Code Manual for Submitted Publication "Computational comparison of time transformed mixed-integer optimal control problems"

Sebastian Sager, Manuel Tetschke, Clemens Zeile

July 3, 2021

Contents

1 Introduction **1**

2 Used software tools **2**

3 The Software Package `ampl_mintoc` **2**
3.1 Main Idea and Problem-Independent Files 3
3.2 How to Set Up and Solve an MIOC 4
3.3 Plotting of Results and Postprocessing 5

4 How to reproduce publication results and plots **5**
4.1 Plots 5
4.2 Tables 6

1 Introduction

The computations of this publication are based on the `AMPL` code `ampl_mintoc`. The latter is designed to solve general optimal control problems (OCP) in switched systems form. A presentation about the general framework can be found in the folder *doc*, named *mintoc.pdf*. In `ampl_mintoc`, problems to be solved are declared via one data file, one model file and one run file each. The file *mintocPre.mod* contains the declaration of general parameters such as time discretization and number of differential states as well as variables such as the right hand side of the ODE f, binary controls ω and so on. In *mintocPost.mod*, we define numerical integration schemes for the ODE's, the STO and POC formulations of the MIOCP formulated as NLP and further problem formulations. Both files should be included in the problem-specific run file and reduce the effort to

formulate the MIOCP significantly.

2 Used software tools

The following software tools were used as part of our numerical results:

1. `AMPL`, we used version 20210226, with solver `Ipopt` 3.12.13.
2. `GNU bash`, version 4.4.20(1)-release (x86_64-pc-linux-gnu) for scripts running `AMPL` problem instances.
3. `sed` (GNU sed) 4.4, for modifying `ampl_mintoc` files as part of the Bash scripts.
4. `gnuplot`, version 5.2 patchlevel 2, for creating plots of individual problem trajectories (e.g. differential states, adjoints, controls)
5. `Python`, version 3.6.9 extracting results from data files.

3 The Software Package `ampl_mintoc`

This section can also be found in more detail in the manuscript.

We designed the `AMPL` code `ampl_mintoc` for the formulation of `M`ixed `INT`eger `O`ptimal `C`ontrol problems, in particular for problems of type (MIOCP). It consists of a set of code files that allow an efficient formulation of control problems, which are then solved by off-the-shelf NLP or MINLP solvers interfaced by `AMPL`. In particular, problem-independent code exists to apply (STO) as well as (POC) formulations to problem-dependent functions.

The advantage of using `AMPL` is that the syntax remains close to the mathematical formulation. Due to the direct approaches for the solution of the OCPs, the implementation is done directly in a discretized setting. The program's preprocessor can eliminate variables and constraints before the solution process. The package `ampl_mintoc` is available as open-source under `https://github.com/czeile/ampl_mintoc`.

A disadvantage of using `AMPL` is that function pointers are not straightforward to use. One possibility is to work with user functions and an extension back end. We do not use the TACO package, but a new approach with lifted (and later eliminated) variables to obtain function evaluations and a convenient interface to model right-hand side functions independent of a discretization scheme. This allows to work completely in `AMPL` without additional programming languages. Although TACO is the method of choice to interface advanced optimal control solvers with adaptive integrators, our approach allows a transparent and easy-to-compare implementation of (STO) and (POC).

In Section 3.1, we describe the crucial files and explain the different algorithmic options and discretizations. In Section 3.2 we deal with how optimal control problems can be formulated with `ampl_mintoc` and, finally, in Section 3.3 we explain how the solutions found can be plotted and restored.

3.1 Main Idea and Problem-Independent Files

In order to save implementation effort, problem independent variables and generic problem structures are outsourced to dedicated `AMPL` files. The file `mintocPre.mod` contains the declaration of general parameters such as the time discretization and the number of differential states and controls. It also defines the variables for the ODE model function, the control values, differential states, and discretized constraint functions. In `mintocPost.mod`, we define different objectives, constraints, numerical integration schemes for the ODEs, and the formulation of NLPs and Combinatorial Integral Approximation problems.

The files `mintocPre.mod` and `mintocPost.mod` can be considered as the core of `ampl_mintoc`. Both files need to be included in the problem specific run file and reduce the effort to formulate the optimal control problems significantly. In `mintocPre.mod`, also the problem independent formulations of the ODE constraints are provided for STO and POC:

Listing 1: AMPL model file with problem-independent reformulations (STO-f) and (POC-f)

```
### Switching Time Optimization
  ode_STO {k in X, o in 1..no, i in IU, ii in 0..nsto-1}:
    x[k,i*ntperu+(o-1)*nsto+ii+1] = (
      if (integrator=="explicitEuler") then
        x[k,i*ntperu+(o-1)*nsto+ii] + wi[o,i] * dt*no *
         (f[k,0,i*ntperu+(o-1)*nsto+ii] + f[k,o,i*ntperu+(o-1)*nsto+ii])
    );

### Partial Outer Convexification
  ode_POC {k in X, i in 0..nt-1}:
    x[k,i+1] =  (
      if (integrator=="explicitEuler") then
        x[k,i] + dt * (f[k,0,i] + sum {o in Omega} w[o,i] * f[k,o,i])
    );
```

Currently available numerical integrator schemes are the Radau collocation and explicit and implicit Euler methods, which can be set via the variable `integrator`. Various problem variants can be solved via setting the variable `mode` accordingly. For instance, in the `Simulate` mode, all control variables are fixed and in `Relaxed` the optimization problem is solved with relaxed integer variables. The file `solve.run` contains the commands to address and run the NLP solver based on the chosen algorithmic options. Thus, one needs to include `include ../solve.run` in the problem-specific run file to let the problem at hand be solved. The files `solveMILP.run` and `solveRound.run` work similarly as `solve.run`, but address MILP solvers for specific combinatorial integral approximation problems and contain specific rounding algorithms, respectively, as part of the combinatorial integral approximation decomposition. The declaration of default settings for solvers, problem classes, and integrators are outsourced in the file `set.run`, while `checkConsistency.run` verifies if valid options/modes have been chosen for solver, integrator, and discretization.

The problem-independent files do not need to be modified by users, but can be extended by algorithm developers.

3.2 How to Set Up and Solve an MIOC

Following AMPL standard, a problem specification usually involves one data file, one model file and one run file. An optimal control problem `prob` can be formulated using `ampl_mintoc` via the problem-specific files `prob.dat`, `prob.mod`, and `prob.run`. First, problem dimensions (e.g. of the differential states and controls), parameters (e.g. the number of discretization intervals), and initial values are specified in `prob.dat`. In the `prob.mod` file, model-specific functions can be defined (in the discretized sense). This includes the model function f, mixed state-control constraints, path constraints c, and vanishing constraints. Listings 2 and 3 show the `AMPL` implementations for the (Lotka) and (Calcium) problems as examples of how to formulate f and, hence, the problem's dynamics.

Listing 2: Problem-dependent functions f_j as AMPL code for (Lotka). The index k denotes the entry of x_k, the index $o \in \{1, \ldots, n_\sigma\}$ the switching mode in f_o, and i is the time index.

```
var f {k in X,o in 0..no,i in I} = (
  if      (k==1 && o==0) then x[1,i] - x[1,i]*x[2,i]
  else if (k==2 && o==0) then - x[2,i] + x[1,i]*x[2,i]
  else if (k==1 && o==1) then - x[1,i]*p[1]
  else if (k==2 && o==1) then - x[2,i]*p[2]
  else if (k==3 && o==0) then sum{l in 1..2} (x[l,i] - xtilde[l])^2
);
```

Listing 3: Problem-dependent functions f_j as AMPL code for (**??**).

```
var f {k in X,o in 0..no,i in I} = (
  if      (k==1 && o==0) then k1 + k2*x[1,i]
    - k3*x[1,i]*x[2,i]/(x[1,i]+K4) - k5*x[1,i]*x[3,i]/(x[1,i]+K6)
  else if (k==2 && o==0) then k7*x[1,i] - k8*x[2,i]/(x[2,i]+K9)
  else if (k==3 && o==0) then k10*x[2,i]*x[3,i]*x[4,i]/(x[4,i]+K11)
    + p18*k12*x[2,i] + k13*x[1,i] - k14*x[3,i]/(1.0*x[3,i]+K15)
    - k16*x[3,i]/(x[3,i]+K17) + x[4,i]/10
  else if (k==4 && o==0) then - k10*x[2,i]*x[3,i]*x[4,i]/(x[4,i]+K11)
    + k16*x[3,i]/(x[3,i]+K17) - x[4,i]/10
  else if (k==3 && o==1) then k14*x[3,i]/(1.0*x[3,i]+K15)
    - k14*x[3,i]/(1.3*x[3,i]+K15)
  else if (k==5 && o==0) then sum{k in 1..4} (x[k,i]-xtilde[k])^2
);
```

Note that the formulation uses the time index i. While this might not be a very elegant way to define functions, it comes with the advantage that also time-delayed systems can be modeled in a flexible way.

Finally, algorithmic choices can be provided in `prob.run`, such as the integration scheme, the problem reformulation, the simulation or optimization algorithm, relaxation of integer variables, or postprocessing of results.

Listing 4: Exemplary run file `lotka.run` as AMPL code.

```
model ../mintocPre.mod;
model lotka.mod;
model ../mintocPost.mod;
data lotka.dat;

let nlpsolver := "ipopt";
```

```
let integrator := "explicitEuler";
let mode := "SimulateSTO";

include ../solve.run;
display objective;
```

Listing 4 shows, as an example, how the (STO) formulation of the (Lotka) problem is simulated, using an explicit Euler scheme and IPOPT.

3.3 Plotting of Results and Postprocessing

Standardized routines are implemented in the `ampl_mintoc` code to postprocess, store, and restore problem solutions. These can be retrieved via the `problem.run` file and the `AMPL` command `include`. For instance, by calling the script `printOutput.run`, detailed solution results can be output, such as the objective function value, constraint violation metrics, and the run time. If the user includes the script `plot.run`, the solution found is stored in a file `problem.plot`, which in turn can be used by `gnuplot` to plot result data such as the state or control trajectories. Furthermore, we can use the scripts `storeTrajectory.run` and `readTrajectory.run` to save or read in problem solutions in order to solve a problem again later in a modified form.

4 How to reproduce publication results and plots

For the convenience of the reader, we provide the scripts, that reproduce the results for the article's figures and tables in the following. You find the scripts in the folder "Sager2021".

4.1 Plots

Fig. 2:
1. Perform the numerical calculations using
   `ampl lotka_example_control_sto.run` and
   `ampl lotka_example_control_poc.run`
2. Produce the ps files by running
   `gnuplot res/figure_poc.plot` and
   `gnuplot res/figure_sto.plot`
3. Results can be found in
   `res/figure_pocRelaxed_1.ps` and
   `res/figure_stoRelaxed_1.ps`

Fig. 3:
1. Perform the numerical calculations using
   `ampl sim2d_lotka.run`
2. Resulting raw data (in format "w1, w2, objective") can be found in
   `out/lotkapoc2d.txt` and
   `out/lotkasto2d.txt`

Fig. 4:
1. Perform the numerical calculations using
   `ampl sim2d_calcium.run`
2. Resulting raw data (in format "w1, w2, objective") can be found in
   `out/calciumpoc2d.txt` and
   `out/calciumsto2d.txt`

Fig. 5:
1. Perform the numerical calculations using
   `ampl sim1d_lotka.run` and
   `ampl sim1d_calcium.run`
2. Resulting raw data (in format "w1, objective") can be found in
   `out/lotkapoc1d.txt`,
   `out/lotkasto1d_N.txt`, $N \in \{1, 2, 3, 4, 8, 16\}$ and
   `out/calciumpoc1d.txt`,
   `out/calciumsto1d_N.txt`, $N \in \{16, 32, 64, 128\}$

Fig.6:
1. Perform the numerical calculations using
   `ampl simncrnd_sto_lotka.run`
   `ampl simncrnd_poc_lotka.run`
   `ampl simncrnd_sto_calcium.run`
   `ampl simncrnd_poc_calcium.run`
2. Resulting raw data (in format "N, R^*") can be found in
   `out/lotkapocncr.txt`, `out/lotkastoncr.txt` and
   `out/calciumpocncr.txt`, `out/calciumstoncr.txt`

4.2 Tables

Table 1:
1. Perform the numerical calculations using
   `ampl table1_lotka.run` and
   `ampl table1_calcium.run`
2. Resulting raw data (in format "n_t, problem, method, objective") can be found in
   `out/table1_lotka.txt` and
   `out/table1_calcium.txt`

The following codes were executed in parallel on multiple screens[1] of the same server, to save time. The longest scripts take more than one week to finish. Not using parallelization would result in a runtime of more than one month.

Table 2:
1. Perform (in parallel) the numerical calculations using
   `./runme_lotka_N2`
   `./runme_lotka_N3`
   `./runme_lotka_N4`

[1] https://help.ubuntu.com/community/Screen

```
./runme_lotka_N5
./runme_lotka_N8
./runme_lotka_N100
```

2. The summary of the raw data can be produced using `python createTableData_lotka.py`, which will result in a printed output in the terminal

Table 3:

1. Perform (in parallel) the numerical calculations using
```
./runme_calcium_N2
./runme_calcium_N3
./runme_calcium_N4
./runme_calcium_N100
```

2. The summary of the raw data can be produced using `python createTableData_calcium.py`, which will result in a printed output in the terminal.