1 Introduction

The advanced step NMPC code compiles into a library that holds the code for the Schur complement computation, and an executable which holds an interface to AMPL. This document describes the usage of the asNMPC code using the C++ and the AMPL interface.

2 AMPL Interface

Since many models are written in AMPL, this interface makes it very easy to apply any of the capabilities of the AsNMPC code to your existing models. The usage of the different tools using this AMPL interface is described below.

2.1 Advanced Step NMPC

Given a discretized DAE model of an optimal control problem in AMPL, applying the advanced step algorithm is very easy. Suppose a model

$$\begin{array}{cc} \min\limits_{z} & \phi(z) \\ s.t. & c(z) = 0 \\ & z_0 - w_0 = 0 \\ & z \geq 0 \end{array} \tag{1}$$

Not that for the asNMPC formulation, there need to be explicitly defined variables z_0 for the initial values (whereas usually, these are just parameters). To define the values for these new variables, additional initial value constraints $z_0 - w_0 = 0$ have to be added. Given this formulation, only four suffixes need to be set to initialize the asNMPC problem:

nmpc_state_0 This suffix has to be set for the initial variables z_0. The suffixes have to be set to values from 1 to length(z_0). This enumeration is crucial.

nmpc_state_1 This suffix has to be set for the discretized variables at timestep t corresponding to the inital variables z_0. They have to be indexed the same way as nmpc_state_0.

nmpc_state_value_1 This suffix holds the perturbed values for the variables at timestep t. It has to be set for the same variables as nmpc_state_1.

nmpc_init_constr This suffix has to be set for the inital value constraints $z_0 - w_0 = 0$. If a constraint is an initial value constraint, set this suffix to 1 (no indexing is necessary).

If you have set these suffixes, the only thing missing is to enable the asNMPC algorithm by setting the solver option

```
option ipopt_options 'run_nmpc yes';
```

This option can alternatively be set in the ipopt.opt file.

To make sure that the initial value constraints are not eliminated by AMPL, the presolve feature should be turned off as well:

```
option presolve 0;
```

An example implementation of the above is provided in the directory

`examples/hicks_ampl`

There, the asNMPC code is applied to the reactor model described in [1].

The AMPL interface as described above only works for simulations, where the future measurement is known beforehand. Using the AMPL model definiton with an outside measurement device is generally possible. If you intend on using this feature in a real-time environment, please contact the author.

2.2 Parametric Sensitivity

The parametric sensitivity feature uses the same suffixes as the asNMPC algorithm. The parameters need to be defined as variables and set using the initial value constraints described above. The only difference in the parametric case is that the suffixes nmpc_state_0 and nmpc_state_1 are not set for different variables, but both for the parameters.

A small example of the parametric sensitivity feature is located in the directory

`examples/parametric_ampl`

2.3 Reduced Hessian

The reduced hessian feature is even easier to use. The critical problem is to decide which variables will be free variables at the optimal solution. The free variables have to be marked by setting the suffix

`red_hessian`

to $1..n$, where n is the number of free variables. The columns of the inverse reduced hessian, which will be printed, is determined by the ordering of these indices.

An example of the reduced hessian calculation can be found in

`examples/red_hess_ampl`

3 C++ Interface

The C++ interface is very simple to apply to an existing `Ipopt::TNLP` implementation. With the member function `TNLP::::get_var_con_metadata`, Ipopt provides a feature very similar to that of AMPL suffixes.

The procedure of making a TNLP class capable to be used with the asNMPC code follows the same lines as in AMPL. First, the parameters / initial values have to be introduced as variables. Then, the initial value equations need to be added to the constraints, and the jacobian computation has to be adjusted accordingly. Finally, the suffixes have to be set the same way they would in AMPL as described above, using the member function `TNLP::::get_var_con_metadata`. This is illustrated in the examples `examples/redhess_cpp` and `examples/parametric_cpp`

4 Options

There are several new options that can be set in the `ipopt.opt` file, that determine the behavior of the asNMPC code. The most important options are the ones that turn the execution of the additional post-optimal code on and off. These options are

```
run_nmpc yes
```

for the advanced step and parametric sensitivity computation, and

```
compute_red_hessian yes
```

for the reduced hessian computation.
Further options for the advanced step code are

select_step This option determines which formula is used to compute the advanced step. The options are `advanced` for the full advanced step with Schur complement and multiplier correction, `sensitivity` for the Schur step without multiplier correction, and `ift` for the fast backsolve without Schur complement computation.

n_nmpc_steps In general, the advanced step is designed to accomodate an arbitrary number of advanced steps. Right now, however, this value has to be set to either 0 or 1.

nmpc_boundcheck If set to `yes`, this option turns on the bound correction algorithm within the advanced step.

nmpc_bound_eps This option sets the value by which the bounds are to be relaxed in the boundcheck mode.

References

[1] G. A. Hicks and W. H. Ray. Approximation methods for optimal control synthesis. *The Canadian Journal of Chemical Engineering*, 49:522–528, 1971.