Basics of MIRT-OC

Massimo De Mauri

February 22, 2021

Contents

1 Introduction

MIRT-OC is a Python implementation of the namesake algorithm for Mixed-Integer Convex Model Predictive Control developed by De Mauri et. al (add citation). The software relies on the C++ package for symbolic programming and automatic differentiation CasADi (cite) and the Julia Mixed-Integer Convex Solver OpenBB.jl (cite).

Draft

2 Installation

Draft

3 Quick Reference Guide

3.1 The input Class

An object of the input class represents an external signal that can be taken into account in the optimization. The default constructor of the input class accepts the name of the input to be defined. In order to use the input in the problem formulation a symbolic reference to it can be obtained by invoking the field: .sym.

3.2 The variable Class

An object of the variable class represents a decision variable in the problem. It can be constructed specifying, in order, the name of the variable, its lower-bound and its upper-bound. In order to use the variable in the problem formulation a symbolic reference to it can be obtained by invoking the field: .sym.

3.3 The constraint Class

An object of the constraint class represents a constraints in the optimal control problem. The default constructor of the constraint class accepts as inputs, in order, a CasADi expression for the body of the constraint, two values for the lower and upper bound of the constraint. There exists three shorthand constructors for the class: 1) eq(expression,rhs) which generates an equality constraints (lower and upper bounds are equal); 2) leq(expression,rhs) which generates a less-or-equal constraint (only the upper bound is defined) 3) geq(expresson,rhs) which generates a greater-or-equal constraint (only the lower-bound is defined)[1].

3.4 The OCmodel Class

In MIRT-OC the optimal control problem to be solved is defined via the construction of an object of the class: OCmodel. The default constructor of OCmodel takes as input a string that constitutes the model name. Upon creation a OCmodel object represents an empty model, in order to define an optimal control problem the user has to fill the fields of the newly created object.

3.4.1 Fields

- **.t and .dt, Symbols for time and delta-time:** The .t and .dt fields are automatically filled by the constructor. They contain two CasADi symbols representing, respectively, the initial point and the size of the current time step [2].
- **.i, External Inputs:** The field expects a list of objects of the class input. The external inputs are signals that the MPC process takes as input at each iteration. The inputs may be used to represent the changes in the environment the optimization has to take into account, reference quantities (such as desired state at each time step) as well as modifiers for the constraints/consts of the optimal control problem.
- **.x, Continuous Transition States:** The field expects a list of objects of the class variable. The states are the main aspect of the problem that evolves over time. The evolution of their value is governed by the expressions in the field .ode via numerical integration.
- **.y, Instantaneous Transition States:** The field expects a list of objects of the class variable. Instantaneous Transition States differ from Continuous Transition States as their value evolution is directly defined by the expressions in the field .itr and not via integration. In practice, it is assumed that such states can change instantaneously value at the beginning of each time step in the optimal control problem, similarly to states of a Finite-State Machine.
- **.a, Algebraic Variables:** The field expects a list of objects of the class variable. The algebraic variables are free decision variables in the in the optimal control problem. They can be used for a variety of purposes as for example serving as slack variables for the definition of otherwise non-smooth/non-linear constraints (e.g. L1 norms) or to lift non-linear objectives in order to obtain a mixed-integer non-linearly constrained linear problem (which is usually better accepted by MINLP solvers). Once the control problem is transformend into an optimization problem via Multiple Shooting a copy of each of the variables in .a is created for each time step, so that its variable at each time step is independent from the values at other time steps.
- **.v, Continuous Controls:** The field expects a list of objects of the class variable. Continuous controls are free decision variables that are used to influence the evolution of the states. They can take any value within the user-provided bounds.
- **.w, Discrete Controls:** The field expects a list of objects of the class variable. Discrete controls are controls that are constrained to take value on the subset of the integer numbers defined by the user-provided bounds.
- **.ode, Ordinary Differential Equations:** The field expects a list of CasADi expressions depending on the symbolic references of any subset of the inputs/variables defined above. The i-th expression in the list corresponds to the right hand side of the equation: $\dot{x}_i = f_i(\dots)$, where x_i is the i-th state defined in .x.

[1] The constructors "eq", "leq" and "geq" accept also symbolic expressions in the right hand side. Providing a symbolic expression in the rhs causes the constraint to be internally reformulated as an equivalent constraint having numerical values upper and lower bounds.

[2] The signal represented by the symbol in .t is piecewise constant, so it is not accurate in between time steps. In order to get a more accurate timer it is possible to define a continuous state with derivative equal to one.

- **.itr, Instantaneous Transition States:** The field expects a list of CasADi expressions depending on the symbolic references of any subset of the inputs/variables defined above. The i-th expression in the list corresponds to the right hand side of the equation: $x_{i,k} = g_i(\dots)$, where $x_{i,k}$ is the value i-th state defined in .y at time t_k and g_i depends on the values of the problem variables at time t_{k-1}.
- **.pcns, Path Constraints:** The field expects a list of objects of the class constraint where the provided expressions depend on the symbolic references of any subset of the inputs/variables defined above. The constraint are evaluated at the beginning of each time step, i.e., on the value of the variables at time t_k for $k \in \{0, 1, \dots, N\}$.
- **.lag, Lagrangian Cost:** The field expects a CasADi expression depending on any subset of the inputs/variables defined above. The Lagrangian Cost is the right hand side of the equation: $\dot{c} = l(...)$ where c is a cost signal (to be minimized) associated with optimal control problem.
- **.ipn, Instantaneous Penalty:** The field expects a CasADi expression depending on any subset of the inputs/variables defined above. The Instantaneous Penalty differs from the Lagrangian Cost for the fact that it is evaluated only at the beginning of each time step and constitutes a spot addition to the cost associated with the control problem.
- **.may, Mayer Cost:** The field expects a CasADi expression depending on any subset of the inputs/variables defined above not containing controls. The Mayer Cost is an Instantaneous Penalty applied only to terminal state of the control problem (or MPC subproblem).

3.4.2 Methods

- **.__str__():** The method provides a high level description of the model. It is invoked automatically with print(model).
- **.epigraph_reformulation(max_orders=[2,2,1],integration_options=None):** max_order collects the maximum polynomial orders for the cost terms in the following order: Lagrange Cost, Discrete Penalty, Mayer Cost. The objective terms of polynomial order higer than max_orders[k] are reformulated as constraints thanks to the creation of an additional algebraic variable. The integration options are used in case of the reformulation of a Lagrange cost term: in this case the term is integrated symbolically before being transormed into a constraint, a description of the allowed values for this option can be found in Section 4.1..

3.5 The MPCcontroller Class

Objects of the MPCcontroller class, contains all the information necessary for obtaining the MPC control actions.

3.5.1 The Constructor

The constructor of MPCcontroller accepts as input an object of the class OCmodel and a dictionary of options. Allowed options are:

- print_level, ({0,1}, Def. 1): Controls the verbosity of the algorithm, 0 meaning "no-output" and 1 meaning "verbose".
- integration_opts, (Def.{}): Passes the options to the numerical integrator used in Multiple Shooting a description of the allowed values for this option can be found in Section 4.1.
- prediction_horizon_length, ($\geq$ 1, Def. 1): Number of time steps in the future considered at each MPC iteration.
- relaxed_tail_length, ($\geq$ 0, Def. 0): Number of relaxed (ignoring discrete requirements) time steps to be added to the end of the prediction horizon as heuristic cost (like in Approximate Dynamic Programming).
- HBB_opts, (Def. {}): Options to be passed to the Hybrid-Branch&Bound algorithm in OpenBB.jl. More details about such options in Section 4.2.

3.5.2 The .iterate(shift_size,measured_state,input_vals,...) method

If enough computation time is allowed, the method returns a the optimal control strategy for the considered prediction horizon plus relaxed tail. The first input, shift_size, communicates to the method how many time steps forward the MPC controller has to shift the subproblem for the current iteration. The second input, measured_state, represents the up-to-date measurement of the state of the system. The third input, input_vals, contains the up-to-date value for the external inputs in the problem[3].

Additionally the method accepts two optional inputs:

- mode, (Def. 'ConstraintsOnly'): For this option there are four allowed values: 1) 'fullRTI', the linearizations generated at each time steps are propagated to the next along with the whole B&B tree; 2) 'hotstart', linearization propagation plus propagation of the B&B nodes of the tree which lies within a distance of 1 in L1 norm from the global optimal solution of the previos MPC subproblem[4]; 3)'constraintsOnly', only the linearization are propagated and the B&B process is restarted form scratch; 4) None, no information is propagated between MPC iterations, i.e., each MPC subproblem is solved as a stand-alone MINLP.
- iterations_for_lob_recomputation, (Def. -1): During the tree propagation, OpenBB.jl recomputes valid lower bounds for the nodes in the tree using dual information. Optionally some iterations of the MI(L/Q)P solver can be run to improve such bounds. Setting the parameter to -1 tells OpenBB to completely resolve each propagated node, 0 provokes OpenBB.jl to rely solely on the dual information stored on the node and $n \in \mathbb{N}^+$ represents the number of iterations of the subsolver to be run.
- timeout, (Def. 0.0): maximum time allowed for the solution of the next MPC subproblem, 0.0 stays for no-limit and $T \in \mathbb{R}^+$ represents the maximum time in seconds.

4 Extra References

4.1 Integration options

In the following, the possible options for the numerical integrator used in MIRT-OC are presented and briefly explained:

- schema, (Def. 'rk4'): This setting represents which integration scheme to use for numerical integration. Possible values are:
  - 'e_euler': Explicit Euler.
  - 'i_euler': Implicit Euler.
  - 'rk4': fourth-order Runge-Kutta.
  - 'gl2','gl4','gl6': Gauss-Legendre of order 2, 4 and 6, respectively.
  - a matrix representing a Butcher Tableau[5]: the correspective integration scheme.
- n_steps, (Def. 1): this setting represents the number of integration steps to perform for each time interval. In case of a number of steps n greater than one, the time interval is subdivided in n equal parts and for each part a numerical integration step is performed.

4.2 Hybrid-Branch&Bound Options

In the following, the possible options for the Hybrid-Branch&Bound algorithm of OpenBB are presented and briefly explained:

- nlpSettings, (Def. IPOPTsettings()): settings to be passed to the NLP subsolver[0].
- mipSettings, (Def. BBsettings()): settings to be passed to the MIP subsolver (see Section 4.3).
- verbose, (Def. false): print info during execution.
- nlpProcesses, (Def. 1): max number of processes for nlp.

[3] In the first iteration, the method expects for each input a complete set of values for the time steps going from 0 to $P_l + R_l$, where P_l is the prediction hrizon length and R_l is the relaxed tail length. In the subsequent interations, the method expects the value of the inputs only for the "new" part of the subproblem, i.e., the last $n =$ shift_size values.

[4] The 'hotstart' mode requires the re-generation of the root for the new subproblem. Therefore, the approach tries only to speed-up the identification of a first close-to-optimal assignment.

[5] As presented in: https://en.wikipedia.org/wiki/List_of_Runge%E2%80%93Kutta_methods

- mipProcesses, (Def. 1): max number of processes for mip.
- nlpStepType, (Def. (:OAnlpStep,)): defines the kind of nlp step to use[0].
- mipStepType, (Def. (:OAmipStep,)): defines the kind of mip step to use[0].
- conservativismLevel, (Def. 0): 0: normal BB, 1: suboptimal nodes are stored, 2: infeasible and suboptimal nodes are stored.
- optimalControlInfo, (Def. (-1,-1,-1)): enables some optimal-control-specific rutines. The values represent: (numStates,numAlgebraicVars,numControls).
- acceptUnreliableSolutions, (Def. false): consider solutions also in case of unreliability[0].
- limitedMemory, (Def. false): use all constraints for linearizations or only active ones.
- primalTolerance, (Def. 1e-4): constraints violation tolerance.
- dualTolerance, (Def. 1e-6): dual constraints violation tolerance.
- objectiveCutoff, (Def. Inf): look only for solutions that are better than the provided upper bound[0].
- gapsComputationMode, (Def. :onAverage): computes the optimality gaps from the objective bounds[0].
- timeLimit, (Def. Inf): max running time.
- iterationsLimit, (Def. 0): max number of iterations (0 means no-limit).
- numIncumbentsLimit, (Def. 0): stop after the given number of integral solutions have been found (0 means no-limit).
- absoluteGapTolerance, (Def. 1e-4): stop if the absolute optimality gap is below the given level.
- relativeGapTolerance, (Def. 1e-6): stop if the relative optimality gap is below the given level.
- customInterruptionRule, (Def. workspace->false): user-defined stopping criterion[0].
- mipStepStoppingCriterion, (Def. (:fullOptimality,)): stopping criterion for MIP step[0].
- heuristicsSettings, (Def. NullHBBheuristicsSettings()): defines which heuristics to use[0].
- heuristicsTriggerCondition, (Def. (:never_trigger,)): rule to trigger the heuristic[0].

4.3 Branch&Bound Options

In the following, the possible options for the Branch&Bound algorithm of OpenBB are presented and briefly explained:

- subsolverSettings, (Def. NullSubsolverSettings()): those are passed directly to the subsolver[0].
- verbose, (Def. false): print info during execution.
- statusInfoPeriod, (Def. 1.0): frequency of status info print.
- numProcesses, (Def. 1): max number of processes to launch.
- conservativismLevel, (Def. 0): 0: normal BB, 1: suboptimal nodes are stored, 2: infeasible and suboptimal nodes are stored.
- nodeSharingPeriod, (Def. 2): send every n-th node to the neighbouring process.
- primalTolerance, (Def. 1e-4): constraints violation tolerance.
- dualTolerance, (Def. 1e-6): dual constraints violation tolerance.
- objectiveCutoff, (Def. Inf): look only for solutions that are better than the provided upper bound.
- gapsComputationMode, (Def. :onUpperBound): computes the optimality gaps from the objective bounds.
- expansionPriorityRule, (Def. (:lower_pseudoObjective,)): ordering of the nodes in the activeQueue[0].

- branchingPriorityRule, (Def. (:pseudoIncrements_geomean,2,1)): ordering of the discrete variables for branching[0].
- unreliablesPriority, (Def. 0): activeQueue insertion priority for unreliable nodes (-1: low, 0: normal, 1: high).
- pseudoCostsInitialization, (Def. (:initialize_to_constant!,1e-4)): function returning the initialization of the pseudo-costs[0].
- customInterruptionRule, (Def. x -> false): user-defined stopping criterion[0].
- timeLimit, (Def. Inf): max running time.
- numSolutionsLimit, (Def. 0): stop after the given number of integral solutions have been found.
- absoluteGapTolerance, (Def. 1e-3): stop if the absolute optimality gap is below the given level.
- relativeGapTolerance, (Def. 1e-3): stop if the relative optimality gap is below the given level.
- heuristicsSettings, (Def. BBroundingHeuristicsSettings()): defines which heuristics to use[0].
- heuristicsTriggerCondition, (Def. (:trigger_on_pseudoObjective,)): rule to trigger the heuristic[0].
- withBoundsPropagation, (Def. false): enables bounds propagation.
- optimalControlInfo, (Def. (-1,-1,-1)): enables some optimal-control-specific rutines. The values represent: (numStates,numAlgebraicVars,numControls).
- acceptUnreliableSolutions, (Def. false): consider solutions also in case of unreliability.
- maxNumberOfLocalCuts, (Def. 5): Each node carries around this number of cuts to improve the relaxations.

[0]Refer to the OpenBB.jl documentation for further details