for nested Gaussian cubature, that is copyright Genz and Keister, 1996,
- for finding the nearest symmetric positive definite matrix, that is copyright D'Errico, 2013,
- for adaptive coordinate descent, that is copyright Loshchilov, Schoenauer, Sebag, 2012,
- for evolutionary optimisation with covariance matrix adaptation, that is copyright Hansen, 2012,
- for the implementation of special functions over a complex domain, that is copyright Godfrey, Acklam, 2004,
- for the implementation of the incomplete Beta function, that is copyright McElwaine, 2010,