GRAMPC documentation

(pronounced gramp-c [græmp'si:])

Version 2.2

Tobias Englert, Andreas Völz, Felix Mesmer,
Sönke Rhein, Knut Graichen

Former contributors (GRAMPC version < 2.0):
Bartosz Käpernik, Tilman Utz

Chair of Automatic Control
Friedrich-Alexander-Universität Erlangen-Nürnberg

October 31, 2019

Contents

1 Introduction

This manual describes the model predictive control tool GRAMPC (gradient-based MPC – [græmp′si:]) for nonlinear continuous-time systems subject to (possibly nonlinear) state and control constraints. The optimization algorithm underlying GRAMPC consists of an augmented Lagrangian scheme in connection with a tailored gradient method. GRAMPC is implemented as C code with an additional user interface to C++, Matlab/Simulink, and dSpace. GRAMPC allows one to cope with (embedded) MPC problems of nonlinear and highly dynamical systems with sampling times in the (sub)millisecond range.

The presented framework is a fundamental revision of version 1.0 of the MPC toolbox GRAMPC [9] that was originally presented for nonlinear systems with pure input constraints. Beside “classical” nonlinear MPC, GRAMPC can be used for MPC on shrinking horizon, general optimal control problems, moving horizon estimation, and parameter optimization problems.

The documentation is outlined as follows. Chapter 2 describes the installation of GRAMPC for use in C and Matlab and gives a brief overview on the software structure. The formulation of the optimization problem is shown in Chapter 3. Furthermore, the chapter describes the available parameters and the implementation of the problem as C functions. Chapter 4 summarizes the optimization algorithm and provides a detailed description of the available options. The usage of GRAMPC in C and Matlab is explained in Chapter 5, which includes initialization, setting of parameters and options, compiling and running as well as the interfaces to Matlab and Simulink. Chapter 6 describes three example problems illustrating the application of GRAMPC to model predictive control, optimal control and moving horizon estimation. Valuable hints for tuning the software to a specific optimization problem are given at multiple places in the documentation, see especially the description of the options in Chapter 4, the provided plot functions in Section 5.2.3 and the tutorials in the last chapter.

Note that the PDF version of this documentation provides many hyperlinks to quickly jump to the definition of parameters, options and functions. In addition, the descriptions of parameters and options are repeated in the Appendix.

2 Installation and structure of GRAMPC

The following two subsections describe the installation procedure of GRAMPC for use in C and Matlab on a MS Windows operating system. Section 2.3 presents the principal structure of the toolbox.

2.1 Installation of GRAMPC for use in C

A convenient way to use GRAMPC in C/C++ under MS Windows is the Linux environment Cygwin. To install Cygwin, download the setup file from the web page `http://www.cygwin.com/` and follow the installation instructions. In the installation process when packages can be selected, you have to choose the `gcc` compiler and `make`. If Cygwin is properly installed, open a Cygwin terminal and perform the following steps:

1. Download the current version of GRAMPC from the web page `https://sourceforge.net/projects/grampc/`.
2. Unpack the archive `grampc_v2.0.zip` to an arbitrary location on your computer. After the unpacking procedure, a new directory with the following subfolders is created:
   - **cpp**: Interface of GRAMPC to C++.
   - **doc**: Contains this GRAMPC documentation.
   - **examples**: This folder contains several executable MPC, MHE and OCP problems as well as templates for implementing your own problems.
   - **include**: The header files of the GRAMPC project are located in this folder.
   - **libs**: This folder is only available after compiling the GRAMPC toolbox and contains the library `libgrampc.a`.
   - **matlab**: Interface of GRAMPC to Matlab/Simulink, also see Section 2.2.
   - **src**: The source files of the GRAMPC project are located in this folder.

   The GRAMPC directory additionally contains a makefile for building the GRAMPC toolbox. For the remainder of this manual, the location of the created GRAMPC folder will be denoted by `<grampc_root>`.
3. Compile GRAMPC by running the following commands in a terminal:

   ```
   $ cd <grampc_root>
   $ make clean all
   ```

   The dollar symbol "`$`" indicates the line prompt of the terminal. The make command compiles the source files and generates the library `libgrampc.a` within `<grampc_root>/libs`, which can now be used to solve a suitable problem in C. The additional argument `clean all` removes previously installed parts of GRAMPC.

2.2 Installation of GRAMPC for use in Matlab

GRAMPC requires a C compiler that is supported by Matlab for a direct use. Details on supported compilers for the current Matlab version as well as previous releases can be found via the MATHWORKS homepage. The correct linkage of the compiler to Matlab can be checked by typing

```
>> mex -setup
```

in the Matlab terminal window and subsequently selecting the corresponding C compiler. The symbol ">>" denotes the Matlab prompt. The GRAMPC installation under Matlab proceeds in two steps:

1. After downloading and unpacking GRAMPC as described in Section 2.1, go to the Matlab directory

   ```
   >> cd <grampc_root>/matlab
   ```

   which contains the following subfolders:

   - **bin**: This folder is only available after compiling the GRAMPC toolbox and contains the object files of GRAMPC.
   - **include**: The header files of the GRAMPC project for the Matlab interface.
   - **mfiles**: Various auxiliary functions for the Matlab interface.
   - **src**: C sources files of the MEX files which provide the interface between GRAMPC and Matlab.

   In addition, the subfolder contains the m-file `make.m` to start the building process.

2. Build the necessary object files for GRAMPC by executing the make function. The compilation of the source files can be performed with the following options:

   - `>> make clean`: removes all previously built GRAMPC files,
   - `>> make`: creates the necessary object files to use GRAMPC,
   - `>> make verbose`: the object files are created in verbose mode, i.e. additional information regarding the building process are provided during the compilation,
   - `>> make debug`: the debug option creates the object files with additional information for use in debugging,
   - `>> make debug verbose`: activates the debug option as well as the verbose mode.

   Similar to the compiling procedure in C as described in Section 2.1, the `make` command compiles the source files to generate object files within `<grampc_root>/matlab/bin`, which can now be used to solve a suitable problem in Matlab.

2.3 Structure of GRAMPC

The aim of GRAMPC is to be portable and executable on different operating systems and hardware devices without the use of external libraries. After the installation procedure as described in Section 2.1 and 2.2, the GRAMPC structure shown in Figure 2.1 is available to cope with problems from optimal control, model predictive control, moving horizon estimation, and parameter optimization. As illustrated in Figure 2.1, the GRAMPC project is implemented in plain C with a user-friendly interface to C++, Matlab/Simulink, and DSpace.

A specific problem can be implemented in GRAMPC using the C template `probfct_TEMPLATE.c` included in the folder `<grampc_root>/examples/TEMPLATES`. A more detailed discussion about this step can be found in Chapter 3. The workspace of a GRAMPC project as well as algorithmic options

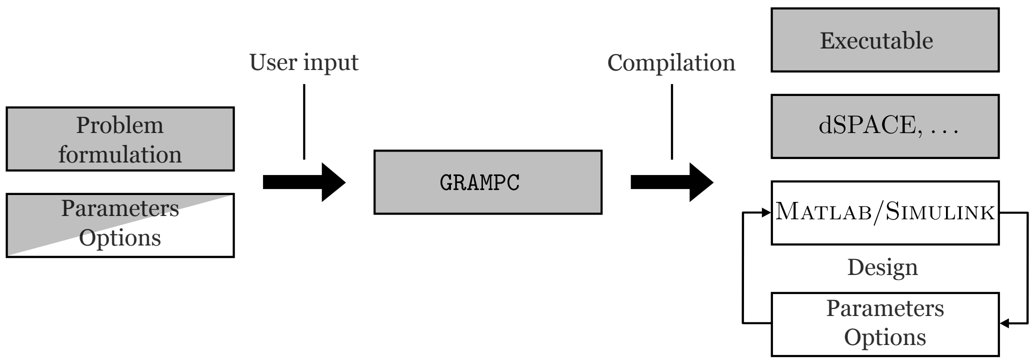

Figure 2.1: General structure of GRAMPC (gray – C code, white – Matlab code).

and parameters are stored by the structure variable `grampc`. While several parameter settings are problem specific and need to be provided, most values are set to their respective default value, see Chapter 4. The GRAMPC structure can be manipulated through a generic interface, e.g. in order to set algorithmic options or parameters for a specific problem without the need to recompile the `grampc` project every time.

A specific example contained in the folder `<grampc_root>/examples` can be compiled in C as well as in Matlab and linked against the GRAMPC toolbox. A more detailed discussion on this step can be found in Chapter 5.

3 Problem formulation and implementation

GRAMPC provides a solver for nonlinear input and state constrained optimal control problems. It is in particular tailored to the application of real-time MPC with a moving or shrinking horizon with focus on a memory and time efficient implementation. Other applications concern the problem of moving horizon estimation and parameter estimation.

This chapter describes the underlying optimization problem and how it can be implemented for a specific application in GRAMPC. In addition, the parameter structure of GRAMPC is introduced.

3.1 Optimization problem and parameters

GRAMPC allows one to cope with optimal control problems of the following type

$$
\begin{aligned}
\min_{\boldsymbol{u},\boldsymbol{p},T} \quad & J(\boldsymbol{u},\boldsymbol{p},T;\boldsymbol{x}_0) = V(\boldsymbol{x}(T),\boldsymbol{p},T) + \int_0^T l(\boldsymbol{x}(t),\boldsymbol{u}(t),\boldsymbol{p},t)\,\mathrm{d}t && \text{(3.1a)}\\
\text{s.t.} \quad & \boldsymbol{M}\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t),\boldsymbol{u}(t),\boldsymbol{p},t_0+t)\,, \quad \boldsymbol{x}(t_0) = \boldsymbol{x}_0 && \text{(3.1b)}\\
& \boldsymbol{g}(\boldsymbol{x}(t),\boldsymbol{u}(t),\boldsymbol{p},t) = \boldsymbol{0}\,, \quad \boldsymbol{g}_T(\boldsymbol{x}(T),\boldsymbol{p},T) = \boldsymbol{0} && \text{(3.1c)}\\
& \boldsymbol{h}(\boldsymbol{x}(t),\boldsymbol{u}(t),\boldsymbol{p},t) \le \boldsymbol{0}\,, \quad \boldsymbol{h}_T(\boldsymbol{x}(T),\boldsymbol{p},T) \le \boldsymbol{0} && \text{(3.1d)}\\
& \boldsymbol{u}(t) \in [\boldsymbol{u}_{\min},\boldsymbol{u}_{\max}] && \text{(3.1e)}\\
& \boldsymbol{p} \in [\boldsymbol{p}_{\min},\boldsymbol{p}_{\max}]\,, \quad T \in [T_{\min},T_{\max}] && \text{(3.1f)}
\end{aligned}
$$

in the context of model predictive control, moving horizon estimation, and/or parameter estimation. The cost functional to be minimized (3.1a) consists of the continuously differentiable terminal cost (Mayer term) $V : \mathbb{R}^{N_{\boldsymbol{x}}} \times \mathbb{R}^{N_{\boldsymbol{p}}} \times \mathbb{R} \to \mathbb{R}$ and integral cost (Lagrange term) $l : \mathbb{R}^{N_{\boldsymbol{x}}} \times \mathbb{R}^{N_{\boldsymbol{u}}} \times \mathbb{R}^{N_{\boldsymbol{p}}} \times \mathbb{R} \to \mathbb{R}$ with the state variables $\boldsymbol{x} \in \mathbb{R}^{N_{\boldsymbol{x}}}$, the control variables $\boldsymbol{u} \in \mathbb{R}^{N_{\boldsymbol{u}}}$, the parameters $\boldsymbol{p} \in \mathbb{R}^{N_{\boldsymbol{p}}}$, and the end time $T \in \mathbb{R}$.

The cost functional $J(\boldsymbol{u},\boldsymbol{p},T;\boldsymbol{x}_0,t_0)$ is minimized with respect to the optimization variables $(\boldsymbol{u},\boldsymbol{p},T)$ subject to the system dynamics (3.1b) with the mass matrix $\boldsymbol{M} \in \mathbb{R}^{N_{\boldsymbol{x}} \times N_{\boldsymbol{x}}}$, the continuously differentiable right hand side $\boldsymbol{f} : \mathbb{R}^{N_{\boldsymbol{x}}} \times \mathbb{R}^{N_{\boldsymbol{u}}} \times \mathbb{R}^{N_{\boldsymbol{p}}} \times \mathbb{R} \to \mathbb{R}^{N_{\boldsymbol{x}}}$, and the initial state $\boldsymbol{x}_0$. GRAMPC allows one to formulate terminal equality and inequality constraints $\boldsymbol{g}_T : \mathbb{R}^{N_{\boldsymbol{x}}} \times \mathbb{R}^{N_{\boldsymbol{p}}} \times \mathbb{R} \to \mathbb{R}^{N_{\boldsymbol{g}_T}}$ and $\boldsymbol{h}_T : \mathbb{R}^{N_{\boldsymbol{x}}} \times \mathbb{R}^{N_{\boldsymbol{p}}} \times \mathbb{R} \to \mathbb{R}^{N_{\boldsymbol{h}_T}}$ as well as general equality and inequality constraints $\boldsymbol{g} : \mathbb{R}^{N_{\boldsymbol{x}}} \times \mathbb{R}^{N_{\boldsymbol{u}}} \times \mathbb{R}^{N_{\boldsymbol{p}}} \times \mathbb{R} \to \mathbb{R}^{N_{\boldsymbol{g}}}$ and $\boldsymbol{h} : \mathbb{R}^{N_{\boldsymbol{x}}} \times \mathbb{R}^{N_{\boldsymbol{u}}} \times \mathbb{R}^{N_{\boldsymbol{p}}} \times \mathbb{R} \to \mathbb{R}^{N_{\boldsymbol{h}}}$. In addition, the optimization variables are limited by the box constraints (3.1e) and (3.1f).

The terminal cost V as well as the integral cost l and all constraints $(\boldsymbol{g},\boldsymbol{g}_{\mathrm{T}},\boldsymbol{h},\boldsymbol{h}_{\mathrm{T}})$ contain an explicit time dependency with regard to the internal time t. In addition, the system dynamics (3.1b) features an explicit time dependency. Note that in the context of MPC, the initial time t_0 and initial state $\boldsymbol{x}_0$ correspond to the sampling instant t_k that is incremented by the sampling time $\Delta t > 0$ in each MPC step k.

A detailed description about the implementation of the optimization problem (3.1) in C code is given in Section 3.2. In addition, some parts of the problem can be configured by parameters (cf. the GRAMPC data structure `param`) and therefore do not require repeated compiling. A list of all parameters with types and allowed values is provided in the appendix (Table A.1) Except for the horizon length `Thor` and the sampling time `dt`, all parameters are optional and initialized to default values. A description of all parameters is as follows:

- `x0`: Initial state vector $\boldsymbol{x}(t_0) = \boldsymbol{x}_0$ at the corresponding sampling time t_0.
- `xdes`: Desired (constant) setpoint vector for the state variables $\boldsymbol{x}$.
- `u0`: Initial value of the control vector $\boldsymbol{u}(t) = \boldsymbol{u}_0 = \text{const.}$, $t \in [0, T]$ that is used in the first iteration of GRAMPC.
- `udes`: Desired (constant) setpoint vector for the control variables $\boldsymbol{u}$.
- `umin`, `umax`: Lower and upper bounds for the control variables $\boldsymbol{u}$.
- `p0`: Initial value of the parameter vector $\boldsymbol{p} = \boldsymbol{p}_0$ that is used in the first iteration of GRAMPC.
- `pmin`, `pmax`: Lower and upper bounds for the parameters $\boldsymbol{p}$.
- `Thor`: Prediction horizon T or initial value if the end time is optimized.
- `Tmin`, `Tmax`: Lower and upper bound for the prediction horizon T.
- `dt`: Sampling time Δt of the considered system for model predictive control or moving horizon estimation. Required for prediction of next state `grampc.sol.xnext` and for the control shift, see Section 4.7.
- `t0`: Current sampling instance t_0 that is provided to the time-varying system dynamics (3.1b).
- `userparam`: Further problem-specific parameters, e.g. system parameters or weights in the cost functions that are passed to the problem functions via a `void`-pointer in C or `typeRNum` array in MATLAB.

Although GRAMPC uses a continuous-time formulation of the optimization problem (3.1), all trajectories are internally stored in discretized form with `Nhor` steps (cf. Section 4.2). This raises the question of whether all constraints are evaluated for the last trajectory point or only the terminal ones. In general, the constraints should be formulated in such a way that there are no conflicts. However, numerical difficulties can arise in some problems if constraints are formulated twice for the last point. Therefore, GRAMPC does not evaluate the constraints $\boldsymbol{g}$ and $\boldsymbol{h}$ for the last trajectory point if terminal constraints are defined, i.e. $N_{\boldsymbol{g}_T} + N_{\boldsymbol{h}_T} > 0$. In contrast, if no terminal constraints are defined, the functions $\boldsymbol{g}$ and $\boldsymbol{h}$ are evaluated for all points. Note that the opposite behavior is easy to implement by including $\boldsymbol{g}$ and $\boldsymbol{h}$ in the terminal constraints $\boldsymbol{g}_T$ and $\boldsymbol{h}_T$.

3.2 Problem implementation

Regardless of what kind of problem statement is considered (i. e. model predictive control, moving horizon estimation or parameter estimation), the optimization problem (3.1) must be implemented in C, cf. Figure 2.1. For this purpose, the C file template `probfct_TEMPLATE.c` is provided within the folder `<grampc_root>/examples/TEMPLATE`, which allows one to describe the structure of the optimization problem (3.1), the cost functional to be minimized (3.1a), the system dynamics (3.1b), and the constraints (3.1c)–(3.1d). The number of C functions to be provided depends on the type of dynamics (3.1b) of the specific problem at hand.

3.2.1 Problems involving explicit ODEs

A special case of the system dynamics (3.1b) and certainly the most important one concerns ordinary differential equations (ODEs) with explicit appearance of the first-order derivatives

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{p}, t_0 + t)\,, \tag{3.2}$$

corresponding to the identity mass matrix $\boldsymbol{M} = \boldsymbol{I}$ in OCP (3.1). In this case, the following functions of the C template file `probfct_TEMPLATE.c` have to be provided:

- `ocp_dim`: Definition of the dimensions of the considered problem, i.e. the number of state variables $N_{\boldsymbol{x}}$, control variables $N_{\boldsymbol{u}}$, parameters $N_{\boldsymbol{p}}$, equality constraints $N_{\boldsymbol{g}}$, inequality constraints $N_{\boldsymbol{h}}$, terminal equality constraints $N_{\boldsymbol{g}_T}$, and terminal inequality constraints $N_{\boldsymbol{h}_T}$.
- `ffct`: Formulation of the system dynamics function $\boldsymbol{f}$ (3.1b).
- `dfdx_vec`, `dfdu_vec`, `dfdp_vec`: Matrix vector products $(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}})^\mathsf{T}\boldsymbol{v}$, $(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}})^\mathsf{T}\boldsymbol{v}$ and $(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{p}})^\mathsf{T}\boldsymbol{v}$ for the system dynamics with an arbitrary vector $\boldsymbol{v}$ of dimension N_x.
- `Vfct`, `lfct`: Terminal and integral cost functions V and l of the cost functional (3.1a).
- `dVdx`, `dVdp`, `dVdT`, `dldx`, `dldu`, `dldp`: Gradients $\frac{\partial V}{\partial \boldsymbol{x}}$, $\frac{\partial V}{\partial \boldsymbol{p}}$, $\frac{\partial V}{\partial T}$, $\frac{\partial l}{\partial \boldsymbol{x}}$, $\frac{\partial l}{\partial \boldsymbol{u}}$, and $\frac{\partial l}{\partial \boldsymbol{p}}$ of the cost functions `Vfct` and `lfct`.
- `hfct`, `gfct`: Inequality and equality constraint functions $\boldsymbol{h}$ and $\boldsymbol{g}$ as defined in (3.1).
- `hTfct`, `gTfct`: Terminal inequality and equality constraint functions $\boldsymbol{h}_T$ and $\boldsymbol{g}_T$ as defined in (3.1).
- `dhdx_vec`, `dhdu_vec`, `dhdp_vec`: Matrix products $(\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{x}})^\mathsf{T}\boldsymbol{v}$, $(\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{u}})^\mathsf{T}\boldsymbol{v}$, and $(\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{p}})^\mathsf{T}\boldsymbol{v}$ for the inequality constraints with an arbitrary vector $\boldsymbol{v}$ of dimension N_h.
- `dgdx_vec`, `dgdu_vec`, `dgdp_vec`: Matrix product functions $(\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}})^\mathsf{T}\boldsymbol{v}$, $(\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{u}})^\mathsf{T}\boldsymbol{v}$, and $(\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{p}})^\mathsf{T}\boldsymbol{v}$ for the equality constraints with an arbitrary vector $\boldsymbol{v}$ of dimension N_g.
- `dhTdx_vec`, `dhTdp_vec`, `dhTdT_vec`: Matrix product functions $(\frac{\partial \boldsymbol{h}_T}{\partial \boldsymbol{x}})^\mathsf{T}\boldsymbol{v}$, $(\frac{\partial \boldsymbol{h}_T}{\partial \boldsymbol{p}})^\mathsf{T}\boldsymbol{v}$, and $(\frac{\partial \boldsymbol{h}_T}{\partial T})^\mathsf{T}\boldsymbol{v}$ for the terminal inequality constraints with an arbitrary vector $\boldsymbol{v}$ of dimension N_{h_T}.
- `dgTdx_vec`, `dgTdp_vec`, `dgTdT_vec`: Matrix product functions $(\frac{\partial \boldsymbol{g}_T}{\partial \boldsymbol{x}})^\mathsf{T}\boldsymbol{v}$, $(\frac{\partial \boldsymbol{g}_T}{\partial \boldsymbol{p}})^\mathsf{T}\boldsymbol{v}$, and $(\frac{\partial \boldsymbol{g}_T}{\partial T})^\mathsf{T}\boldsymbol{v}$ for the terminal equality constraints with an arbitrary vector $\boldsymbol{v}$ of dimension N_{g_T}.

The respective problem function templates only have to be filled in if the corresponding constraints and cost functions are defined for the problem at hand and depending on the actual choice of optimization variables ($\boldsymbol{u}$, $\boldsymbol{p}$, and/or T). For example, if only the control $\boldsymbol{u}$ is optimized, the partial derivatives with respect to $\boldsymbol{p}$ and T are not required.

The gradients $\frac{\partial V}{\partial \boldsymbol{x}}$, $\frac{\partial V}{\partial \boldsymbol{p}}$, $\frac{\partial V}{\partial T}$, $\frac{\partial l}{\partial \boldsymbol{x}}$, $\frac{\partial l}{\partial \boldsymbol{u}}$, and $\frac{\partial l}{\partial \boldsymbol{p}}$ as well as the matrix product functions listed above appear in the partial derivatives $H_{\boldsymbol{x}}$, $H_{\boldsymbol{u}}$ and $H_{\boldsymbol{p}}$ of the Hamiltonian H within the gradient method (see Section 4.1.2). The matrix product formulation is chosen over the definition of Jacobians (e.g. $(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}})^\mathsf{T}\boldsymbol{v}$ instead of $\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}$) in order to avoid unnecessary zero multiplications for sparse matrices or alternatively the usage of sparse numerics.

3.2.2 Problems involving semi-implicit ODEs and DAEs

Beside explicit ODEs, GRAMPC supports semi-implicit ODEs with mass matrix $\boldsymbol{M} \neq \boldsymbol{I}$ and DAEs with $\boldsymbol{M}$ being singular. The underlying numerical integrations of the dynamics (3.1b) is carried out using the integrator RODAS [7, 12]. In this case, additional C functions must be provided and several RODAS-specific options must be set, cf. Section 4.2.2 and Section 5.1.3. Especially, the option `IMAS`=1 must be set to indicate that a mass matrix is given. The numerical integrations performed with RODAS can be accelerated by providing partial derivatives. In summary, the following additional C functions are used by GRAMPC for semi-implicit ODEs and DAE systems:

- `Mfct`, `Mtrans`: Definition of the mass matrix $\boldsymbol{M}$ and its transpose $\boldsymbol{M}^\mathsf{T}$, which is required for the adjoint dynamics, cf. Section 4.1.2 in the projected gradient algorithm. The matrices must be specified column-wise. If the mass matrix has a band structure, only the respective elements above and below the main diagonal are specified. This only applies if the options `IMAS`=1 and `MLJAC`$< N_x$ are selected. Non-existent elements above or below the main diagonal must be filled with zeros so that the same number of elements is specified for each column.
- `dfdx`, `dfdxtrans`: The Jacobians $\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}$ and $(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}})^\mathsf{T} = \frac{\partial^2 H}{\partial \boldsymbol{x} \partial \boldsymbol{\lambda}}$ are provided by these functions if the option `IJAC`=1 is set. This allows one to evaluate the right hand sides of the canonical equations time efficiently. The Jacobians must be implemented in vector form by arranging the successive columns for $\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}$ and $(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}})^\mathsf{T}$. If the option `MLJAC`$< N_x$ is set to exploit the band structure of the Jacobians, only the corresponding elements above and below the main diagonal must be specified.
- `dfdt`, `dHdxdt`: The partial derivatives $\frac{\partial \boldsymbol{f}}{\partial t}$ and $\frac{\partial^2 H}{\partial \boldsymbol{x} \partial t}$ allow for evaluating the right hand sides of the canonical equations time efficiently, if the problem explicitly depends on time t. These functions must only be provided if the options `IFCN`=1 and `IDFX`=1 are used.

An MPC example with a semi-implicit system dynamics is included in `<grampc_root>/examples` (`Reactor_PDE`). The problem formulation is derived from a quasi-linear diffusion-convection-reaction system that is spatially discretized using finite elements.

3.2.3 Example: Ball-on-plate system

The appropriate definition of the C functions of the template `probfct_TEMPLATE` is described for the example of a ball-on-plate system [11] in the context of MPC. The problem is also included in `<grampc_root>/examples` (`BallOnPlate`). The underlying optimization problem reads as

$$\min_{u(\cdot)} \quad J(u;\boldsymbol{x}_0) = \frac{1}{2}\Delta\boldsymbol{x}^\mathsf{T}(T)\boldsymbol{P}\Delta\boldsymbol{x}(T) + \frac{1}{2}\int_0^T \Delta\boldsymbol{x}^\mathsf{T}(T)\boldsymbol{Q}\Delta\boldsymbol{x} + R\Delta u^2 \,\mathrm{d}\tau \tag{3.3a}$$

$$\text{s.t.} \quad \begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}0 & 1\\ 0 & 0\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} + \begin{bmatrix}-0.04\\ -7.01\end{bmatrix}u\,, \quad \begin{bmatrix}x_1(0)\\ x_2(0)\end{bmatrix} = \begin{bmatrix}x_{k,1}\\ x_{k,2}\end{bmatrix} \tag{3.3b}$$

$$\begin{bmatrix}-0.2\\ -0.1\end{bmatrix} \leq \begin{bmatrix}x_1\\ x_2\end{bmatrix} \leq \begin{bmatrix}0.01\\ 0.1\end{bmatrix}, \quad |u| \leq 0.0524\ . \tag{3.3c}$$

The cost functional to be minimized (3.3a) penalizes the state and input error $\Delta\boldsymbol{x} = \boldsymbol{x} - \boldsymbol{x}_\text{des}$ and $\Delta u = u - u_\text{des}$ in a quadratic manner using the weights

$$\boldsymbol{P} = \boldsymbol{Q} = \begin{bmatrix}100 & 0\\ 0 & 10\end{bmatrix}, \quad R = 1\ . \tag{3.3d}$$

The system dynamics (3.3b) describes a simplified linear model of a single axis of a ball-on-plate system [11]. An optimal solution of the optimization problem has to satisfy the input and state constraints (3.3c).

The user must provide the C functions `ocp_dim` and `ffct` to describe the general structure and the system dynamics of the optimization problem (3.3), also see Section 3.2. As shown in Listing 3.1, the C function `ocp_dim` is used to define the number of states, control inputs, and number of (terminal) inequality and equality constraints. Note that GRAMPC uses the generic type `typeInt` for integer values. The word size of this integer type can be changed in the header file `grampc_macro.h` within the folder `<grampc_root>/include`. This is particularly advantageous with regard to implementing GRAMPC on embedded hardware.

```
/** OCP dimensions **/
void ocp_dim(typeInt *Nx, typeInt *Nu, typeInt *Np, typeInt *Ng, typeInt *Nh,
             typeInt *NgT, typeInt *NhT, typeUSERPARAM *userparam)
{
  *Nx = 2;  *Nu = 1;  *Np = 0;
  *Nh = 4;  *Ng = 0;
  *NhT = 0; *NgT = 0;
}
```

Listing 3.1: Settings of the general structure of optimization problem (3.3).

The system dynamics (3.3b) are described by the C function `ffct` shown in Listing 3.2. The example is given in explicit ODE form, for which the functions `Mfct` and `Mtrans` for the mass matrix M are not required. Similar to the generic integer type `typeInt`, the data type `typeRNum` is used to adress floating point numbers of different word sizes, e.g. float or double (cf. the header file `grampc_macro.h` included in the folder `<grampc_root>/include`).

```
/** System function f(t,x,u,p,userparam) **/
void ffct(typeRNum *out, ctypeRNum t, ctypeRNum *x, ctypeRNum *u, ctypeRNum *p,
          typeUSERPARAM *userparam)
{
  out[0] = x[1]-0.04*u[0];
  out[1] =     -7.01*u[0];
}
```

Listing 3.2: Formulation of the system dynamics of optimization problem (3.3).

The cost functions in (3.3a) are defined via the functions `lfct` and `Vfct`, cf. Listing 3.3. Note that the input argument `userparam` is used to parametrize the cost functional in a generic way.

```
/** Integral cost l(t,x(t),u(t),p,xdes,udes,userparam) **/
void lfct(typeRNum *out, ctypeRNum t, ctypeRNum *x, ctypeRNum *u, ctypeRNum *p,
          ctypeRNum *xdes, ctypeRNum *udes, typeUSERPARAM *userparam)
{
  ctypeRNum* param = (ctypeRNum*)userparam;
  out[0] = (param[0] * (x[0] - xdes[0]) * (x[0] - xdes[0])
           + param[1] * (x[1] - xdes[1]) * (x[1] - xdes[1])
           + param[2] * (u[0] - udes[0]) * (u[0] - udes[0])) / 2;
}

/** Terminal cost V(T,x(T),p,xdes,userparam) **/
void Vfct(typeRNum *out, ctypeRNum T, ctypeRNum *x, ctypeRNum *p,
          ctypeRNum *xdes, typeUSERPARAM *userparam)
{
  ctypeRNum* param = (ctypeRNum*)userparam;
  out[0] = (param[3] * (x[0] - xdes[0]) * (x[0] - xdes[0])
           + param[4] * (x[1] - xdes[1]) * (x[1] - xdes[1])) / 2;
}
```

Listing 3.3: Formulation of the cost functional of optimization problem (3.3).

Listing 3.4 shows the formulation of the inequality constraints $\boldsymbol{h}(\boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{p}, t) \leq \boldsymbol{0}$ as defined by Equation (3.3c). For the sake of completeness, Listing 3.4 also contains the corresponding functions for equality constraints $\boldsymbol{g}(\boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{p}, t) = \mathbf{0}$, terminal inequality constraints $\boldsymbol{h}_T(\boldsymbol{x}(T), \boldsymbol{p}, T) \leq \mathbf{0}$ as well as terminal equality constraints $\boldsymbol{g}_T(\boldsymbol{x}(T), \boldsymbol{p}, T) = \mathbf{0}$, which are not defined for the ball-on-plate example. Similar to the formulation of the cost functional, the input argument **`userparam`** is used to parametrize the inequality constraints.

```
/** Inequality constraints h(t,x(t),u(t),p,uperparam) <= 0 **/
void hfct(typeRNum *out, ctypeRNum t, ctypeRNum *x, ctypeRNum *u,
          ctypeRNum *p, typeUSERPARAM *userparam)
{
  ctypeRNum* param = (ctypeRNum*)userparam;
  out[0] =  param[5] - x[0];
  out[1] = -param[6] + x[0];
  out[2] =  param[7] - x[1];
  out[3] = -param[8] + x[1];

/** Equality constraints g(t,x(t),u(t),p,uperparam) = 0 **/
void gfct(typeRNum *out, ctypeRNum t, ctypeRNum *x,
          ctypeRNum *u, ctypeRNum *p, typeUSERPARAM *userparam)
{
}

/** Terminal inequality constraints hT(T,x(T),p,uperparam) <= 0 **/
void hTfct(typeRNum *out, ctypeRNum T, ctypeRNum *x,
           ctypeRNum *p, typeUSERPARAM *userparam)
{
}

/** Terminal equality constraints gT(T,x(T),p,uperparam) = 0 **/
void gTfct(typeRNum *out, ctypeRNum T, ctypeRNum *x,
           ctypeRNum *p, typeUSERPARAM *userparam)
{
}
```

Listing 3.4: Formulation of the state constraints of optimization problem (3.3)

The Jacobians of the single functions defined in Listing 3.2–3.4 with respect to state $\boldsymbol{x}$ and control $\boldsymbol{u}$ are required for evaluating the optimality conditions of optimization problem (3.3) within the gradient algorithm, see Section 4.1.2. If applicable, the Jacobians of the above-mentioned functions are also required with respect to the optimization variables $\boldsymbol{p}$ and T. Listing 3.5 shows the corresponding Jacobians for the ball-on-plate example. For the matrix product functions `dfdx_vec`, `dfdu_vec`, and `dhdx_vec`, the pointer to a generic vector `vec` is passed as input argument that corresponds to the adjoint state, respectively a vector that accounts for the Lagrange multiplier and penalty term of state constraints, cf. Section 4.1. Note that `vec` is of appropriate dimension for the respective matrix product function, i.e. of dimension $N_{\boldsymbol{x}}$ or $N_{\boldsymbol{h}}$. A complete C function template and further examples concerning the problem formulation are included in the GRAMPC software package.

```
/** Jacobian df/dx multiplied by vector vec, i.e. (df/dx)^T*vec or vec^T*(df/dx) **/
void dfdx_vec(typeRNum *out, ctypeRNum t, ctypeRNum *x, ctypeRNum *vec,
              ctypeRNum *u, ctypeRNum *p, typeUSERPARAM *userparam)
{
  out[0] = 0;
  out[1] = vec[0];
}

/** Jacobian df/du multiplied by vector vec, i.e. (df/du)^T*vec or vec^T*(df/du) **/
void dfdu_vec(typeRNum *out, ctypeRNum t, ctypeRNum *x, ctypeRNum *vec,
              ctypeRNum *u, ctypeRNum *p, typeUSERPARAM *userparam)
{
  out[0] = -0.04*vec[0]-7.01*vec[1];
}
```

```
/** Gradient dl/dx **/
void dldx(typeRNum *out, ctypeRNum t, ctypeRNum *x, ctypeRNum *u, ctypeRNum *p,
          ctypeRNum *xdes, ctypeRNum *udes, typeUSERPARAM *userparam)
{
  ctypeRNum* param = (ctypeRNum*)userparam;
  out[0] = param[0]*(x[0]-xdes[0]);
  out[1] = param[1]*(x[1]-xdes[1]);
}

/** Gradient dl/du **/
void dldu(typeRNum *out, ctypeRNum t, ctypeRNum *x, ctypeRNum *u, ctypeRNum *p,
          ctypeRNum *xdes, ctypeRNum *udes, typeUSERPARAM *userparam)
{
  out[0] = param[2]*(u[0]-udes[0]);
}

/** Gradient dV/dx **/
void dVdx(typeRNum *out, ctypeRNum T, ctypeRNum *x, ctypeRNum *p,
          ctypeRNum *xdes, typeUSERPARAM *userparam)
{
  ctypeRNum* param = (ctypeRNum*)userparam;
  out[0] = param[3]*(x[0]-xdes[0]);
  out[1] = param[4]*(x[1]-xdes[1]);
}

/** Jacobian dh/dx multiplied by vector vec, i.e. (dh/dx)^T*vec or vec^T*(dg/dx) **/
void dhdx_vec(typeRNum *out, ctypeRNum t, ctypeRNum *x, ctypeRNum *u, ctypeRNum *p,
               ctypeRNum *vec, typeUSERPARAM *userparam)
{
  out[0] = -vec[0]+vec[1];
  out[1] = -vec[2]+vec[3];
}
...
```

Listing 3.5: Jacobians of the system dynamics and inequality constraint.

4 Optimization algorithm and options

This chapter summarizes the optimization scheme that is used to solve the optimization problem (3.1). This includes the basic algorithm as well as the options that can be adjusted by the user. Note that all options as well as their corresponding default values are listed in Table A.2. The setting of the options is detailed in Section 5 for usage in C and Matlab.

4.1 Optimization algorithm

The optimization algorithm of GRAMPC is based on an augmented Lagrangian formulation with an inner projected gradient method as minimization step and an outer multiplier and penalty update. This section gives a brief sketch of the algorithm. Note that a more detailed description is given in [5].

4.1.1 Augmented Lagrangian method

GRAMPC implements the augmented Lagrangian approach to handle the equality and inequality constraints (3.1c) and (3.1d). The constraints are adjoined to the integral cost function using the time-dependent multipliers $\boldsymbol{\mu} = [\boldsymbol{\mu}_{\boldsymbol{g}}^\mathsf{T}, \boldsymbol{\mu}_{\boldsymbol{h}}^\mathsf{T}]^\mathsf{T}$ and penalties $\boldsymbol{c} = [\boldsymbol{c}_{\boldsymbol{g}}^\mathsf{T}, \boldsymbol{c}_{\boldsymbol{h}}^\mathsf{T}]^\mathsf{T}$. Similarly, multipliers $\boldsymbol{\mu}_T = [\boldsymbol{\mu}_{\boldsymbol{g}_T}^\mathsf{T}, \boldsymbol{\mu}_{\boldsymbol{h}_T}^\mathsf{T}]^\mathsf{T}$ and penalties $\boldsymbol{c}_T = [\boldsymbol{c}_{\boldsymbol{g}_T}^\mathsf{T}, \boldsymbol{c}_{\boldsymbol{h}_T}^\mathsf{T}]^\mathsf{T}$ are used for the terminal constraints. Where appropriate, the syntax $\bar{\boldsymbol{\mu}} = (\boldsymbol{\mu}_{\boldsymbol{g}}, \boldsymbol{\mu}_{\boldsymbol{h}}, \boldsymbol{\mu}_{\boldsymbol{g}_T}, \boldsymbol{\mu}_{\boldsymbol{h}_T})$ and $\bar{\boldsymbol{c}} = (\boldsymbol{c}_{\boldsymbol{g}}, \boldsymbol{c}_{\boldsymbol{h}}, \boldsymbol{c}_{\boldsymbol{g}_T}, \boldsymbol{c}_{\boldsymbol{h}_T})$ is used to denote all multipliers and penalties. The algorithm requires a reformulation of the inequality constraints (3.1d) that leads to the transformed functions (see [5] for details)

$$\bar{\boldsymbol{h}}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{p}, t, \boldsymbol{\mu}_{\boldsymbol{h}}, \boldsymbol{c}_{\boldsymbol{h}}) = \mathbf{max}\left\{\boldsymbol{h}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{p}, t), -\boldsymbol{C}_{\boldsymbol{h}}^{-1}\boldsymbol{\mu}_{\boldsymbol{h}}\right\} \tag{4.1a}$$

$$\bar{\boldsymbol{h}}_T(\boldsymbol{x}, \boldsymbol{p}, T, \boldsymbol{\mu}_{\boldsymbol{h}_T}, \boldsymbol{c}_{\boldsymbol{h}_T}) = \mathbf{max}\left\{\boldsymbol{h}_T(\boldsymbol{x}, \boldsymbol{p}, T), -\boldsymbol{C}_{\boldsymbol{h}_T}^{-1}\boldsymbol{\mu}_{\boldsymbol{h}_T}\right\} \tag{4.1b}$$

with the component-wise **max**-function and the diagonal matrix syntax $\boldsymbol{C} = \mathrm{diag}(\boldsymbol{c})$. The augmented Lagrangian function is defined as

$$\bar{J}(\boldsymbol{u}, \boldsymbol{p}, T, \bar{\boldsymbol{\mu}}, \bar{\boldsymbol{c}}; \boldsymbol{x}_0) = \bar{V}(\boldsymbol{x}, \boldsymbol{p}, T, \boldsymbol{\mu}_T, \boldsymbol{c}_T) + \int_0^T \bar{l}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{p}, t, \boldsymbol{\mu}, \boldsymbol{c})\,\mathrm{d}t \tag{4.2}$$

with the augmented terminal cost term

$$\begin{aligned}\bar{V}(\boldsymbol{x}, \boldsymbol{p}, T, \boldsymbol{\mu}_T, \boldsymbol{c}_T) = {} & V(\boldsymbol{x}, \boldsymbol{p}, T) + \boldsymbol{\mu}_{\boldsymbol{g}_T}^\mathsf{T}\,\boldsymbol{g}_T(\boldsymbol{x}, \boldsymbol{p}, T) + \frac{1}{2}\|\boldsymbol{g}_T(\boldsymbol{x}, \boldsymbol{p}, T)\|_{\boldsymbol{C}_{\boldsymbol{g}_T}}^2 \\ & + \boldsymbol{\mu}_{\boldsymbol{h}_T}^\mathsf{T}\,\bar{\boldsymbol{h}}_T(\boldsymbol{x}, \boldsymbol{p}, T, \boldsymbol{\mu}_{\boldsymbol{h}_T}, \boldsymbol{c}_{\boldsymbol{h}_T}) + \frac{1}{2}\|\bar{\boldsymbol{h}}_T(\boldsymbol{x}, \boldsymbol{p}, T, \boldsymbol{\mu}_{\boldsymbol{h}_T}, \boldsymbol{c}_{\boldsymbol{h}_T})\|_{\boldsymbol{C}_{\boldsymbol{h}_T}}^2\end{aligned} \tag{4.3}$$

and the augmented integral cost term

$$\begin{aligned}\bar{l}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{p}, t, \boldsymbol{\mu}, \boldsymbol{c}) = {} & l(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{p}, t) + \boldsymbol{\mu}_{\boldsymbol{g}}^\mathsf{T}\,\boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{p}, t) + \frac{1}{2}\|\boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{p}, t)\|_{\boldsymbol{C}_{\boldsymbol{g}}}^2 \\ & + \boldsymbol{\mu}_{\boldsymbol{h}}^\mathsf{T}\,\bar{\boldsymbol{h}}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{p}, t, \boldsymbol{\mu}_{\boldsymbol{h}}, \boldsymbol{c}_{\boldsymbol{h}}) + \frac{1}{2}\|\bar{\boldsymbol{h}}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{p}, t, \boldsymbol{\mu}_{\boldsymbol{h}}, \boldsymbol{c}_{\boldsymbol{h}})\|_{\boldsymbol{C}_{\boldsymbol{h}}}^2\,.\end{aligned} \tag{4.4}$$

Instead of solving the original problem (3.1), the algorithm solves the max-min-problem

$$\begin{aligned}
\max_{\bar{\boldsymbol{\mu}}} \min_{\boldsymbol{u},\boldsymbol{p},T} \quad & \bar{J}(\boldsymbol{u},\boldsymbol{p},T,\bar{\boldsymbol{\mu}},\bar{\boldsymbol{c}};\boldsymbol{x}_0) && \text{(4.5a)}\\
\text{s.t.} \quad & \boldsymbol{M}\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x},\boldsymbol{u},\boldsymbol{p},t)\,, \quad \boldsymbol{x}(0)=\boldsymbol{x}_0 && \text{(4.5b)}\\
& \boldsymbol{u}(t) \in [\boldsymbol{u}_{\min},\boldsymbol{u}_{\max}]\,, \quad t\in[0,T] && \text{(4.5c)}\\
& \boldsymbol{p} \in [\boldsymbol{p}_{\min},\boldsymbol{p}_{\max}]\,, \quad T\in[T_{\min},T_{\max}]\,, && \text{(4.5d)}
\end{aligned}$$

whereby the augmented Lagrangian function (4.2) is maximized with respect to the multipliers $\bar{\boldsymbol{\mu}}$ and minimized with respect to the controls $\boldsymbol{u}$, the parameters $\boldsymbol{p}$ and the end time $\boldsymbol{T}$. Note that the full set of optimization variables $(\boldsymbol{u},\boldsymbol{p},T)$ is considered in what follows for the sake of completeness. The max-min-problem (4.5) corresponds to the dual problem of (3.1) in the case of $\bar{\boldsymbol{c}}=\boldsymbol{0}$. The maximization step is performed by steepest ascent using the constraint residual as direction and the penalty parameter as step size. See [5] for a detailed description of the augmented Lagrangian algorithm.

4.1.2 Projected gradient method

GRAMPC uses a projected gradient method to solve the inner minimization problem subject to the dynamics (3.1b) as well as the box constraints (3.1e) and (3.1f). The algorithm is based on the first-order optimality conditions that can be compactly stated using the Hamiltonian

$$H(\boldsymbol{x},\boldsymbol{u},\boldsymbol{p},\boldsymbol{\lambda},t,\boldsymbol{\mu},\boldsymbol{c}) = \bar{l}(\boldsymbol{x},\boldsymbol{u},\boldsymbol{p},t,\boldsymbol{\mu},\boldsymbol{c}) + \boldsymbol{\lambda}^{\mathsf{T}}\boldsymbol{f}(\boldsymbol{x},\boldsymbol{u},\boldsymbol{p},t) \tag{4.6}$$

with the adjoint states $\boldsymbol{\lambda}$. The canonical equations are then given by

$$\begin{aligned}
\boldsymbol{M}\dot{\boldsymbol{x}} &= \boldsymbol{f}(\boldsymbol{x},\boldsymbol{u},\boldsymbol{p},t)\,, & \boldsymbol{x}(0) &= \boldsymbol{x}_0\,, && \text{(4.7a)}\\
\boldsymbol{M}^{\mathsf{T}}\dot{\boldsymbol{\lambda}} &= -H_{\boldsymbol{x}}(\boldsymbol{x},\boldsymbol{u},\boldsymbol{p},\boldsymbol{\lambda},t,\boldsymbol{\mu},\boldsymbol{c})\,, & \boldsymbol{M}^{\mathsf{T}}\boldsymbol{\lambda}(T) &= \bar{V}_{\boldsymbol{x}}(\boldsymbol{x}(T),\boldsymbol{p},T,\boldsymbol{\mu}_T,\boldsymbol{c}_T) && \text{(4.7b)}
\end{aligned}$$

consisting of the original system (4.7a) and the adjoint system (4.7b). The canonical equations can be iteratively solved in forward and backward time for given initial values of the optimization variables. In each iteration and depending on the optimization variables of the actual problem to be solved, the gradients

$$\begin{aligned}
\boldsymbol{d}_{\boldsymbol{u}} &= H_{\boldsymbol{u}}(\boldsymbol{x},\boldsymbol{u},\boldsymbol{p},\boldsymbol{\lambda},t,\boldsymbol{\mu},\boldsymbol{c}) && \text{(4.8a)}\\
\boldsymbol{d}_{\boldsymbol{p}} &= \bar{V}_{\boldsymbol{p}}(\boldsymbol{x}(T),\boldsymbol{p},T,\boldsymbol{\mu}_T,\boldsymbol{c}_T) + \int_0^T H_{\boldsymbol{p}}(\boldsymbol{x},\boldsymbol{u},\boldsymbol{p},\boldsymbol{\lambda},t,\boldsymbol{\mu},\boldsymbol{c})\,\mathrm{d}t && \text{(4.8b)}\\
d_T &= \bar{V}_T(\boldsymbol{x}(T),\boldsymbol{p},T,\boldsymbol{\mu}_T,\boldsymbol{c}_T) + H(\boldsymbol{x}(T),\boldsymbol{u}(T),\boldsymbol{p},\boldsymbol{\lambda}(T),T,\boldsymbol{\mu}(T),\boldsymbol{c}(T)) && \text{(4.8c)}
\end{aligned}$$

with respect to the controls $\boldsymbol{u}$, parameters $\boldsymbol{p}$, and end time T are used to formulate a line search problem

$$\min_{\alpha} \bar{J}\left(\boldsymbol{\psi}_{\boldsymbol{u}}(\boldsymbol{u}-\alpha\boldsymbol{d}_{\boldsymbol{u}}),\boldsymbol{\psi}_{\boldsymbol{p}}(\boldsymbol{p}-\gamma_{\boldsymbol{p}}\alpha\boldsymbol{d}_{\boldsymbol{p}}),\psi_T(T-\gamma_T\alpha d_T);\bar{\boldsymbol{\mu}},\bar{\boldsymbol{c}},\boldsymbol{x}_0\right) \tag{4.9}$$

with projection functions $\boldsymbol{\psi}_{\boldsymbol{u}}$, $\boldsymbol{\psi}_{\boldsymbol{p}}$ and ψ_T and, finally, to update the optimization variables according to

$$\boldsymbol{u} \leftarrow \boldsymbol{\psi}_{\boldsymbol{u}}(\boldsymbol{u}-\alpha\boldsymbol{d}_{\boldsymbol{u}})\,, \quad \boldsymbol{p} \leftarrow \boldsymbol{\psi}_{\boldsymbol{p}}(\boldsymbol{p}-\gamma_{\boldsymbol{p}}\alpha\boldsymbol{d}_{\boldsymbol{p}})\,, \quad T \leftarrow \psi_T(T-\gamma_T\alpha d_T)\,. \tag{4.10}$$

See [6, 9, 5] for a detailed description of the projected gradient algorithm. GRAMPC provides two methods for the approximate solution of the line search problem, which are explained in Section 4.3.

Algorithm 1 Basic algorithmic structure of GRAMPC.

```
Optional: Shift trajectories by sampling time $\Delta t$                      ▷ cf. Section 4.7
For $i = 1$ to $i_{\max}$ do                                 ▷ Outer multiplier loop for maximization
    For $j = 1$ to $j_{\max}$ do                              ▷ Inner gradient loop for minimization
        If $i > 1$ and $j = 1$ then
            Set $\boldsymbol{x}^{i|j} = \boldsymbol{x}^{i-1}$
        else
            Compute $\boldsymbol{x}^{i|j}$ by forward time integration of system dynamics   ▷ cf. Section 4.2
            Evaluate all constraints
        End If
        Compute $\boldsymbol{\lambda}^{i|j}$ by backward time integration of adjoint system   ▷ cf. Section 4.2
        Evaluate gradients $\boldsymbol{d}_{\boldsymbol{u}}^{i|j}$, $\boldsymbol{d}_{\boldsymbol{p}}^{i|j}$, and $d_T^{i|j}$
        Solve line search problem to determine step size $\alpha^{i|j}$            ▷ cf. Section 4.3
        Update controls $\boldsymbol{u}^{i|j+1}$, parameters $\boldsymbol{p}^{i|j+1}$, and end time $\boldsymbol{T}^{i|j+1}$
        If minimization is converged then                              ▷ cf. Section 4.5
            Break inner loop
        End If
    End For
    Set $\boldsymbol{u}^i = \boldsymbol{u}^{i|j+1}$, $\boldsymbol{p}^i = \boldsymbol{p}^{i|j+1}$, and $T^i = T^{i|j+1}$
    Compute $\boldsymbol{x}^i$ by forward time integration of system dynamics          ▷ cf. Section 4.2
    Evaluate all constraints
    Update multipliers $\bar{\boldsymbol{\mu}}^{i+1}$ and penalties $\bar{\boldsymbol{c}}^{i+1}$                  ▷ cf. Section 4.4
    If minimization is converged & constraint thresholds are satisfied then   ▷ cf. Section 4.5
        Break outer loop
    End If
End For
Compute cost $J$ and norm of constraints
```

4.1.3 Algorithmic structure

The basic structure of the algorithm that is implemented in the main calling function `grampc_run` is outlined in Algorithm 1. The projected gradient method is realized in the inner loop and consists of the forward and backward integration of the canonical equations as well as the update of the optimization variables based on the gradient and the approximate solution of the line search problem. The outer loop corresponds to the augmented Lagrangian method consisting of the solution of the inner minimization problem and the update of the multipliers and penalty parameters.

As an alternative to the augmented Lagrangian framework, the user can choose external penalty functions in GRAMPC that handle the equality and inequality constraints as "soft" constraints. In this case, the multipliers $\bar{\boldsymbol{\mu}}$ are fixed at zero and only the penalty parameters are updated in the outer loop. Note that the user can set the options `PenaltyIncreaseFactor` and `PenaltyDecreaseFactor` to 1.0 in order to keep the penalty parameters at the initial value `PenaltyMin`. The single steps of the algorithm and the related options are described in more detail in the following sections.

The following options can be used to adjust the basic algorithm. The corresponding default values are listed in Table A.2 in the appendix.

- `MaxMultIter`: Sets the maximum number of augmented Lagrangian iterations $i_{\max} \geq 1$. If the option `ConvergenceCheck` is activated, the algorithm evaluates the convergence criterion and terminates if the inner minimization converged and all constraints are satisfied within the tolerance defined by `ConstraintsAbsTol`.

- `MaxGradIter`: If the option `ConvergenceCheck` is activated, the algorithm terminates the inner loop as soon as the convergence criterion is fulfilled.
- `EqualityConstraints`: Equality constraints $\boldsymbol{g}(\boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{p}, t) = \mathbf{0}$ can be disabled by the option value `off`.
- `InequalityConstraints`: To disable inequality constraints $\boldsymbol{h}(\boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{p}, t) \leq \mathbf{0}$, set this option to `off`.
- `TerminalEqualityConstraints`: To disable terminal equality constraints $\boldsymbol{g}_T(\boldsymbol{x}(T), \boldsymbol{p}, T) = \mathbf{0}$, set this option to `off`.
- `TerminalInequalityConstraints`: To disable terminal inequality constraints $\boldsymbol{h}_T(\boldsymbol{x}(T), \boldsymbol{p}, T) \leq \mathbf{0}$, set this option to `off`.
- `ConstraintsHandling`: State constraints are handled either by means of the augmented Lagrangian approach (option value `auglag`) or as soft constraints by outer penalty functions (option value `extpen`).
- `OptimControl`: Specifies whether the cost functional should be minimized with respect to the control variable $\boldsymbol{u}$.
- `OptimParam`: Specifies whether the cost functional should be minimized with respect to the optimization parameters $\boldsymbol{p}$.
- `OptimTime`: Specifies whether the cost functional should be minimized with respect the horizon length T (free end time problem) or if T is kept constant.

4.2 Numerical Integration

GRAMPC employs a continuous-time formulation of the optimization problem. However, internally, all time-dependent functions are stored in discretized form with N_{hor} elements and numerical integration is performed to compute the cost functional and to solve the differential equations.

4.2.1 Integration of cost functional and explicit ODEs

The approximate line search method requires the evaluation of the cost functional. To this end, the integral cost is integrated numerically with either the trapezodial or the Simpson rule and the terminal cost is added. The cost values are additionally evaluated at the end of the optimization as an add-on information for the user.

The gradient algorithm involves the sequential forward integration of the system dynamics and backward integration of the adjoint dynamics. GRAMPC implements three integrators with fixed step size (Euler, modified Euler, and Heun). Furthermore, a 4th-order Runge-Kutta m ethod and a semi-implicit Rosenbrock solver (RODAS [7, 12]) with variable step size are available.

The following options can be used to adjust the numerical integrations:

- `Nhor`: Number of discretization points within the time interval $[0, T]$.
- `IntegralCost`, `TerminalCost`: Indicate if the integral and/or terminal cost functions are defined.
- `IntegratorCost`: This option specifies the integration scheme for the cost functionals. Possible values are `trapezodial` and `simpson`.

- `Integrator`: This option specifies the integration scheme for the system and adjoint dynamics. Possible values are `euler`, `modeuler` and `heun` with fixed step size and `ruku45` and `rodas` with variable step size.
- `IntegratorMinStepSize`: Minimum step size for RODAS and the Runge-Kutta integrator.
- `IntegratorMaxSteps`: Maximum number of steps for RODAS and the Runge-Kutta integrator.
- `IntegratorRelTol`: Relative tolerance for RODAS and the Runge-Kutta integrator with variable step size. Note that this option may be insignificant if the minimum step size is chosen too high or the maximum number of steps is set too low.
- `IntegratorAbsTol`: Absolute tolerance for RODAS and the Runge-Kutta integrator with variable step size. Note that this option may be insignificant if the minimum step size is chosen too high or the maximum number of steps is set too low.

4.2.2 Integration of semi-implicit ODEs and DAEs (RODAS)

GRAMPC supports problem descriptions with ordinary differential equations in semi-implicit form as well as differential algebraic equations using the solver RODAS for numerical integration (see Section 3.2.2). If a semi-implicit problem or differential algebraic equations are considered, the mass matrix $\boldsymbol{M}$ and its transposed version $\boldsymbol{M}^\mathsf{T}$ must be defined by the C functions `Mfct` and `Mtrans`. The numerical integration can be accelerated by additionally providing the C functions `dfdx`, `dfdxtrans`, `dfdt` and `dHdxdt`. See Section 3.2 for a detailed description of the problem implementation.

The integration with RODAS is configured by a number of flags that are passed to the solver using the vector `FlagsRodas` with the elements [`IFCN`, `IDFX`, `IJAC`, `IMAS`, `MLJAC`, `MUJAC`, `MLMAS`, `MUMAS`]. See [12, 7] for a detailed description of these flags. The default values $[0, 0, 0, 0, N_x, N_x, N_x, N_x]$ correspond to an autonomous system with an identity matrix as mass matrix. The following options can be adjusted via `FlagsRodas`:

- `IFCN`: Specifies if the right hand side of the system dynamics $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{p}, T)$ explicitly depends on time t (`IFCN`=1) or if the problem is autonomous (`IFCN`=0).
- `IDFX`: Specifies how the computation of the partial derivatives $\frac{\partial \boldsymbol{f}}{\partial t}$ and $\frac{\partial^2 H}{\partial x \partial t}$ is carried out. The partial derivatives are computed internally by finite differences (`IDFX`=0) or are provided by the functions `dfdt` and `dHdxdt` (`IDFX`=1) as described in Section 3.2.
- `IJAC`: Specifies how the computation of the Jacobians $\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}$ and $(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}})^\mathsf{T} = \frac{\partial^2 H}{\partial x \partial \lambda}$ is carried out for numerically solving the canonical equations. The Jacobians are computed internally by finite differences (`IJAC`=0) or are provided by the functions `dfdx` and `dfdxtrans` (`IJAC`=1), also see Section 3.2.
- `IMAS`: Gives information on the mass matrix $\boldsymbol{M}$, which is either an identity matrix (`IMAS`=0) or is specified by the function `Mfct` (`IMAS`=1). Note that the adjoint dynamics requires the transposed mass matrix that has to be provided by the function `Mtrans`.
- `MLJAC`: Gives information on the banded structure of the Jacobian $\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}$ and $(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}})^\mathsf{T}$, respectively. The Jacobian is either a full matrix (`MLJAC`=N_x) or is of banded structure. In the latter case, the number of non-zero diagonals below the main diagonal are specified by $0 \leq$ `MLJAC` $< N_x$.
- `MUJAC`: Specifies the number of non-zero diagonals above the main diagonal of the Jacobian $\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}$. This flag needs not to to be defined if `MLJAC`=N_x. Since the partial derivative of the right hand side of the adjoint dynamics with respect to the adjoint state $\boldsymbol{\lambda}$ is given by $(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}})^\mathsf{T}$, the meaning of the flags `MLJAC` and `MUJAC` switches in this case.

- `MLMAS` and `MUMAS`: Both options have the same meaning as `MLJAC` and `MUJAC`, but refer to the mass matrix $\boldsymbol{M}$.

If a semi-implicit problem (option `IMAS`=1) with Mayer term (option `TerminalCost=on`) is considered, the terminal conditions of the adjoint system must be specified in a specific form. To provide RODAS [7, 12] the proper terminal condition $\boldsymbol{\lambda}(T)$, the function `dVdx` must be specified as follows

$$\boldsymbol{\lambda}(T) = \underbrace{\left(\boldsymbol{M}^{\mathsf{T}}\right)^{-1} \bar{V}_{\boldsymbol{x}}(\boldsymbol{x}(T), \boldsymbol{p}, T, \boldsymbol{\mu}_T, \boldsymbol{c}_T)}_{\texttt{dVdx}} \tag{4.11}$$

cf. Equation (4.7b). In the case of a DAE system, the mass matrix is singular and therefore only the elements of the mass matrix for the differential equations are inverted. For an example of using RODAS and the respective options, take a look at the MPC problem `Reactor_PDE` in the folder `<grampc_root>/examples`.

4.3 Line search

The projected gradient method as part of the GRAMPC algorithm (Table 1) requires the solution of the line search problem [5]

$$\min_{\alpha} \bar{J}\left(\boldsymbol{\psi}_{\boldsymbol{u}}\left(\boldsymbol{u}^{i|j} - \alpha \boldsymbol{d}_{\boldsymbol{u}}^{i|j}\right), \boldsymbol{\psi}_{\boldsymbol{p}}\left(\boldsymbol{p}^{i|j} - \gamma_{\boldsymbol{p}}\alpha \boldsymbol{d}_{\boldsymbol{p}}^{i|j}\right), \psi_T\left(T^{i|j} - \gamma_T \alpha d_T^{i|j}\right); \bar{\boldsymbol{\mu}}, \bar{\boldsymbol{c}}, \boldsymbol{x}_0\right) . \tag{4.12}$$

GRAMPC implements two efficient strategies to solve this problem in an approximate manner. The following options apply to both line search methods that are detailed below (Section 4.3.1 and 4.3.2):

- `LineSearchType`: This option selects either the adaptive line search strategy (value `adaptive`) or the explicit approach (value `explicit1` or `explicit2`).
- `LineSearchMax`: This option sets the maximum value $\alpha_{\max}$ of the step size α.
- `LineSearchMin`: This option sets the minimum value $\alpha_{\min}$ of the step size α.
- `LineSearchInit`: Indicates the initial value $\alpha_{\text{init}} > 0$ for the step size α. If the adaptive line search is used, the sample point α_2 is set to$\alpha_2 = \alpha_{\text{init}}$.
- `OptimParamLineSearchFactor`: This option sets the adaptation factor $\gamma_{\boldsymbol{p}}$ that weights the update of the parameter vector $\boldsymbol{p}$ against the update of the control $\boldsymbol{u}$.
- `OptimTimeLineSearchFactor`: This option sets the adaptation factor γ_T that weights the update of the end time T against the update of the control $\boldsymbol{u}$.

4.3.1 Adaptive line search

An appropriate way to determine the step size is the adaptive line search approach from [6], where a polynomial approximation of the cost (4.2) is used and an adaptation of the search intervals is performed. More precisely, the cost functional is evaluated at three sample points $\alpha_1 < \alpha_2 < \alpha_3$ with $\alpha_2 = \frac{1}{2}(\alpha_1 + \alpha_3)$, which are used to construct a quadratic polynomial of the cost according to

$$\begin{aligned} &\bar{J}\left(\boldsymbol{\psi}_{\boldsymbol{u}}\left(\boldsymbol{u}^{i|j} - \alpha \boldsymbol{d}_{\boldsymbol{u}}^{i|j}\right), \boldsymbol{\psi}_{\boldsymbol{p}}\left(\boldsymbol{p}^{i|j} - \gamma_{\boldsymbol{p}}\alpha \boldsymbol{d}_{\boldsymbol{p}}^{i|j}\right), \psi_T\left(T^{i|j} - \gamma_T \alpha d_T^{i|j}\right); \bar{\boldsymbol{\mu}}, \bar{\boldsymbol{c}}, \boldsymbol{x}_0\right) \\ &\qquad\qquad\qquad\qquad \approx \Phi(\alpha) = p_0 + p_1\alpha + p_2\alpha_2 \,. \end{aligned} \tag{4.13}$$

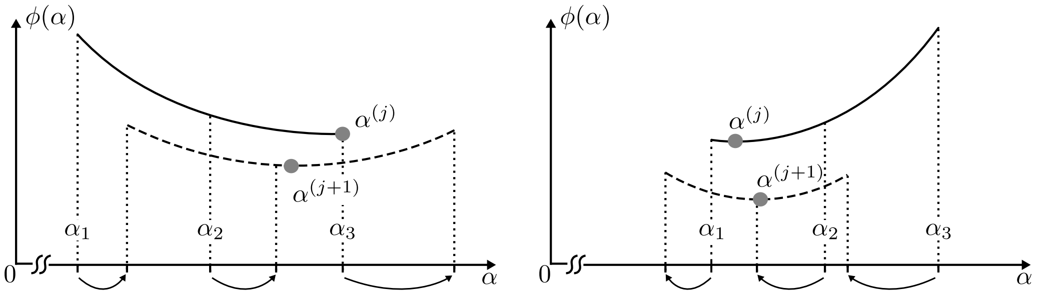

Figure 4.1: Adaptation of line search interval.

Subsequently, a step size α^j is computed by minimizing the cost approximation (4.13). If necessary, the interval $[\alpha_1, \alpha_3]$ is adapted for the next gradient iteration in the following way

$$[\alpha_1, \alpha_3] \leftarrow \begin{cases} \kappa\,[\alpha_1, \alpha_3] & \text{if } \alpha \geq \alpha_3 - \varepsilon_\alpha(\alpha_3 - \alpha_1) \text{ and } \alpha_3 \leq \alpha_{\max} \text{ and } |\Phi(\alpha_1) - \Phi(\alpha_3)| > \varepsilon_\phi \\ \frac{1}{\kappa}\,[\alpha_1, \alpha_3] & \text{if } \alpha \leq \alpha_1 + \varepsilon_\alpha(\alpha_3 - \alpha_1) \text{ and } \alpha_1 \geq \alpha_{\min} \text{ and } |\Phi(\alpha_1) - \Phi(\alpha_3)| > \varepsilon_\phi \\ [\alpha_1, \alpha_3] & \text{otherwise} \end{cases} \tag{4.14a}$$

$$\alpha_2 \leftarrow \frac{1}{2}(\alpha_1 + \alpha_3) \tag{4.14b}$$

with the adaptation factor $\kappa > 1$, the interval tolerance $\varepsilon_\alpha \in (0, 0.5)$, the absolute cost tolerance $\varepsilon_\phi \in [0, \infty)$ for adapting the interval and the interval bounds $\alpha_{\max} > \alpha_{\min} > 0$. The modification (4.14) of the line search interval tracks the minimum point α^j of the line search problem in the case when α^j is either outside of the interval $[\alpha_1, \alpha_3]$ or close to one of the outer bounds α_1, α_3, as illustrated in Figure 4.1. The adaptation factor κ accounts for scaling as well as shifting of the interval $[\alpha_1, \alpha_3]$ in the next gradient iteration, if α^j lies in the vicinity of the interval bounds $[\alpha_1, \alpha_3]$ as specified by the interval tolerance ε_α. This adaptive strategy allows one to track the minimum of the line search problem (4.12) over the gradient iterations j and MPC steps k, while guaranteeing a fixed number of operations in view of a real-time MPC implementation. The absolute tolerance ε_ϕ of the difference in the (scaled) costs at the interval bounds $|\Phi(\alpha_1) - \Phi(\alpha_3)|$ avoids oscillations of the interval width in regions where the cost function $\bar{J}$ is almost constant.

The following options apply specifically to the adaptive line search strategy:

- `LineSearchAdaptAbsTol`: This option sets the absolute tolerance ε_ϕ of the difference in costs at the interval bounds α_1 and α_2. If the difference in the (scaled) costs on these bounds falls below ε_ϕ, the adaption of the interval is stopped in order to avoid oscillations.
- `LineSearchAdaptFactor`: This option sets the adaptation factor $\kappa > 1$ in (4.14) that determines how much the line search interval can be adapted from one gradient iteration to the next.
- `LineSearchIntervalTol`: This option sets the interval tolerance $\varepsilon_\alpha \in (0, 0.5)$ in (4.14) that determines for which values of α the adaption is performed.
- `LineSearchIntervalFactor`: This option sets the interval factor $\beta \in (0, 1)$ that specifies the interval bounds $[\alpha_1, \alpha_3]$ according to $\alpha_1 = \alpha_2(1 - \beta)$ and $\alpha_3 = \alpha_2(1 + \beta)$, whereby the mid sample point is initialized as $\alpha_2 = \alpha_{\text{init}}$.

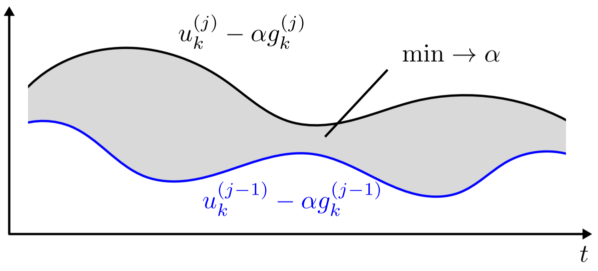
$u_k^{(j)} - \alpha g_k^{(j)}$
$\min \to \alpha$
$u_k^{(j-1)} - \alpha g_k^{(j-1)}$
t

Figure 4.2: Motivation for the explicit line search strategy.

4.3.2 Explicit line search

An alternative way to determine the step size in order to further reduce the computational effort for time-critical problems is the explicit line search approach originally discussed in [1] and adapted in [8] for the optimal control case. The motivation is to minimize the difference between two consecutive control updates $u_k^{i|j}(\tau)$ and $u_k^{i|j+1}(\tau)$ in the unconstrained case and additionally assuming the same step size $\alpha^{i|j}$, i.e.

$$\begin{aligned}\alpha^{i|j} &= \arg\min_{\alpha>0} \left\|\boldsymbol{u}^{i|j+1} - \boldsymbol{u}^{i|j}\right\|^2_{L^2_m[0,T]} + \left\|\boldsymbol{p}^{i|j+1} - \boldsymbol{p}^{i|j}\right\|^2_2 + \left|T^{i|j+1} - T^{i|j}\right| \\ &= \arg\min_{\alpha>0} \Bigg\|\underbrace{\boldsymbol{u}^{i|j} - \boldsymbol{u}^{j-1}}_{=:\Delta \boldsymbol{u}^{i|j}} - \alpha \underbrace{\left(\boldsymbol{d}_{\boldsymbol{u}}^{i|j} - \boldsymbol{d}_{\boldsymbol{u}}^{j-1}\right)}_{=:\Delta \boldsymbol{d}_{\boldsymbol{u}}^{i|j}}\Bigg\|^2_{L^2_m[0,T]} + \Bigg\|\underbrace{\boldsymbol{p}^{i|j} - \boldsymbol{p}^{j-1}}_{=:\Delta \boldsymbol{p}^{i|j}} - \gamma_{\boldsymbol{p}}\alpha \underbrace{\left(\boldsymbol{d}_{\boldsymbol{p}}^{i|j} - \boldsymbol{d}_{\boldsymbol{p}}^{j-1}\right)}_{=:\Delta \boldsymbol{d}_{\boldsymbol{p}}^{i|j}}\Bigg\|^2_2 \\ &\quad + \Bigg\|\underbrace{T^{i|j} - T^{j-1}}_{=:\Delta T^{i|j}} - \gamma_T \alpha \underbrace{\left(d_T^{i|j} - d_T^{j-1}\right)}_{=:\Delta d_T^{i|j}}\Bigg\|^2_2\end{aligned} \tag{4.15}$$

with $||z||^2_{L^2_m[0,T]} = \langle z, z\rangle := \int_0^T z^\mathsf{T}(t)z(t)\mathrm{d}t$. Figure 4.2 illustrates the general idea behind (4.15). To solve (4.15), consider the following function

$$\begin{aligned}q(\alpha) :&= \left\|\Delta u_k^j - \alpha \Delta d_k^j\right\|^2_{L^2_m[0,T]} \\ &= \int_0^T \left(\Delta u_k^j - \alpha\Delta d_k^j\right)^\mathsf{T} \left(\Delta u_k^j - \alpha\Delta d_k^j\right)\,\mathrm{d}t \\ &= \int_0^T \left(\Delta u_k^j\right)^\mathsf{T} \Delta u_k^j\,\mathrm{d}t + \alpha^2 \int_0^T \left(\Delta d_k^j\right)^\mathsf{T} \Delta d_k^j\,\mathrm{d}t - 2\alpha \int_0^T \left(\Delta u_k^j\right)^\mathsf{T} \Delta d_k^j\,\mathrm{d}t.\end{aligned} \tag{4.16}$$

The minimum has to satisfy the stationarity condition

$$\frac{\partial q(\alpha)}{\partial \alpha} = 2\alpha \int_0^T \left(\Delta d_k^j\right)^\mathsf{T} \Delta d_k^j\,\mathrm{d}t - 2\int_0^T \left(\Delta u_k^j\right)^\mathsf{T} \Delta d_k^j\,\mathrm{d}t = 0. \tag{4.17}$$

A suitable step size α^j then follows to

$$\alpha^j = \frac{\int_0^T \left(\Delta u_k^j\right)^\mathsf{T} \Delta d_k^j\,\mathrm{d}t}{\int_0^T \left(\Delta d_k^j\right)^\mathsf{T} \Delta d_k^j\,\mathrm{d}t} = \frac{\langle \Delta u_k^j, \Delta d_k^j\rangle}{\langle \Delta d_k^j, \Delta d_k^j\rangle}\,. \tag{4.18}$$

Another way to compute an appropriate step size for the control update can be achieved by reformulating (4.16) in the following way:

$$q(\alpha) = \left\| \Delta u_k^j - \alpha \Delta d_k^j \right\|^2_{L^2_m[0,T]} = \alpha^2 \left\| \frac{1}{\alpha} \Delta u_k^j - \Delta d_k^j \right\|^2_{L^2_m[0,T]} =: \alpha^2 \bar{q}(\alpha). \tag{4.19}$$

In the subsequent, the new function $\bar{q}(\alpha)$ is minimized w.r.t. the step size leading to a similar solution

$$\alpha^j = \frac{\langle \Delta u^j, \Delta u^j \rangle}{\langle \Delta u^j, \Delta d_k^j \rangle} . \tag{4.20}$$

For the original problem (4.15), the solution

$$\alpha^{i|j} = \frac{\langle \Delta \boldsymbol{u}^{i|j}, \Delta \boldsymbol{d}_{\boldsymbol{u}}^{i|j} \rangle + \gamma_{\boldsymbol{p}} \langle \Delta \boldsymbol{p}^{i|j}, \Delta \boldsymbol{d}_{\boldsymbol{p}}^{i|j} \rangle + \gamma_T \Delta T^{i|j} \Delta d_T^{i|j}}{\langle \Delta \boldsymbol{d}_{\boldsymbol{u}}^{i|j}, \Delta \boldsymbol{d}_{\boldsymbol{u}}^{i|j} \rangle + \gamma_{\boldsymbol{p}}^2 \langle \Delta \boldsymbol{d}_{\boldsymbol{p}}^{i|j\,\mathsf{T}} \Delta \boldsymbol{d}_{\boldsymbol{p}}^{i|j} \rangle + \gamma_T^2 \big(\Delta d_T^{i|j}\big)^2} \tag{4.21}$$

$$\alpha^{i|j} = \frac{\langle \Delta \boldsymbol{u}^{i|j}, \Delta \boldsymbol{u}^{i|j} \rangle + \gamma_{\boldsymbol{p}} \langle \Delta \boldsymbol{p}^{i|j}, \Delta \boldsymbol{p}^{i|j} \rangle + \gamma_T \big(\Delta T^{i|j}\big)^2}{\langle \Delta \boldsymbol{u}^{i|j}, \Delta \boldsymbol{d}_{\boldsymbol{u}}^{i|j} \rangle + \gamma_{\boldsymbol{p}}^2 \langle \Delta \boldsymbol{p}^{i|j}, \Delta \boldsymbol{d}_{\boldsymbol{p}}^{i|j} \rangle + \gamma_T^2 \Delta T^{i|j} \Delta d_T^{i|j}} \tag{4.22}$$

follows.

In the GRAMPC implementation, both approaches (4.21) and (4.22) are available. In addition, the step size α^j is bounded by the upper and lower values $\alpha_{\max} > \alpha_{\min} > 0$. However, if the originally computed step size α^j is less than zero[1], either the initial step size $\alpha^j = \alpha_{\text{init}}$ or the automatic fallback strategy that is detailed in the next subsection is used in order to achieve a valid step size. The fallback strategy is set with the following option (only available for the explicit line search strategies):

- `LineSearchExpAutoFallback`: If this option is activated, the automatic fallback strategy is used in the case that the explicit formulas result in negative step sizes.

4.3.3 Fallback strategy for explicit line search

While the initial step size can be used as fallback solution if the explicit step size computation yields negative values, it often requires problem-specific tuning of α_{init} for achieving optimal performance. As alternative, GRAMPC implements an automatic fallback strategy that is based on the idea of using at most 1% of the control range defined by `umax` and `umin`. For this purpose, the maximum absolute value $\boldsymbol{d}_{\boldsymbol{u},\max}^{i|j} = \|\boldsymbol{d}_{\boldsymbol{u}}^{i|j}(t)\|_{L^\infty}$ of the search direction $\boldsymbol{d}_{\boldsymbol{u}}^{i|j}(t)$ over the horizon is determined. Subsequently, the step size

$$\alpha^{i|j} = \frac{1}{100} \cdot \min_{k \in \{1,\dots,N_{\boldsymbol{u}}\}} \left\{ \frac{u_{\max,k} - u_{\min,k}}{d_{\boldsymbol{u},\max,k}^{i|j}} \right\} \tag{4.23}$$

follows as the minimal step size required to perform a step of 1% with respect to the range of at least one control in at least one time step. Additionally, the step size is limited to 10% of the maximum step size $\alpha_{\max}$. Since this strategy requires reasonable limits for the controls, it is only executed if `LineSearchExpAutoFallback` is activated and if these limits are defined by the user. Furthermore, this strategy can only be used if `OptimControl` is switched `on`. In all other case, the initial step size $\alpha^j = \alpha_{\text{init}}$ will be used as fallback solution.

[1]It can be shown in the one-dimensional case that the step size α is negative if the cost function is locally non-convex.

4.4 Update of multipliers and penalties

GRAMPC handles general nonlinear constraints using an augmented Lagrangian approach or, alternatively, using an external penalty method. The key to the efficient solution of constrained problems using these approaches are the updates of the multipliers in the outer loop for $i = 1, \ldots, i_{\max}$.

4.4.1 Update of Lagrangian multipliers

The outer loop of the GRAMPC algorithm in Table 1 maximizes the augmented Lagrangian function with respect to the multipliers $\bar{\boldsymbol{\mu}}$. This update is carried out in the direction of steepest ascent that is given by the constraint residual. The penalty parameter is used as step size, as it is typically done in augmented Lagrangian methods.

For an arbitrary equality constraint $g^i = g(\boldsymbol{x}^i, \dots)$ with multiplier μ_g^i, penalty c_g^i, and tolerance ε_g, the update is defined by

$$\mu_g^{i+1} = \zeta_g(\mu_g^i, c_g^i, g^i, \varepsilon_g) = \begin{cases} \mu^i + (1-\rho)c_g^i g^i & \text{if } \left|g^i\right| > \varepsilon_g \wedge \eta^i \leq \varepsilon_{\text{rel,u}} \\ \mu_g^i & \text{else}\,. \end{cases} \tag{4.24}$$

The update is not performed if the constraint is satisfied within its tolerance ε or if the inner minimization is not sufficiently converged, which is checked by the maximum relative gradient η^i (see (4.31) for the definition) and the threshold $\varepsilon_{\text{rel,u}}$. Similarly, for an inequality constraint $h^i = h(\boldsymbol{x}^i, \dots)$ with multiplier μ_h^i, penalty c_h^i, and tolerance ε_h, the update is defined by

$$\mu_h^{i+1} = \zeta_h(\mu_h^i, c_h^i, \bar{h}^i, \varepsilon_h) = \begin{cases} \mu_h^i + (1-\rho)c_h^i \bar{h}^i & \text{if } \left(\bar{h}^i > \varepsilon_h \wedge \eta^i \leq \varepsilon_{\text{rel,u}}\right) \vee \bar{h}^i < 0 \\ \mu_h^i & \text{else}\,. \end{cases} \tag{4.25}$$

Similar update rules are used for the terminal equality and terminal inequality constraints.

GRAMPC provides several means to increase the robustness of the multiplier update, which may be required if few iterations $j_{\max}$ are used for the suboptimal solution of the inner minimization problem. The damping factor $\rho \in [0, 1)$ can be used to scale the step size of the steepest ascent and the tolerance $\varepsilon_{\text{rel,u}}$ can be used to skip the multiplier update in case that the minimization is not sufficiently converged. Furthermore, the multipliers are limited by lower and upper bounds $\mu_g \in [-\mu_{\max}, \mu_{\max}]$ for equalities and $\mu_h \leq \mu_{\max}$ for inequalities, respectively, to avoid unlimited growth. A status flag is set if one of the multipliers reaches this bound and the user should check the problem formulation as this case indicates an ill-posed or even infeasible optimization problem.

The following options can be used to adjust the update of the Lagrangian multipliers:

- `MultiplierMax`: Upper bound $\mu_{\max}$ and lower bound $-\mu_{\max}$ for the Lagrangian multpliers.
- `MultiplierDampingFactor`:Damping factor $\rho \in [0, 1)$ for the multiplier update.
- `AugLagUpdateGradientRelTol`: Threshold $\varepsilon_{\text{rel,u}}$ for the maximum relative gradient of the inner minimization problem.
- `ConstraintsAbsTol`: Thresholds $(\boldsymbol{\varepsilon_g}, \boldsymbol{\varepsilon_h}, \boldsymbol{\varepsilon_{g_T}}, \boldsymbol{\varepsilon_{h_T}}) \in \mathbb{R}^{N_c}$ for the equality, inequality, terminal equality, and terminal inequality constraints.

4.4.2 Update of penalty parameters

The penalty parameters $\bar{\boldsymbol{c}}$ are adapted in each outer iteration according to a heuristic rule that is motivated by the LANCELOT package [3, 10]. A carefully tuned adaptation of the penalties can speed-up the convergence significantly and is therefore highly recommended (also see Section 4.4.3 and the tutorial in Section 6.1). Note that the penalty parameters are also updated if external penalties are used instead of the augmented Lagrangian method, i.e. `ConstraintsHandling` is set to `extpen`. In order to keep the penalty parameters at the initial value `PenaltyMin`, the options `PenaltyIncreaseFactor` and `PenaltyDecreaseFactor` can be set to 1.0, which basically deactivates the penalty update.

For an arbitrary equality constraint $g^i = g(\boldsymbol{x}^i, \dots)$ with penalty c_g^i and tolerance ε_g, the update is defined by

$$c_g^{i+1} = \xi_g(c_g^i, g^i, g^{i-1}, \varepsilon_g) = \begin{cases} \beta_{\text{in}}\, c_g^i & \text{if } \left|g^i\right| \geq \gamma_{\text{in}} \left|g^{i-1}\right| \,\wedge\, \left|g^i\right| > \varepsilon_g \,\wedge\, \eta^i \leq \varepsilon_{\text{rel,u}} \\ \beta_{\text{de}}\, c_g^i & \text{else if } \left|g^i\right| \leq \gamma_{\text{de}}\, \varepsilon_g \\ c_g^i & \text{else}\,. \end{cases} \tag{4.26}$$

The penalty c_g^i is increased by the factor $\beta_{\text{in}} > 1$ if the (sub-optimal) solution of the inner minimization problem does not generate sufficient progress in the constraint, which is rated by the factor $\gamma_{\text{in}} > 0$ and compared to the previous iteration $i-1$. This update is skipped if the inner minimization is not sufficiently converged, which is checked by the maximum relative gradient η^i and the threshold $\varepsilon_{\text{rel,u}}$. The penalty c_g^i is decreased by the factor $\beta_{\text{de}} < 1$ if the constraint g^i is sufficiently satisfied within its tolerance, whereby currently the constant factor $\gamma_{\text{de}} = 0.1$ is used. The setting $\beta_{\text{in}} = \beta_{\text{de}} = 1$ can be used to keep the penalty constant, i.e., to deactivate the penalty adaptation. Similarly, for an inequality constraint $\bar{h}^i = \bar{h}(\boldsymbol{x}^i, \dots)$ with penalty c_h^i and tolerance ε_h, the update is defined by

$$c_h^{i+1} = \xi_h(c_h^i, \bar{h}^i, \bar{h}^{i-1}, \varepsilon_h) = \begin{cases} \beta_{\text{in}}\, c_h^i & \text{if } \bar{h}^i \geq \gamma_{\text{in}} \bar{h}^{i-1} \,\wedge\, \bar{h}^i > \varepsilon_h \,\wedge\, \eta^i \leq \varepsilon_{\text{rel,u}} \\ \beta_{\text{de}}\, c_h^i & \text{else if } \bar{h}^i \leq \gamma_{\text{de}}\, \varepsilon_h \\ c_h^i & \text{else}\,. \end{cases} \tag{4.27}$$

Similar update rules are used for the terminal equality and inequality constraints. In analogy to the multiplier update, the penalty parameters are restricted to upper and lower bounds $c_{\max} \gg c_{\min} > 0$ in order to avoid unlimited growth as well as negligible values.

The following options can be used to adjust the update of the penalty parameters:

- `PenaltyMax`: This option sets the upper bound $c_{\max}$ of the penalty parameters.
- `PenaltyMin`: This option sets the lower bound $c_{\min}$ of the penalty parameters.
- `PenaltyIncreaseFactor`: This option sets the factor β_{in} by which penalties are increased.
- `PenaltyDecreaseFactor`: This option sets the factor β_{de} by which penalties are decreased.
- `PenaltyIncreaseThreshold`: This option sets the factor γ_{in} that rates the progress in the constraints between the last two iterates.
- `AugLagUpdateGradientRelTol`: Threshold $\varepsilon_{\text{rel,u}}$ for the maximum relative gradient of the inner minimization problem.
- `ConstraintsAbsTol`: Thresholds $(\boldsymbol{\varepsilon_g}, \boldsymbol{\varepsilon_h}, \boldsymbol{\varepsilon_{g_T}}, \boldsymbol{\varepsilon_{h_T}}) \in \mathbb{R}^{N_c}$ for the equality, inequality, terminal equality, and terminal inequality constraints.

4.4.3 Estimation of minimal penalty parameter

In real-time or embedded MPC applications, where only a limited number of iterations per step is computed, it is crucial that the penalty parameter is not decreased below a certain threshold $c_{\min}$. This lower bound should be large enough that an inactive constraint that becomes active is still sufficiently penalized in the augmented Lagrangian cost functional. However, it should not be chosen too high to prevent ill-conditioning. A suitable value of $c_{\min}$ tailored to the given MPC problem therefore is of importance to ensure a high performance of GRAMPC.

In order to support the user, GRAMPC offers the routine **`grampc_estim_penmin`** to compute a problem-specific estimate of $c_{\min}$. The basic idea behind this estimation is to determine $c_{\min}$ such that the actual costs J are in the same order of magnitude as the squared constraints multiplied by $c_{\min}$, see equation (4.2). This approach requires initial values for the states $\boldsymbol{x}$, controls $\boldsymbol{u}$, and cost J. If the GRAMPC `rws` structure includes only default values, i.e. zeros, (cold start) the estimation function `grampc_estim_penmin` can be called with the argument `rungrampc`=1 to perform one optimization or MPC step, where the possible maximum numbers of gradient iterations `MaxGradIter` and augmented Lagrangian iterations `MaxMultIter` are limited to 20. Afterwards, the estimated value for `PenaltyMin` is set as detailed below and, if `rungrampc` = 1, the initial states $\boldsymbol{x}$, controls $\boldsymbol{u}$ and costs J are reset.

Based on the initial values, a first estimate of the minimal penalty parameter is computed according to

$$\hat{c}_{\min}^{\mathrm{I}} = \frac{2\,|J|}{\|\boldsymbol{g}(\boldsymbol{x}(t),\boldsymbol{u}(t),\boldsymbol{p},t)\|_{L_2}^2 + \|\boldsymbol{h}(\boldsymbol{x}(t),\boldsymbol{u}(t),\boldsymbol{p},t)\|_{L_2}^2 + \|\boldsymbol{g}_T(\boldsymbol{x}(T),\boldsymbol{p},T)\|_2^2 + \|\boldsymbol{h}_T(\boldsymbol{x}(T),\boldsymbol{p},T)\|_2^2} \tag{4.28}$$

However, if the inequality constraints are initially inactive and are far away from their bounds (i.e. large negative values are returned), the estimate $\hat{c}_{\min}^{\mathrm{I}}$ may be too small.

To deal with these cases, a second estimate for the minimal penalty parameter

$$\hat{c}_{\min}^{\mathrm{II}} = \frac{2\,|J|}{T\left(\|\boldsymbol{\varepsilon}_{\boldsymbol{g}}\|_2^2 + \|\boldsymbol{\varepsilon}_{\boldsymbol{h}}\|_2^2\right) + \|\boldsymbol{\varepsilon}_{\boldsymbol{g}_T}\|_2^2 + \|\boldsymbol{\varepsilon}_{\boldsymbol{h}_T}\|_2^2} \tag{4.29}$$

is computed in the same spirit using the constraint tolerances (see `ConstraintsAbsTol`, $\boldsymbol{\varepsilon}_{\boldsymbol{g}}$, $\boldsymbol{\varepsilon}_{\boldsymbol{h}}$, $\boldsymbol{\varepsilon}_{\boldsymbol{g}_T}$ and $\boldsymbol{\varepsilon}_{\boldsymbol{h}_T}$) instead of the constraint values[2]. Since the norms of the tolerances are summed, more conservative values for $c_{\min}$ are estimated and therefore instabilities can be avoided. Note that it is recommended to scale all constraints so that they are in the same order of magnitude, see e.g. the PMSM example in Section 6.1.

Finally, the minimal penalty parameter

$$\hat{c}_{\min} = \min\left\{\max\left\{\hat{c}_{\min}^{\mathrm{I}},\, \kappa\,\hat{c}_{\min}^{\mathrm{II}}\right\},\, \frac{c_{\max}}{500}\right\} \tag{4.30}$$

is chosen as the maximum of (4.28) and (4.29) and additionally limited to 0.2 % of the maximum penalty parameter $c_{\max}$. This limitation ensures reasonable values even with very small constraint tolerances. The relation factor κ has been determined to 1×10^{-6} on the basis of various example systems. Please note that this estimation is intended to assist the user in making an initial guess. Problem-specific tuning of `PenaltyMin` can lead to further performance improvements and is therefore recommended. All MPC example problems in `<grampc_root>/examples` contain an initial call of `grampc_estim_penmin` to estimate $\hat{c}_{\min}$ and, as alternative, manually tuned values that can further enhance the performance of GRAMPC for fixed numbers of iterations.

4.5 Convergence criterion

While the usage of fixed iteration counts $i_{\max}$ and $j_{\max}$ for the outer and inner loops is typical in real-time MPC or MHE applications, GRAMPC also provides an optional convergence check that is

[2] The integration behind the L^2-norm can be replaced by a multiplication by the horizon length T, as the constraint tolerances $\boldsymbol{\varepsilon}_{\boldsymbol{g}}$ and $\boldsymbol{\varepsilon}_{\boldsymbol{h}}$ are no functions of time.

useful for solving optimal control or parameter optimization problems, or if the computation time in MPC is of minor importance.

The inner gradient loop in Table 1 evaluates the maximum relative gradient

$$\eta^{i|j+1} = \max\left\{ \frac{\|\boldsymbol{u}^{i|j+1} - \boldsymbol{u}^{i|j}\|_{L_2}}{\|\boldsymbol{u}^{i|j+1}\|_{L_2}}, \frac{\left\|\boldsymbol{p}^{i|j+1} - \boldsymbol{p}^{i|j}\right\|_2}{\left\|\boldsymbol{p}^{i|j+1}\right\|_2}, \frac{|T^{i|j+1} - T^{i|j}|}{T^{i|j+1}} \right\} \tag{4.31}$$

in each iteration $i|j$ and terminates if

$$\eta^{i|j} \le \varepsilon_{\text{rel,c}} \,. \tag{4.32}$$

Otherwise, the inner loop is continued until the maximum number of iterations $j_{\max}$ is reached. The last value $\eta^i = \eta^{i|j+1}$ is returned to the outer loop and used for the convergence check as well as for the update of multipliers and penalties.

The augmented Lagrangian loop is terminated in iteration i if the inner loop is converged, that is $\eta^i \le \varepsilon_{\text{rel,c}}$, and all constraints are sufficiently satisfied, i.e.

$$\begin{bmatrix} |\, \boldsymbol{g}^i(t) \,| \\ \mathbf{max}\{\mathbf{0}, \bar{\boldsymbol{h}}^i(t)\} \end{bmatrix} \le \begin{bmatrix} \boldsymbol{\varepsilon}_g \\ \boldsymbol{\varepsilon}_h \end{bmatrix} \ \forall t \in [0,T] \quad \wedge \quad \begin{bmatrix} |\, \boldsymbol{g}_T^i \,| \\ \mathbf{max}\{\mathbf{0}, \bar{\boldsymbol{h}}_T^i\} \end{bmatrix} \le \begin{bmatrix} \boldsymbol{\varepsilon}_{g_T} \\ \boldsymbol{\varepsilon}_{h_T} \end{bmatrix}, \tag{4.33}$$

whereby the notation $|\cdot|$ denotes the component-wise absolute value. Otherwise, the outer loop is continued until the maximum number of iterations $i_{\max}$ is reached. The thresholds $\boldsymbol{\varepsilon}_g, \boldsymbol{\varepsilon}_h, \boldsymbol{\varepsilon}_{g_T}, \boldsymbol{\varepsilon}_{h_T}$ are vector-valued in order to rate each constraint individually.

The following options can be used to adjust the convergence criterion:

- `ConvergenceCheck`: This option activates the convergence criterion. Otherwise, the inner and outer loops always perform the maximum number of iterations, see the options `MaxGradIter` and `MaxMultIter`.

- `ConvergenceGradientRelTol`: This option sets the threshold $\varepsilon_{\text{rel,c}}$ for the maximum relative gradient of the inner minimization problem that is used in the convergence criterion. Note that this threshold is different from the one that is used in the update of multipliers and penalties.

- `ConstraintsAbsTol`: Thresholds $(\boldsymbol{\varepsilon_g}, \boldsymbol{\varepsilon_h}, \boldsymbol{\varepsilon_{g_T}}, \boldsymbol{\varepsilon_{h_T}}) \in \mathbb{R}^{N_c}$ for the equality, inequality, terminal equality, and terminal inequality constraints.

4.6 Scaling

Scaling is recommended for improving the numerical conditioning when the states $\boldsymbol{x}$ and the optimization variables $(\boldsymbol{u}, \boldsymbol{p}, T)$ of the given optimization problem differ in several orders of magnitude. Although GRAMPC allows one to scale a specific problem automatically using the option `ScaleProblem` = 1, it should be noted that this typically increases the computational load due to the additional multiplications in the algorithm, cf. the tutorial on controlling a permanent magnet synchronous machine in Section 6.1. This issue can be avoided by directly formulating the scaled problem within the C file template `probfct_TEMPLATE.c` included in the folder `examples/TEMPLATE`, also see Section 3.2.

The scaling in GRAMPC is performed according to

$$\bar{\boldsymbol{x}}(t) = (\boldsymbol{x}(t) - \boldsymbol{x}_{\text{offset}}) \,./\, \boldsymbol{x}_{\text{scale}} \tag{4.34a}$$

$$\bar{\boldsymbol{u}}(t) = (\boldsymbol{u}(t) - \boldsymbol{u}_{\text{offset}}) \,./\, \boldsymbol{u}_{\text{scale}}\,, \quad \bar{\boldsymbol{p}} = (\boldsymbol{p} - \boldsymbol{p}_{\text{offset}}) \,./\, \boldsymbol{p}_{\text{scale}}\,, \quad \bar{T} = \frac{T - T_{\text{offset}}}{T_{\text{scale}}}\,, \tag{4.34b}$$

where $\boldsymbol{x}_{\text{offset}} \in \mathbb{R}^x$, $\boldsymbol{u}_{\text{offset}} \in \mathbb{R}^u$, $\boldsymbol{p}_{\text{offset}} \in \mathbb{R}^p$ and $\boldsymbol{T}_{\text{offset}} \in \mathbb{R}$ denote offset values and $\boldsymbol{x}_{\text{scale}} \in \mathbb{R}^x$, $\boldsymbol{u}_{\text{scale}} \in \mathbb{R}^u$, $\boldsymbol{p}_{\text{scale}} \in \mathbb{R}^p$ and $\boldsymbol{T}_{\text{scale}} \in \mathbb{R}$ are scaling values. The symbol ./ in (4.34) denotes element-wise division by the scaling vectors.

Furthermore, GRAMPC provides a scaling factor J_{scale} for the cost functional as well as scaling factors $\boldsymbol{c}_{\text{scale}} = [\boldsymbol{c}_{\text{scale},\boldsymbol{g}}, \boldsymbol{c}_{\text{scale},\boldsymbol{h}}, \boldsymbol{c}_{\text{scale},\boldsymbol{g}_T}, \boldsymbol{c}_{\text{scale},\boldsymbol{h}_T}] \in \mathbb{R}^{N_c}$ for the constraints. The scaling of the cost functional is relevant as the constraints are adjoined to the cost functional by means of Lagrangian multipliers and penalty parameters and the original cost functional should be of the same order of magnitude as these additional terms.

The following options can be used to adjust the scaling:

- `ScaleProblem`: Activates or deactivates scaling. Note that GRAMPC requires more computation time if scaling is active.
- `xScale`, `xOffset`: Scaling factors $\boldsymbol{x}_{\text{scale}}$ and offsets $\boldsymbol{x}_{\text{offset}}$ for each state variable.
- `uScale`, `uOffset`: Scaling factors $\boldsymbol{u}_{\text{scale}}$ and offsets $\boldsymbol{u}_{\text{offset}}$ for each control variable.
- `pScale`, `pOffset`: Scaling factors $\boldsymbol{p}_{\text{scale}}$ and offsets $\boldsymbol{p}_{\text{offset}}$ for each parameter.
- `TScale`, `TOffset`: Scaling factor $\boldsymbol{T}_{\text{scale}}$ and offset $\boldsymbol{T}_{\text{offset}}$ for the horizon length.
- `JScale`: Scaling factor J_{scale} for the cost functional.
- `cScale`: Scaling factors $\boldsymbol{c}_{\text{scale}}$ for each state constraint. The elements of the vector refer to the equality, inequality, terminal equality and terminal inequality constraints.

4.7 Control shift

The principle of optimality for an infinite horizon MPC problem motivates to shift the control trajectory $\boldsymbol{u}(t)$, $t \in [0, T]$ from the previous MPC step $k-1$ by the sampling time Δt before the first GRAMPC iteration in the current MPC step k, cf. Table 1. The last time segment of the shifted trajectory is hold on the last value of the trajectory. Shifting the control will lead to a faster convergence behavior of the gradient algorithm for most MPC problems.

If the control shift is activated for a problem with free end time T, GRAMPC assumes a shrinking horizon problem, because time optimization is unusual in classical model predictive control. The principle of optimality then motivates to subtract the sampling time `dt` from the horizon T after each MPC step, which corresponds to a control shift for the end time.

- `ShiftControl`: Activates or deactivates the shifting of the control trajectory and the adaptation of T in case of a free end time, i.e., if `OptimTime` is active.

4.8 Status flags

Several status flags are set in the solution structure `grampc.sol.status` during the execution of Algorithm 1. These flags can be printed as short messages by the function `grampc_printstatus` for the levels error, warn, info and debug.

The following status flags are printed on the level `STATUS_LEVEL_ERROR` and require immediate action:

- `STATUS_INTEGRATOR_INPUT_NOT_CONSISTENT`: This flag is set by the integrator `rodas` if the input values are not consistent. See [12] for further details.

- `STATUS_INTEGRATOR_MAXSTEPS`: This flag is set by the integrators `ruku45` or `rodas` if too many steps are required.
- `STATUS_INTEGRATOR_STEPS_TOO_SMALL`: This flag is set by the integrator `rodas` if the step size becomes too small.
- `STATUS_INTEGRATOR_MATRIX_IS_SINGULAR`: This flag is set by the integrator `rodas` if a singular Jacobian $\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}$ or $\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\right)^{\mathsf{T}}$ is detected. See [12] for further details.
- `STATUS_INTEGRATOR_H_MIN`: This flag is set by the integrator `ruku45` if a smaller step size than the minimal allowed value is required.

The following flags are printed in addition to the previous ones on the level `STATUS_LEVEL_WARN`:

- `STATUS_MULTIPLIER_MAX`: This flag is set if one of the multipliers $\bar{\boldsymbol{\mu}}$ reaches the upper limit $\mu_{\max}$ or the lower limit $-\mu_{\max}$. The situation may occur for example if the problem is infeasible or ill-conditioned or if the penalty parameters are too high.
- `STATUS_PENALTY_MAX`: This flag is set if one of the penalty parameters $\bar{\boldsymbol{c}}$ reaches the upper limit $c_{\max}$. The situation may occur for example if the problem is infeasible or ill-conditioned or if the penalty increase factor β_{in} is too high.
- `STATUS_INFEASIBLE`: This flag is set if the constraints are not satisfied and one run of Algorithm 1 does not reduce the norm of the constraints. The situation may occur in single runs, if few iterations $i_{\max}$ and $j_{\max}$ are used for a suboptimal solution. However, if the flag is set in multiple successive runs, it is a strong indicator for an infeasible optimization problem.

The following flags are printed in addition to the previous ones on the level `STATUS_LEVEL_INFO`:

- `STATUS_GRADIENT_CONVERGED`: This flag is set if the convergence check is activated and the relative tolerance $\varepsilon_{\text{rel,c}}$ is satisfied for the controls $\boldsymbol{u}$, the parameters $\boldsymbol{p}$, and the end time T.
- `STATUS_CONSTRAINTS_CONVERGED`: This flag is set if the convergence check is activated and the absolute tolerances $\boldsymbol{\varepsilon}_g, \boldsymbol{\varepsilon}_h, \boldsymbol{\varepsilon}_{g_T}, \boldsymbol{\varepsilon}_{h_T}$ are satisfied for all constraints.
- `STATUS_LINESEARCH_INIT`: This flag is set if the gradient algorithm uses the initial step size α_{init} as fallback for the explicit line search strategy in one iteration.

The following flags are printed in addition to the previous ones on the level `STATUS_LEVEL_DEBUG`:

- `STATUS_LINESEARCH_MAX`: This flag is set if the gradient algorithm uses the maximum step size $\alpha_{\max}$ in one iteration.
- `STATUS_LINESEARCH_MIN`: This flag is set if the gradient algorithm uses the minimum step size $\alpha_{\min}$ in one iteration.
- `STATUS_MULTIPLIER_UPDATE`: This flag is set if the relative tolerance $\varepsilon_{\text{rel,u}}$ is satisfied for the controls $\boldsymbol{u}$, the parameters $\boldsymbol{p}$ and the end time T and therefore the update of the multipliers $\bar{\boldsymbol{\mu}}$ and the penalty parameters $\bar{\boldsymbol{c}}$ is performed, cf. Section 4.4.

5 Usage of GRAMPC

This chapter describes how to use GRAMPC in C and via the interface to Matlab/Simulink. Note that the implementation of the problem formulation is described in Section 3.2 and therefore not repeated here.

5.1 Using GRAMPC in C

A high level of portability is provided by GRAMPC in view of different operating systems and hardware devices. The main components are written in plain C without the use of external libraries, also see the discussion in Section 2.3. This section describes the usage of GRAMPC in C including initialization, compilation and running the MPC framework.

5.1.1 Main components of GRAMPC

As illustrated in Figure 5.1, GRAMPC contains initializing as well as running files, different integrators for the system and adjoint dynamics (cf. Chapter 3 and 4) and functions to alter the options and parameters (also see Chapter 3 and 4).

In more detail, GRAMPC comprises the following main files (see also Appendix A.4 for the function interface):

- `grampc_init.c`: Functions for initializing GRAMPC.
- `grampc_alloc.c`: Functions for memory allocation and deallocation.
- `grampc_run.c`: Functions for running GRAMPC including the implemented augmented Lagrangian algorithm and the underlying gradient algorithm. Further functions of this file are concerned with the line search strategies and the update steps of the primal and dual variables as described in Chapter 4.
- `grampc_setparam.c`: Provides several functions for setting problem-related parameters, see the description in Chapter 3 and the list of parameters in Table A.1.
- `grampc_setopt.c`: Provides several functions for setting algorithmic options, see the description in Chapter 4 and the list of options in Table A.2.
- `grampc_mess.c`: Function for printing information regarding the initialization as well as execution of GRAMPC (e. g. errors, convergence behavior of the augmented Lagrangian algorithm or status of integrators).
- `grampc_util.c`: Auxiliary functions for the GRAMPC toolbox such as memory management and trajectory interpolation. It also contains the function `grampc_estim_penmin` to compute an estimate of the minimal penalty parameter, cf. Section 4.4.3.
- `simpson.c`: Function for integrating the integral cost by means of the Simpson rule.
- `trapezodial.c`: Function for integrating the integral cost by means of the trapezodial rule.

- `euler1.c`: Euler forward integration scheme with fixed step size.
- `eulermod2.c`: Modified Euler integration scheme with fixed step size.
- `heun2.c`: Heun integration scheme with fixed step size.
- `rodas.c`: Semi-implicit Rosenbrock integration scheme with variable step size. The code follows from an f2c conversion of the original Fortran files of RODAS [12] and are directly included in the GRAMPC source files.
- `ruku45.c`: Runge-Kutta integration scheme of order 4 with variable step size.

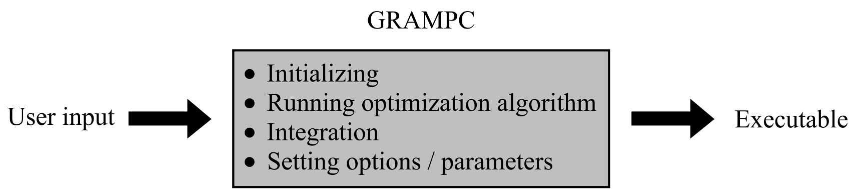
GRAMPC
User input
• Initializing
• Running optimization algorithm
• Integration
• Setting options / parameters
Executable

Figure 5.1: Main components of GRAMPC.

5.1.2 Initialization of GRAMPC

The global initialization of GRAMPC is done via the routine

```
void grampc_init(typeGRAMPC **grampc, typeUSERPARAM *userparam)
```

where the overall structure variable `grampc` contains the following substructures (see Appendix A.3 for the definition of the data type):

- `sol` (data type `typeGRAMPCsol`): Contains the optimization variables $(\boldsymbol{u}^{i+1}, \boldsymbol{p}^{i+1}, T^{i+1})$ and the interpolated state $\boldsymbol{x}^{i+1}$ in the next MPC step as well as the corresponding cost values $\bar{J}(\boldsymbol{u}^{i+1}, \boldsymbol{p}^{i+1}, T^{i+1}, \boldsymbol{\mu}^{i+1}, \boldsymbol{c}^{i+1}; \boldsymbol{x}_0)$ and $J(\boldsymbol{u}^{i+1}, \boldsymbol{p}^{i+1}, T^{i+1}; \boldsymbol{x}_0)$, respectively. The control $\boldsymbol{u}^{i+1}$ and the state $\boldsymbol{x}^{i+1}$ refer to the corresponding trajectories evaluated at the sampling time Δt. In addition, the substructure `sol` contains the evaluated functions of the defined state constraints, some status information, and an array in which the number of gradient iterations are stored in each augmented Lagrangian step.
- `param` (data type `typeGRAMPCparam`): Contains the parameter structure of GRAMPC. A detailed description of all parameters is given in Chapter 3.
- `opt` (data type `typeGRAMPCopt`): Contains the option structure of GRAMPC. A detailed description of all options is given in Chapter 4.
- `rws` (data type `typeGRAMPCrws`): Contains the real-time workspace of GRAMPC including calculations of the augmented Lagrangian algorithm and the gradient algorithm along the prediction horizon.
- `userparam` (data type `typeUSERPARAM`): Can be used to define parameters, e.g. to parametrize the cost functional (3.1a), the system dynamics (3.1b) or the state constraints (3.1c)–(3.1d).

The definition of these data types is given in Appendix A.3. The deallocation of `grampc` is done by means of the function

```
void grampc_free(typeGRAMPC **grampc)
```

5.1.3 Setting options and parameters

The single options in Table A.2 are set via the functions

```
void grampc_setopt_real(const typeGRAMPC *grampc,
                        const typeChar *optName, ctypeRNum optValue)
void grampc_setopt_real_vector(const typeGRAMPC *grampc,
                               const typeChar *optName, ctypeRNum *optValue)

void grampc_setopt_int(const typeGRAMPC *grampc,
                       const typeChar *optName, ctypeInt optValue)
void grampc_setopt_int_vector(const typeGRAMPC *grampc,
                              const typeChar *optName, ctypeInt *optValue)

void grampc_setopt_string(const typeGRAMPC *grampc,
                          const typeChar *optName, const typeChar *optValue)
```

for option values with floating point, integer and string type, respectively. An overview of the current options can be displayed by using

```
void grampc_printopt(typeGRAMPC *grampc)
```

Example (Setting options in C)
The number of gradient iterations, the integration scheme, and the relative tolerance of the integrator can be set in the following way

```
...
/********* declaration *********/
typeGRAMPC *grampc;
...

/********* option definition *********/
/* Basic algorithmic options */
ctypeInt MaxGradIter = 2;

/* System integration */
const char* Integrator = "ruku45";
ctypeRNum IntegratorRelTol = 1e-3;
...

/********* grampc init *********/
grampc_init(&grampc, userparam);
...

/********* setting options *********/
grampc_setopt_int(grampc, "MaxGradIter", MaxGradIter);

grampc_setopt_string(grampc, "Integrator", Integrator);
grampc_setopt_real(grampc, "IntegratorRelTol", IntegratorRelTol);
...
```

Similar to setting the GRAMPC options, the parameters in Table A.1 are set according to their data type with the following functions

```
void grampc_setparam_real(const typeGRAMPC *grampc,
                          const typeChar *paramName, ctypeRNum paramValue);

void grampc_setparam_real_vector(const typeGRAMPC *grampc,
                                 const typeChar *paramName, ctypeRNum *paramValue);
```

An overview of the parameters can be displayed by using

```
void grampc_printparam(const typeGRAMPC *grampc);
```

Example (Setting parameters in C)
The sampling time, the prediction horizon, and the initial conditions for a system with two states and one control input can be set in the following way:

```
...
/********* parameter definition *********/
ctypeRNum dt = (typeRNum)0.001;
ctypeRNum Thor = 6.0;

ctypeRNum x0[NX] = {-1,-1};
ctypeRNum u0[NU] = {0};


/********* setting parameters *********/
grampc_setparam_real(grampc, "dt", dt);
grampc_setparam_real(grampc, "Thor", Thor);
grampc_setparam_real_vector(grampc, "x0", x0);
grampc_setparam_real_vector(grampc, "u0", u0);
...
```

5.1.4 Compiling and calling GRAMPC

GRAMPC can be integrated into an executable program after the problem formulation (cf. Chapter 3) as well as options and parameters are provided. A makefile for compilation purposes is provided in the folder `<grampc_root>/examples/TEMPLATES`. The main calling routine for GRAMPC is

```
void grampc_run(const typeGRAMPC *grampc)
```

The following code demonstrates how to integrate GRAMPC within an MPC loop by completing the ball-on-plate example, cf. Section 3.2.3. The code is part of the file `main_BALL_ON_PLATE.c` contained in the GRAMPC folder `<grampc_root>/examples/BallOnPlate`.

Example (C code for running GRAMPC **within an MPC loop)**

```
#include "grampc.h"   /* contains all necessary header files */

#define NX  2
#define NU  1
#define NC  4
#define NP  0

int main()
{
  /********* Declaration and parameter definition *********/
  typeGRAMPC *grampc;
  typeInt iMPC, MaxSimIter, i;
  ctypeRNum Tsim = 8.0;
  clock_t tic, toc;
  typeRNum *CPUtimeVec;
  typeRNum CPUtime = 0;

  ctypeRNum x0[NX] = {0.1,0.01};   ctypeRNum xdes[NX] = {-0.2,0.0};
  ctypeRNum u0[NU] = {0.0};        ctypeRNum udes[NU] = {0.0};
  ctypeRNum umax[NU] = {0.0524};   ctypeRNum umin[NU] = {-0.0524};

  ctypeRNum Thor = 0.3;
  ctypeRNum dt = (typeRNum)0.01;
  typeRNum t = (typeRNum)0.0;

  /********* userparam *********/
  typeRNum param[9] = { 100, 10, 1, 100, 10, -0.2, 0.01, -0.1, 0.1 };
  typeUSERPARAM *userparam = param;

  /********* grampc init *********/
  grampc_init(&grampc, userparam);

  /********* Option definition *********/
  /* Basic algorithmic options */
  ctypeRNum Nhor = 10;
  ctypeInt MaxGradIter = 2;
  ctypeInt MaxMultIter = 3;

  ctypeRNum AugLagUpdateGradientRelTol = (typeRNum)1e0;
  ctypeRNum ConstraintsAbsTol[4] = {1e-3, 1e-3, 1e-3, 1e-3};

  /********* set parameters and option *********/
  grampc_setparam_real_vector(grampc, "x0", x0);
  grampc_setparam_real_vector(grampc, "xdes", xdes);

  grampc_setparam_real_vector(grampc, "u0", u0);
  grampc_setparam_real_vector(grampc, "udes", udes);
  grampc_setparam_real_vector(grampc, "umax", umax);
  grampc_setparam_real_vector(grampc, "umin", umin);

  grampc_setparam_real(grampc, "Thor", Thor);
  grampc_setparam_real(grampc, "dt", dt);
  grampc_setparam_real(grampc, "t0", t);

  /********* Option definition *********/
  grampc_setopt_int(grampc, "Nhor", Nhor);
  grampc_setopt_int(grampc, "MaxGradIter", MaxGradIter);
  grampc_setopt_int(grampc, "MaxMultIter", MaxMultIter);

  grampc_setopt_real(grampc,"AugLagUpdateGradientRelTol",AugLagUpdateGradientRelTol);
  grampc_setopt_real_vector(grampc,"ConstraintsAbsTol",ConstraintsAbsTol);

  /********* estimate and set PenaltyMin *********/
  grampc_estim_penmin(grampc, 1);

  /* MPC loop */
  for (iMPC = 0; iMPC <= MaxSimIter; iMPC++) {
      tic = clock();
    grampc_run(grampc);
    toc = clock();
    CPUtimeVec[iMPC] = (typeRNum)((toc - tic) * 1000 / CLOCKS_PER_SEC);

    /* check solver status */
    if (grampc->sol->status > 0) {
      if (grampc_printstatus(grampc->sol->status, STATUS_LEVEL_ERROR)) {
        myPrint("at iteration %i:\n -----\n", iMPC);
      }
    }

    /* update state and time */
    t = t + dt;
    grampc_setparam_real_vector(grampc, "x0", grampc->sol->xnext);
  }

  /* Memory deallocation */
  grampc_free(&grampc);
  free(CPUtimeVec);
  return 0;
}
```

The structure `grampc` is initialized at the beginning and some options are set. Afterwards, the mandatory as well as some optional parameters are defined in the code. GRAMPC is then repetitively executed until a defined simulation time is reached. Note that the estimate of the minimal penalty value $c_{\min}$ is determined via the function `grampc_estim_penmin` before the augmented Lagrangian algorithm is executed, also see the discussion in Section 4.4.3.

For a more convenience MPC design, there are several status flags that can be printed after each MPC step. As shown in the above example, the variable `grampc->sol->status` is used to check whether new status informations are available. Subsequently, the function `grampc_printstatus` is used to visualize the corresponding informations on the console. In the current GRAMPC version, there are status informations concerning the integration scheme, the update of Lagrange multipliers, convergence properties of the augmented Lagrangian and gradient algorithm, and the line search method, see Section 4.8 for more details. In addition, the GRAMPC examples provide functions to print key variables such as the system state or the controls into text files. These files are stored in the subfolder `res` of the example directory.

The ball-on-plate example described above can be compiled by running the following commands in a (Cygwin) terminal:

```
$ cd <grampc_root>/examples/BallOnPlate
$ make
```

The make command compiles the file `main_BALL_ON_PLATE.c` and links it against the GRAMPC toolbox, i. e., against the library `libgrampc.a` within `<grampc_root>/libs`.[1] As a result, the executable `startMPC` is generated, which can now be used to solve and/or design the MPC problem.

5.1.5 Using GRAMPC without dynamic memory allocation

GRAMPC restricts the usage of dynamic memory allocation, i.e. `calloc` and `realloc`, to initialization and option setting, such that no allocations are performed while the MPC is running. However, some microcontrollers and embedded devices do not support dynamic memory allocation at all. Using GRAMPC on these devices requires to replace all dynamically-sized arrays by fixed-size arrays and to remove all functions that involve dynamic memory allocation.

To this end, a header file `fixedsize_settings.h` must be placed in the search path of the compiler. This header file defines several constant parameters such as the number of states `NX`, the number of controls `NU`, as well as several constant options, e.g. the number of discretization steps `NHOR`, the number of gradient iterations `MAXGRADITER`, and the number of multiplier iterations `MAXMULTITER`. Note that these options cannot be changed during run-time, but are fixed at compile-time. A template file is provided in the folder `<grampc_root>/examples/templates`.

In addition, the GRAMPC structures must be created on the stack instead of the heap. To this end, the macro `TYPE_GRAMPC_POINTER` is provided that allows to use the same code both with and without dynamic memory allocation:

```
int main()
{
  TYPE_GRAMPC_POINTER(grampc);
  ...
  grampc_init(&grampc, userparam);
  ...
```

[1]Note that compiling an application example in the folder `<grampc_root>/examples/` requires the previous compilation of the GRAMPC toolbox as described in Chapter 2.

```
  grampc_free(&grampc);
}
```

The makefile can be called with the parameter `FIXEDSIZE=1`, which automatically defines the required preprocessor macro `FIXEDSIZE`. Thus compiling and running the ball-on-plate example without dynamic memory allocation is done by executing the following commands in the terminal:

```
$ cd <grampc_root>/examples/BallOnPlate
$ make FIXEDSIZE=1
$ ./startMPC_fixedsize
```

Note that in this case a separate GRAMPC library named `libgrampc_fixedsize.a` is created in the problem folder that depends on the constants defined in `fixedsize_settings.h`. Changing these settings requires to recompile both the library and the application.

5.2 Using GRAMPC in Matlab/Simulink

As already mentioned, the main components of GRAMPC are implemented in plain C to ensure a high level of portability. However, GRAMPC also provides a user-friendly interface with Matlab/Simulink to allow for a convenient MPC design.

5.2.1 Interface to Matlab

Each main component of GRAMPC (cf. Section 5.1.1) has a related Mex routine that is included in the directory `<grampc_root>/matlab/src`, also see Figure 5.2. This allows one to run GRAMPC in Matlab/Simulink as well as altering options and parameters without recompilation.

A makefile to compile GRAMPC for use in Matlab/Simulink is provided in `<grampc_root>/matlab`. The makefile compiles the source files to generate object files within `<grampc_root>/matlab/bin`. In order to obtain an actual Mex compilation for a given problem, the object files must be linked against the object file obtained by compiling the problem function, since at least some of these functions depend on the actual problem formulation, e.g. the state dimension $N_{\boldsymbol{x}}$.

The m-file `startMPC.m` in each of the examples under `<grampc_root>/examples` contains a flag `compile`. Setting this flag to 1 leads to a compilation of the problem file and to the generation of the Mex files. Setting `compile` to 2 leads to an additional recompilation of the toolbox by calling the makefile in the directory `<grampc_root>/matlab/src`. The Mex files are stored in the local subfolder `+CmexFiles`. The folder name begins with a plus sign allowing the user to call the functions stored in this folder using the command `CmexFiles.<function name>`. The S-function files are stored in the local application directory, since Simulink requires the S-functions to be in the Matlab path.

The structure of the main components of GRAMPC in C and Matlab for setting options and parameters are slightly different, as it is not allowed (or at least not very elegant) to manipulate the input argument of Mex routines. Consequently, each Mex routine returns the manipulated structure variable `grampc` as an output argument. If, for example, the initial condition $\boldsymbol{x}_0$ should be set to a specific value in C, the function `grampc_setparam_real_vector` must be used, as already discussed in Section 5.1.2. In Matlab, however, this is done using the Mex routine `grampc_setparam_Cmex` with the structure variable `grampc` as an input argument. The manipulated structure variable `grampc` is returned as an output argument including the initial condition x_0. For instance, the parameter setting in C

```
ctypeRNum x0[NX] = {-1,-1};
grampc_setparam_real_vector(grampc, "x0", x0);
```

would read as follows in Matlab:

```
grampc = grampc_setparam_Cmex(grampc,'x0',[-1;-1]);
```

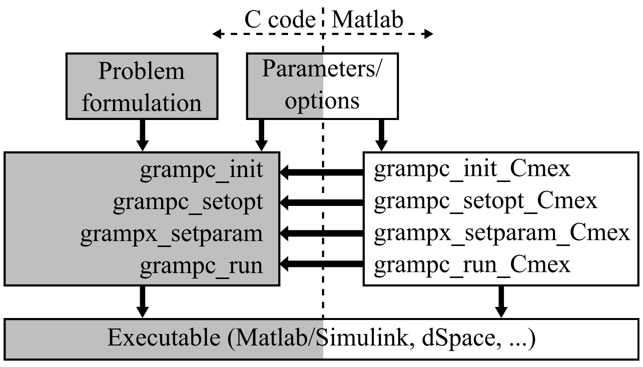

Figure 5.2: Interface of GRAMPC to Matlab/Simulink (gray – C code, white – Matlab code)

Note that the Mex routine `grampc_setparam_Cmex` does not distinguish between vectors and scalars, but handles the different dimensions of parameters internally. The same applies to the Mex routine `grampc_setopt_Cmex` for changing algorithmic options in GRAMPC. The data type of the parameter or option to be set can either be double or the corresponding data type in the parameter structure `param` or option structure `opt`, see also Table A.1 or Table A.2.

In order to simplify changing parameters and options in Matlab, GRAMPC also provides the routine `grampc_update_struct_grampc(grampc,user)` included in `<grampc_root>/matlab/mfiles`. The purpose of this function is to allow the user to define the options and parameters to be set as structure variable instead of requiring to call the functions `grampc_setopt_Cmex` and `grampc_setparam_Cmex` manually for each chosen parameter/option. In detail, the structure variable `user` must contain the substructures `param` and `opt` that define the parameters and options to be set. The corresponding function call under Matlab is as follows.

```
...
%% Parameter definition
% Initial values of the states
user.param.x0 = [-1;-1];
...

%% Option definition
% Basic algorithmic options
user.opt.Nhor = 10;
...

%% User parameter definition
% e.g. system parameters or weights for the cost function
userparam = [100, 10, 1, 100, 10, -0.2, 0.01, -0.1, 0.1];

%% Grampc initialization
grampc = CmexFiles.grampc_init_Cmex(userparam);

%% Update grampc struct while ensuring correct data types
grampc = grampc_update_struct_grampc(grampc,user);
...
```

Similar to Section 5.1.2 and Section 5.1.4, the following lines show the main steps to run the ball-on-plate example in Matlab. An executable version of this example within Matlab can be found in the folder `<grampc_root>/examples/BallOnPlate`. In analogy to the C implementation, the simulation loop and the evaluation are implemented in the main file `startMPC.m`. The parameters and options are defined in the separate file `initData.m` that is called within `startMPC.m` for the sake of readability and to use the settings directly in the Simulink model, see Section 5.2.2. Pleas note a template file `initData_TEMPLATE.m` can be found in the folder `<grampc_root>/examples/TEMPLATES`.

Example (Matlab code for setting options and parameters, see `initData.m`)

```
%% Parameter definition
user.param.x0    = [ 0.1, 0.01];
user.param.xdes  = [-0.2, 0.0];

% Initial values, setpoints and limits of the inputs
user.param.u0    = 0;
user.param.udes  = 0;
user.param.umax  =  0.0524;
user.param.umin  = -0.0524;

% Time variables
user.param.Thor  = 0.3;         % Prediction horizon

user.param.dt    = 0.01;        % Sampling time
user.param.t0    = 0.0;         % time at the current sampling step

%% Option definition
user.opt.Nhor        = 10;      % Number of steps for the system integration
user.opt.MaxMultIter = 3;       % Maximum number of augmented Lagrangian iterations

% Constraints thresholds
user.opt.ConstraintsAbsTol = 1e-3*[1 1 1 1];

%% User parameter definition, e.g. system parameters or weights for the cost function
userparam = [100, 10, 180, 100, 10, -0.2, 0.2, -0.1, 0.1];

%% Grampc initialization
grampc = CmexFiles.grampc_init_Cmex(userparam);

%% Update grampc struct while ensuring correct data types
grampc = grampc_update_struct_grampc(grampc,user);

%% Estimate and set PenaltyMin (optional)
grampc = CmexFiles.grampc_estim_penmin_Cmex(grampc,1);ot_stat(vec,grampc,phpS);
...
```

Example (Matlab code for running GRAMPC within an MPC loop, see `startMPC.m`)

```
...
%% Initialization
[grampc,Tsim] = initData;
CmexFiles.grampc_printopt_Cmex(grampc);
CmexFiles.grampc_printparam_Cmex(grampc);

% init solution structure
vec = grampc_init_struct_sol(grampc, Tsim);

% init plots and store figure handles
phpP = grampc_init_plot_pred(grampc,figNr);      figNr = figNr+1;
phpT = grampc_init_plot_sim(vec,figNr);          figNr = figNr+1;
phpS = grampc_init_plot_stat(vec,grampc,figNr); figNr = figNr+1;

%% MPC loop
i = 1;
while 1
  % set current time and current state
  grampc = CmexFiles.grampc_setparam_Cmex(grampc,'t0',vec.t(i));
  grampc = CmexFiles.grampc_setparam_Cmex(grampc,'x0',vec.x(:,i));

  % run MPC and save results
  [grampc,vec.CPUtime(i)] = CmexFiles.grampc_run_Cmex(grampc);
  vec = grampc_update_struct_sol(grampc, vec, i);

  % print solver status
  printed = CmexFiles.grampc_printstatus_Cmex(grampc.sol.status,'Error');

  % check for end of simulation
  if i+1 > length(vec.t)
    break;
  end

  % simulate system
  [~,xtemp] = ode45(@CmexFiles.grampc_ffct_Cmex,vec.t(i)+[0 double(grampc.param.dt)],
                    vec.x(:,i),odeopt,vec.u(:,i),vec.p(:,i),grampc.userparam);
  vec.x(:,i+1) = xtemp(end,:);

  % evaluate time-dependent constraints
  vec.constr(:,i) = CmexFiles.grampc_ghfct_Cmex(vec.t(i), vec.x(:,i), vec.u(:,i),
                                                 vec.p(:,i), grampc.userparam);

  % update iteration counter
  i = i + 1;

  % plot data
  grampc_update_plot_pred(grampc,phpP);
  grampc_update_plot_sim(vec,phpT);
  grampc_update_plot_stat(vec,grampc,phpS);
end
```

Similar to the C example in Section 5.1.4, the structure variable `grampc` is initialized before the options as well as optional parameters are set. In addition, the plot functions (see Section 5.2.3) are initialized before GRAMPC is started within an MPC loop, where the current state of the system (new initial condition) is provided to GRAMPC. After computing the new controls, the status of GRAMPC is printed, see Section 4.8 for more details. Subsequently, a reference integration of the system is performed and the constraints are evaluated before the plots are updated.

5.2.2 Interface to Matlab/Simulink

GRAMPC also allows a Matlab/Simulink integration via the S-function `grampc_run_Sfct.c` (also included in the directory `<grampc_root>/matlab/src`). A corresponding Simulink template can be found in the folder `<grampc_root>/examples/TEMPLATES` for a number of Matlab versions. The directory also contains the m-file `initData_TEMPLATE.m` which can be used for initializing GRAMPC's options and parameters, as mentioned in the previous subsection. Each of the Matlab examples in the directory `<grampc_root>/examples` also includes a corresponding Simulink model for simulation. The build procedure of the MEX routines additionally compiles the S-function for the Simulink block.

The Matlab/Simulink model of GRAMPC is shown in Figure 5.3. The block `MPC-Subsystem` contains algorithmic components of GRAMPC (implemented within the S-function `grampc_run_Sfct.c`). The block `Click to init grampc` must be executed by a double-click in order to initialize the structure variable `grampc` that is required by the Matlab/Simulink model. This generates also the Simulink-specific structure variable `grampc_sdata`. For the sake of convenience, the blocks `Click to compile the toolbox` and `Click to compile probfct` are included in the model to be able to compile the GRAMPC toolbox and the specific problem directly from Matlab/Simulink. The block `sysfct sfunction` is added in order to numerically integrate the system dynamics after each MPC

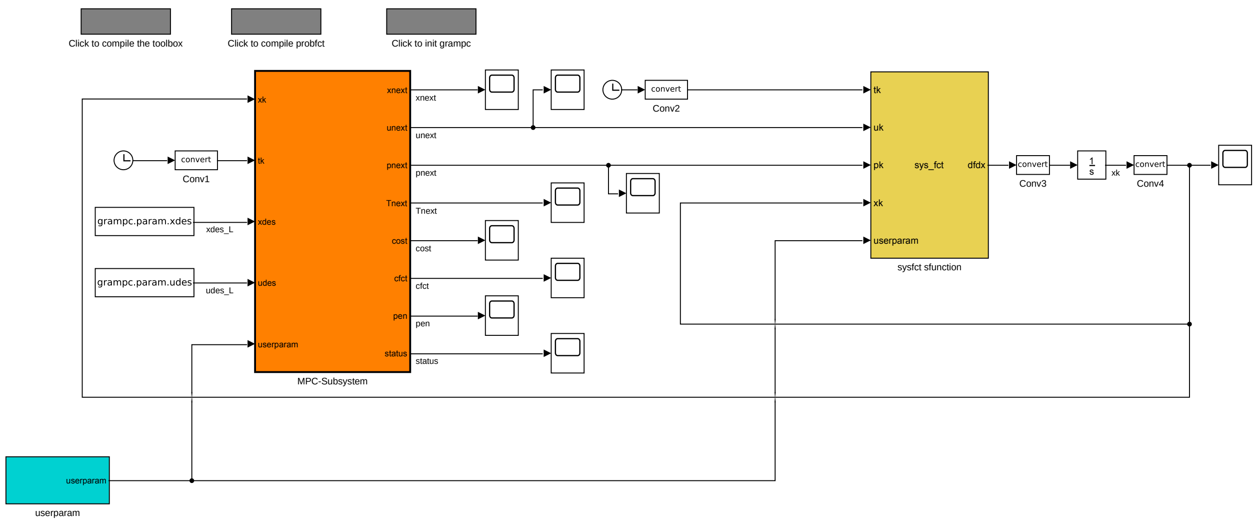

Figure 5.3: Matlab/Simulink model of GRAMPC.

step k for the sampling time Δt and to return the new state value $\boldsymbol{x}_{k+1}$ corresponding to the next sampling instant t_{k+1} that is fed back to the MPC block.

Furthermore, the S-function `grampc_run_Sfct.c` satisfies the additional restrictions of the Matlab code generator. Therefore, the block can be used in models implemented for running on various hardware platforms, such as dSpace real-time systems. Please note that especially in case of dSpace applications, the include folders `<grampc_root>/include` and `<grampc_root>/matlab/include` as well as all C source files in `<grampc_root>/src`, the source file of the S-function `<grampc_root>/matlab/src/grampc_run_Sfct.c` and the problem function must be listed as additional build information in the `Model Configuration Parameters` of the Simulink model under `Code Generation / Custom Code`. It is recommended to use absolute paths at least for the S-function file.

5.2.3 Plot functions

GRAMPC offers various plot functions in the folder `<grampc_root>/matlab/mfiles`. Each plot must be initialized at first using the m-files `grampc_init_plot_XYZ.m`. During the simulation the plots can be updated by the m-files `grampc_update_plot_XYZ.m`. Beside the trajectories of the simulated system dynamics (file ending `XYZ=sim`) and trajectories along the prediction horizon (file ending `XYZ=pred`), also some statistics (file ending `XYZ=stat`) can be plotted. When solving OCPs instead of MPC problems, the plot along the prediction horizon shows the actual results. The plotted quantities depend on the parameter and option settings of GRAMPC, i.e. whether constraints are considered or not. The available plots are listed in more detail in the following lines. (Also see the example problems under `<grampc_root>/examples` for code samples on how to use the plot routines.)

System dynamics plot (plot_sim)

- **States**: This plot illustrates the trajectories of the state $\boldsymbol{x}$ along the simulation time.
- **Adjoint states**: This plot illustrates the trajectories of the adjoint state $\boldsymbol{\lambda}$ along the simulation time.
- **Controls**: This plot illustrates the trajectories of the controls $\boldsymbol{u}$ along the simulation time.

- **Constraints**: This plot appears only if equality and/or inequality constraints are defined ($N_{\boldsymbol{g}}$ and/or $N_{\boldsymbol{h}}$ is larger than zero as specified in `ocp_dim`). The plot shows the trajectories of the constraints $\boldsymbol{g}$ and $\boldsymbol{h}$ along the simulation time.
- **Lagrange multipliers**: This plot appears only if equality and/or inequality constraints are defined. The plot shows the trajectories of the multipliers $\boldsymbol{\mu}_{\boldsymbol{g}}$ and $\boldsymbol{\mu}_{\boldsymbol{h}}$ along the simulation time. If any Lagrange multiplier reaches the limit $\mu_{\max}$ (specified by `MultiplierMax`), it indicates that the penalty parameters are too high or that the problem is not well-conditioned or that the costs are badly scaled.
- **Penalty parameters**: This plot appears only if equality and/or inequality constraints are defined. The plot shows the trajectories of the penalties $\boldsymbol{c}_{\boldsymbol{g}}$ and $\boldsymbol{c}_{\boldsymbol{h}}$ along the simulation time. If any penalty reaches the maximum value $c_{\max}$, set by `PenaltyMax`, it indicates that either the limit is not high enough or the update is too aggressive.

Prediction plot (plot_pred)

- **Predicted states**: This plot illustrates the trajectories of the state $\boldsymbol{x}$ along the prediction horizon.
- **Predicted adjoint states**: This plot illustrates the trajectories of the adjoint state $\boldsymbol{\lambda}$ along the prediction horizon.
- **Predicted controls**: This plot illustrates the trajectories of the controls $\boldsymbol{u}$ along the prediction horizon.
- **Predicted constraints**: This plot appears only if (terminal) equality and/or inequality constraints are defined. The plot shows the predicted violation of the equality constraints $\boldsymbol{g}$ and inequality constraints $\mathbf{max}(\boldsymbol{h}, \mathbf{0})$ along the prediction horizon and the predicted violation of the terminal equality constraints $\boldsymbol{g}_T$ and inequality constraints $\mathbf{max}(\boldsymbol{h}_T, \mathbf{0})$ at the end of the prediction horizon. Please note that except for OCPs, these are not the actual but predicted internal constraint violations of the current GRAMPC iteration.
- **Lagrange multipliers**: This plot appears only if (terminal) equality and/or inequality constraints are defined. The plot shows the trajectories of the multipliers $\boldsymbol{\mu}_{\boldsymbol{g}}$ and $\boldsymbol{\mu}_{\boldsymbol{h}}$ along the prediction horizon and the multipliers $\boldsymbol{\mu}_{\boldsymbol{g}_T}$ and $\boldsymbol{\mu}_{\boldsymbol{h}_T}$. If any Lagrange multiplier reaches the limit $\mu_{\max}$, set by `MultiplierMax`, it indicates that the penalty parameters are too high or that the problem is not well-conditioned or that the costs are badly scaled.
- **Penalty parameters**: This plot appears only if (terminal) equality and/or inequality constraints are defined. The plot shows the trajectories of the penalties $\boldsymbol{c}_{\boldsymbol{g}}$ and $\boldsymbol{c}_{\boldsymbol{h}}$ along the prediction horizon and the penalties $\boldsymbol{c}_{\boldsymbol{g}_T}$ and $\boldsymbol{c}_{\boldsymbol{h}_T}$. If any penalty reaches the maximum value $c_{\max}$ set by `PenaltyMax`, it indicates that either the limit is not high enough or the update is too aggressive, see also Section 4.4.

Statistics plot (plot_stat)

- **Costs**: This plot illustrates the costs J along the simulation time or along the augmented Lagrangian iterations. If constraints are defined, the augmented costs $\bar{J}$ are shown as well.
- **Computation time**: This plot illustrates the computation time of one MPC or optimization step of GRAMPC along the simulation time or along the augmented Lagrangian iterations. The time measurement is done in the function `grampc_run_Cmex.c` using operation system specific

timer functions. Consequently, the time excludes the overhead resulting from the Mex interface as well as the time consumed by the plot functions.

- **Line search step size**: This plot illustrates the step size α of the last gradient iteration along the simulation time or along the augmented Lagrangian iterations. If the adaptive line search is used (see Section 4.3.1), the plot also illustrates the three corresponding sample points α_1, α_2, and α_3. A step size equal to the maximum or minimum value $\alpha_{\max}$ or $\alpha_{\min}$ indicates that these values may have to be adapted or the problem may have to be scaled. Additionally, if the explicit line search is chosen and the fallback strategy is not activated (see Sections 4.3.2 and 4.3.3), a frequent use of the initial value α_{init} indicates an ill-conditioned problem.

- **Gradient iterations**: This plot appears only if the option `ConvergenceCheck` is set to `on`. It illustrates the number of executed gradient iterations along the simulation time or along the augmented Lagrangian iterations. In particular, the plot depicts whether the maximum number of gradient iterations $j_{\max}$ is reached or the convergence check caused a premature termination of the minimization.

- **Prediction horizon**: This plot appears only if the option `OptimTime` is set to `on`. It illustrates the prediction horizon T along the simulation time or along the augmented Lagrangian iterations. In shrinking horizon applications the value should decrease linearly after a short settling phase.

- **Norm of constraints over horizon**: This plot appears only if constraints are defined. It illustrates the norm $\frac{1}{T}\sqrt{\|\boldsymbol{g}\|_{L_2}^2 + \|\mathbf{max}(\boldsymbol{h},\mathbf{0})\|_{L_2}^2 + \|\boldsymbol{g}_T\|_2^2 + \|\mathbf{max}(\boldsymbol{h}_T,\mathbf{0})\|_2^2}$ over all predicted constraints plotted over the simulation time or the augmented Lagrangian iterations. Especially when solving OCPs, the value should decrease continuously.

- **Norm of penalty parameters over horizon**: This plot appears only if the number of equality $N_{\boldsymbol{g}}$, inequality $N_{\boldsymbol{h}}$, terminal equality $N_{\boldsymbol{g}_T}$ or terminal inequality $N_{\boldsymbol{h}_T}$ constraints is not zero. It illustrates the norm $\frac{1}{T}\sqrt{\|\bar{\boldsymbol{c}}\|_{L_2}}$ over all predicted penalty parameters along the simulation time or along the augmented Lagrangian iterations.

- **Terminal constraints**: This plot appears only if terminal constraints are defined. It illustrates the violation of the terminal equality constraints $\boldsymbol{g}_T$ and inequality constraints $\mathbf{max}(\boldsymbol{h}_T,\mathbf{0})$ along the simulation time or along the augmented Lagrangian iterations in case of OCPs.

- **Terminal Lagrangian multipliers**: This plot appears only if terminal constraints are defined. It illustrates the multipliers $\boldsymbol{\mu}_{\boldsymbol{g}_T}$ and $\boldsymbol{\mu}_{\boldsymbol{h}_T}$ along the simulation time or along the augmented Lagrangian iterations in case of OCPs.

- **Terminal penalty parameters**: This plot appears only if terminal constraints are defined. It illustrates the penalties $\boldsymbol{c}_{\boldsymbol{g}_T}$ and $\boldsymbol{c}_{\boldsymbol{h}_T}$ along the simulation time or along the augmented Lagrangian iterations in case of OCPs.

6 Tutorials

This chapter presents some application examples of GRAMPC and how to tune GRAMPC to improve the performance compared to the default settings. In addition to a model predictive control and an optimal control problem, a moving horizon estimation example is presented. Please note that the plot functions, described in Section 5.2.3, and the status of GRAMPC, see Section 4.8, also provide important information for tuning GRAMPC.

6.1 Model predictive control of a PMSM

The torque or current control of a permanent magnet synchronous machine (PMSM) is a challenging example for nonlinear constrained model predictive control. The following subsections illustrate the problem formulation as well as useful options of GRAMPC to improve the control performance. Corresponding m and C files can be found in the folder `<grampc_root>/examples/PMSM`.

6.1.1 Problem formulation

The system dynamics of a PMSM [4]

$$L_\mathrm{d}\tfrac{\mathrm{d}}{\mathrm{d}t}i_\mathrm{d} = -Ri_\mathrm{d} + L_\mathrm{q}\omega i_\mathrm{q} + u_\mathrm{d} \tag{6.1a}$$

$$L_\mathrm{q}\tfrac{\mathrm{d}}{\mathrm{d}t}i_\mathrm{q} = -Ri_\mathrm{q} - L_\mathrm{d}\omega i_\mathrm{d} - \omega\psi_\mathrm{p} + u_\mathrm{q} \tag{6.1b}$$

$$J\tfrac{\mathrm{d}}{\mathrm{d}t}\omega = \left(\tfrac{3}{2}z_\mathrm{p}\left(\psi_\mathrm{p}i_\mathrm{q} + i_\mathrm{d}i_\mathrm{q}(L_\mathrm{d} - L_\mathrm{q})\right) - \tfrac{\mu_\mathrm{f}}{z_\mathrm{p}}\omega - T_\mathrm{L}\right)z_\mathrm{p} \tag{6.1c}$$

$$\tfrac{\mathrm{d}}{\mathrm{d}t}\phi = \omega \tag{6.1d}$$

is given in the dq-coordinates. The system state $\boldsymbol{x} = [i_\mathrm{d}, i_\mathrm{q}, \omega, \phi]^\mathsf{T}$ comprises the dq-currents, the electrical rotor speed as well as the electrical angle. The dq-voltages serve as controls $\boldsymbol{u} = [u_\mathrm{d}, u_\mathrm{q}]^\mathsf{T}$. Further system parameters are the stator resistance $R = 3.5\,\Omega$, the number of pole-pairs $z_\mathrm{p} = 3$, the permanent magnet flux $\psi_p = 0.17\,\mathrm{V\,s}$, the dq-inductivities $L_\mathrm{d} = L_\mathrm{q} = 17.5\,\mathrm{mH}$, the moment of inertia $J = 0.9\,\mathrm{g\,m^2}$ as well as the friction coefficient $\mu_\mathrm{f} = 0.4\,\mathrm{mN\,m\,s}$.

The magnitude of the dq-currents is limited by the maximum current $I_\mathrm{max} = 10\,\mathrm{A}$, i.e.

$$i_\mathrm{abs} = i_\mathrm{d}^2 + i_\mathrm{q}^2 \le I_\mathrm{max}^2\,, \tag{6.2}$$

in order prevent damage of the electrical components. Another constraint concerns the dq-voltages. Through the modulation stage between the controller and the voltage source inverter, the dq-voltages are limited inside the circle

$$u_\mathrm{abs} = u_\mathrm{d}^2 + u_\mathrm{q}^2 \le U_\mathrm{max}^2 \tag{6.3}$$

with the maximum voltage $U_\mathrm{max} = 323\,\mathrm{V}$.

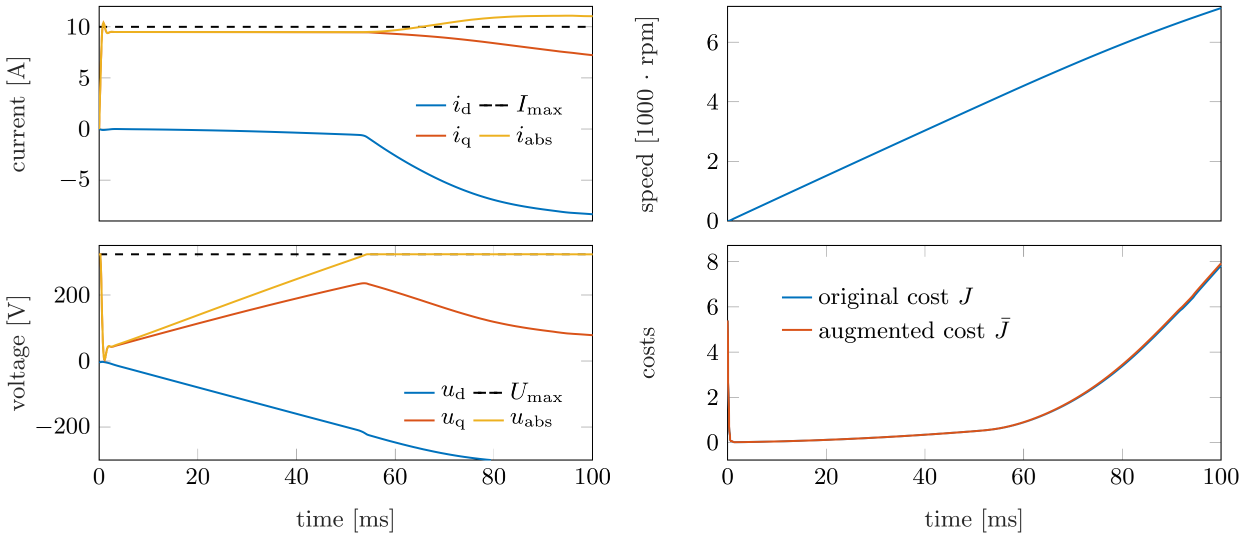

Figure 6.1: Simulated MPC trajectories for the PMSM example with default settings.

The optimal control problem

$$
\begin{aligned}
\min_{\boldsymbol{u}} \quad & J(\boldsymbol{u};\boldsymbol{x}_k) = \int_0^T l(\boldsymbol{x}(\tau),\boldsymbol{u}(\tau))\,\mathrm{d}\tau && \text{(6.4a)}\\
\text{s.t.} \quad & \dot{\boldsymbol{x}}(\tau) = \boldsymbol{f}(\boldsymbol{x}(\tau),\boldsymbol{u}(\tau))\,, \quad \boldsymbol{x}(0) = \boldsymbol{x}_k && \text{(6.4b)}\\
& h_1(\boldsymbol{x}(\tau)) = x_1(\tau)^2 + x_2(\tau)^2 - I_\mathrm{max}^2 \le 0 && \text{(6.4c)}\\
& h_2(\boldsymbol{u}(\tau)) = u_1(\tau)^2 + u_2(\tau)^2 - U_\mathrm{max}^2 \le 0 && \text{(6.4d)}\\
& \boldsymbol{u}(\tau) \in [\boldsymbol{u}_\mathrm{min}, \boldsymbol{u}_\mathrm{max}] && \text{(6.4e)}
\end{aligned}
$$

is subject to the system dynamics (6.4b) given by (6.1) and the constraints (6.4d)-(6.4e) given by (6.2) and (6.3). The control constraints (6.3) are nonlinear and are therefore handled by the augmented Lagrangian framework and not by the projection gradient method itself. It is therefore reasonable to add the box constraints (6.4e) with $\boldsymbol{u}_\mathrm{min} = [-U_\mathrm{max}, -U_\mathrm{max}]^\mathsf{T}$ and $\boldsymbol{u}_\mathrm{max} = [U_\mathrm{max}, U_\mathrm{max}]^\mathsf{T}$ to the OCP formulation to enhance the overall robustness of the algorithm. The cost functional consists of the integral part

$$
l(\boldsymbol{x},\boldsymbol{u}) = q_1(\boldsymbol{i}_\mathrm{d} - i_\mathrm{d,des})^2 + q_2(\boldsymbol{i}_\mathrm{q} - i_\mathrm{q,des})^2 + (\boldsymbol{u} - \boldsymbol{u}_\mathrm{des})^\mathsf{T}\boldsymbol{R}(\boldsymbol{u} - \boldsymbol{u}_\mathrm{des})\,, \tag{6.5}
$$

with the setpoints for the states $i_\mathrm{d,des}$, $i_\mathrm{q,des}$ and controls $\boldsymbol{u}_\mathrm{des} \in \mathbb{R}^2$ respectively. The weights are set to $q_1 = 8\,\mathrm{A}^{-2}$, $q_2 = 200\,\mathrm{A}^{-2}$ and $\boldsymbol{R} = \mathrm{diag}(0.001\,\mathrm{V}^{-2}, 0.001\,\mathrm{V}^{-2})$. The example considers a startup of the motor from standstill by defining the setpoints $i_\mathrm{d,des} = 0\,\mathrm{A}$ and $i_\mathrm{q,des} = 10\,\mathrm{A}$, corresponding to a constant torque demand of $7.65\,\mathrm{N\,m}$. The desired controls are set to $\boldsymbol{u}_\mathrm{d,des} = [0\,\mathrm{V}, 0\,\mathrm{V}]^\mathsf{T}$.

The resulting OCP is solved by GRAMPC with the sampling time $\Delta t = 125\,\mu\mathrm{s}$ (parameter `dt`) and the horizon $T = 5\,\mathrm{ms}$ (parameter `Thor`) using standard options almost exclusively. Only the number of discretization points `Nhor` $= 11$, the number of gradient iterations `MaxGradIter` $= 3$ and the number of augmented Lagrangian iterations `MaxMultIter` $= 3$ are adapted to the problem. In addition, the constraints tolerances `ConstraintsAbsTol` are set to 0.1% of the respective limit, i.e. $0.1\,\mathrm{A}^2$ and $104.5\,\mathrm{V}^2$. `PenaltyMin` is set to 2.5×10^{-7} by the estimation method of GRAMPC, see Section 4.4.3.

Figure 6.1 illustrates the simulation results. The setpoints are reached very fast and are stabilized almost exactly. However, an overshoot can be observed, which also leads to a small violation of the

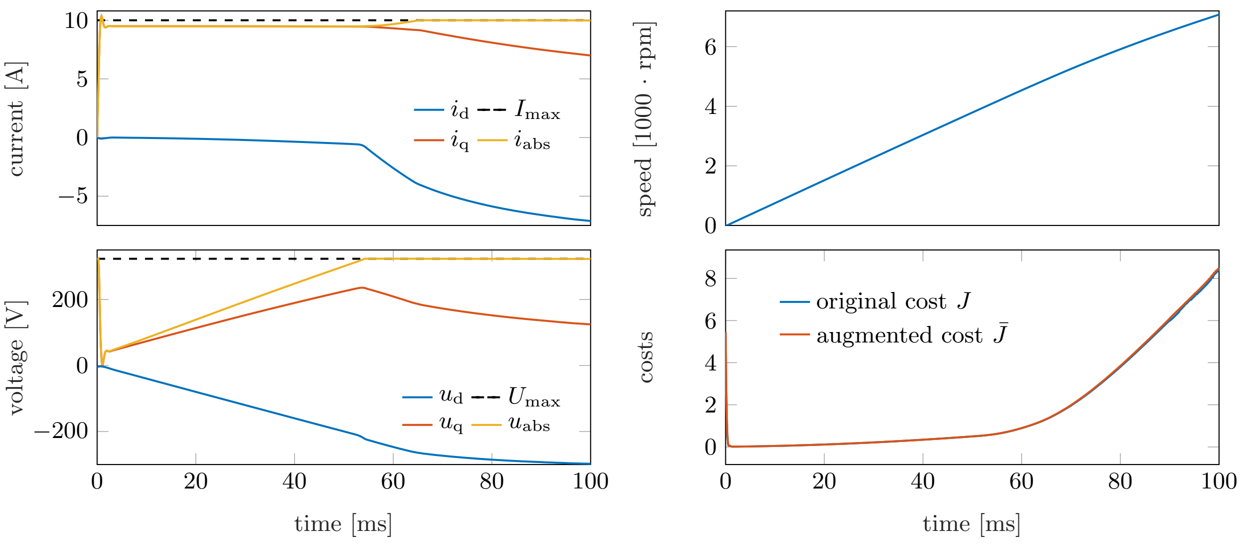

Figure 6.2: Simulated MPC trajectories for the PMSM example with scaled constraints.

dq-current constraint by 0.45 A. With increasing rotor speed, the voltage also increases until the voltage constraint becomes active. While the voltage constraint is almost exactly hold, the dq-current constraint is clearly violated. The figure shows that at the end of the simulation the dq-current constraint violation is more than 1A or 10%. Though a larger number of iterations might be used to reduce the constraint violation, the main reason for this deviation is that the nonlinear voltage and current constraints differ in several orders of magnitude. The next section therefore shows how to scale the problem in GRAMPC.

Also note that the increase of the cost functional does not indicate instability, but can be explained by the increasing speed, which affects the control term in the cost with the control setpoints ($\boldsymbol{u}_{\mathrm{d,des}} = [0\,\mathrm{V}, 0\,\mathrm{V}]^\mathsf{T}$). Moreover, the current setpoints ($i_{\mathrm{d,des}} = 0\,\mathrm{A}$, $i_{\mathrm{q,des}} = 10\,\mathrm{A}$) cannot be hold due to the constraints (6.2) and (6.3), which leads to an additional cost increase.

6.1.2 Constraints scaling

The two spherical constraints (6.2) and (6.3) lie in very different orders of magnitude, i.e. $I_\mathrm{max}^2 = 100\,\mathrm{A}^2$ and $U_\mathrm{max}^2 = 104\,329\,\mathrm{V}^2$. Consequently, the two constraints should be scaled by the maximum value

$$\frac{i_\mathrm{d}^2 + i_\mathrm{q}^2}{I_\mathrm{max}^2} - 1 \leq 0\,, \qquad \frac{u_\mathrm{d}^2 + u_\mathrm{q}^2}{U_\mathrm{max}^2} - 1 \leq 0\,. \tag{6.6}$$

This scaling can either be done by hand directly in the problem function or by activating the option `ScaleProblem` and setting `cScale`$= [I_\mathrm{max}^2, U_\mathrm{max}^2]$. The scaling option of GRAMPC, however, causes additional computing effort (approx. 45 % for the PMSM problem). Hence, this option is suitable for testing the scaling, but eventually should be done manually in the problem formulation to achieve the highest computational efficiency. In accordance with the scaling of the constraints, the tolerances are also adapted to 1×10^{-3} corresponding to 0.1% of the scaled constraints limits.

Besides the scaling and `PenaltyMin` that is set to 2×10^3 by the estimation routine of GRAMPC, all parameters and options are the same as in the last subsection. Please note that the estimation method for `PenaltyMin` strongly depends on reasonable constraint tolerances. In general, the method returns rather conservative values, which may lead to constraint violations if the order of magnitude of the constraints is very different.

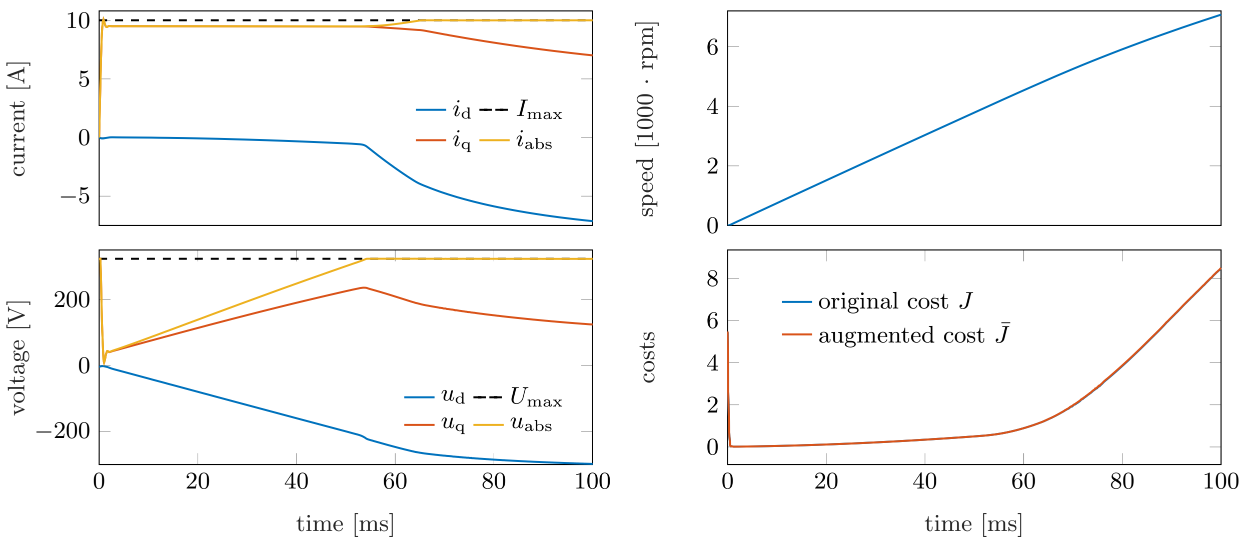

Figure 6.3: Simulated MPC trajectories for the PMSM example with scaled constraints and optimized penalty update.

Figure 6.2 shows a clear improvement in terms of the dq-current constraint that is now fully exploited. The only violation results from the overshoot at the beginning, which is in the same range as in the unscaled case (approx. 0.35 A). Further improvements, e.g. reduction of the overshoot, can be achieved by optimizing the penalty update as described in the next subsection.

6.1.3 Optimization of the penalty update

In order to improve compliance with the dq-current constraint at the beginning of the simulation, the number of augmented Lagrangian updates is increased. To this end `AugLagUpdateGradientRelTol` is raised to 1, which means that in every outer iteration an update of the multipliers and penalties is performed, even if the inner minimization is not converged. Furthermore, `PenaltyMin` is raised to 1×10^4 compared to the estimated value of 2×10^3. In addition, the plot of the step size, see the plot functions described in Section 5.2.3, shows that the maximum value $\alpha_{\max}$ is often used. Consequently, setting the maximum step size `LineSearchMax` to 10 allows larger optimization steps, especially at the beginning and at the end of the simulation. All other parameters and options, in particular the scaling options, are the same as in the previous subsection.

Figure 6.3 illustrates the simulation result with the optimized penalty update. The initial dq-current overshoot is further reduced and the constraint is only violated by less than 0.07 A. Furthermore, no oscillations occur in the costs and the augmented and original cost are almost the same, which indicates that GRAMPC is well tuned.

The computation time on a Windows 10 machine with Intel(R) Core(TM) i5-5300U CPU running at 2.3 GHz using the Microsoft Visual C++ 2013 Professional (C) compiler amounts to 0.032 ms. On the dSpace real-time hardware DS1202, the computation time is 0.13 ms.

6.2 Optimal control of a double integrator

This section describes how GRAMPC can be used to solve OCPs by considering the example of a double integrator problem. The problem formulation includes equality and inequality constraints. Both the control variable u and the end time T serve as optimization variables. In addition to the problem formulation of the OCP, one focus of the following discussion will be the appropriate

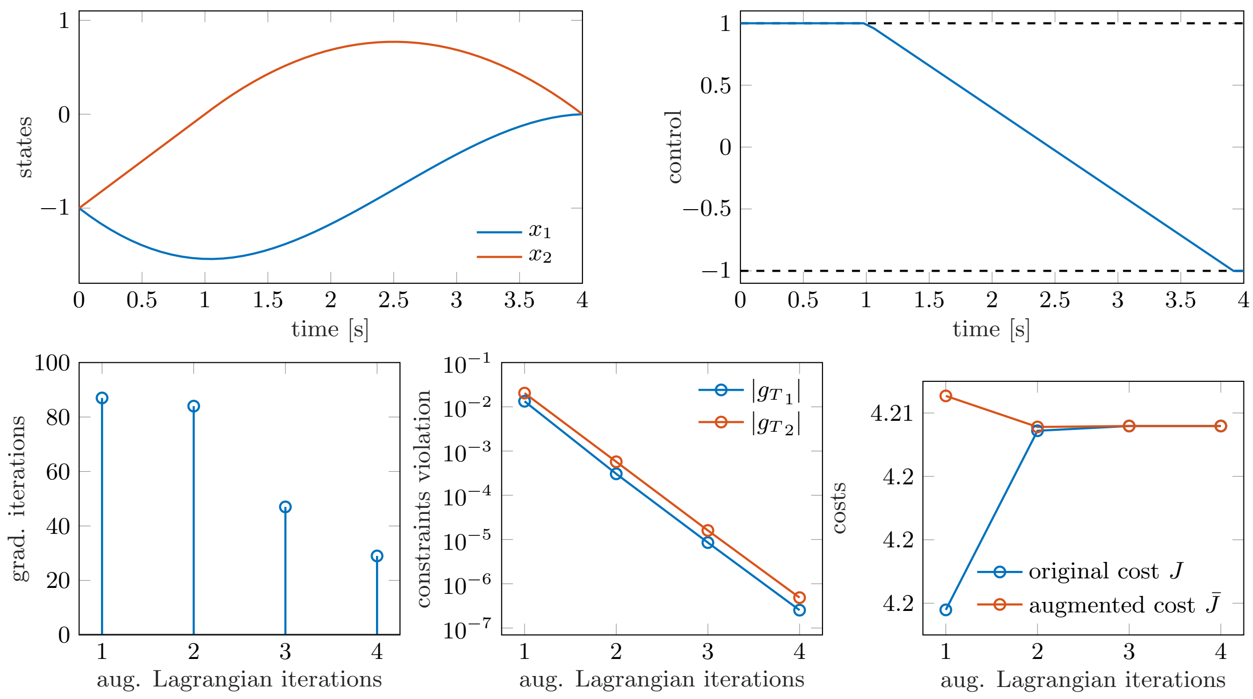

Figure 6.4: Numerical OCP solution of the double integrator problem with fixed end time.

convergence check of the augmented Lagrangian and gradient method, and the optimization of the end time.

6.2.1 Problem formulation

The double integrator problems reads as

$$\min_{u,T} \quad J(u,T;\boldsymbol{x}_0) = T + \int_0^T q_1 u^2(t)\,\mathrm{d}t \tag{6.7a}$$

$$\text{s.t.} \quad \dot{x}_1(t) = x_2(t)\,, \quad x_1(0) = x_{1,0} \tag{6.7b}$$

$$\dot{x}_2(t) = u(t)\,, \quad x_2(0) = x_{2,0} \tag{6.7c}$$

$$\boldsymbol{g}_T(\boldsymbol{x}(T))) = [x_1(T)\,, x_2(T)]^\mathsf{T} = \boldsymbol{0} \tag{6.7d}$$

$$h(\boldsymbol{x}(t)) = x_2(t) - 0.5 \le 0 \tag{6.7e}$$

$$u(t) \in [u_{\min}, u_{\max}]\,, \quad T \in [T_{\min}, T_{\max}] \tag{6.7f}$$

including two terminal equality constraints (6.7d) and one general inequality constraint (6.7e). The control task consists in a setpoint transition from the initial state $x_{1,0} = x_{2,0} = -1$ to the origin $x_1(T) = x_2(T) = 0$. The cost functional (6.7a) represents a trade-off between time and energy optimality depending on the weight $q_1 = 0.1$.

The problem is formulated in GRAMPC using the C file `probfct_DOUBLE_INTEGRATOR`, which can be found in `<grampc_root>/examples/Double_Integrator`. In particular, the terminal equality constraints (6.7d) are formulated in GRAMPC via the functions `gTfct`, `dgTdx_vec`, and `dgTdT_vec`. The number of terminal equality constraints is set to `NgT`=2 in the function `ocp_dim`. Similarly, the inequality constraint (6.7e) is formulated by means of the functions `hfct`, `dhdx_vec`, and `dhdu_vec` and setting to `Nh`=1 in `ocp_dim`. More details on implementing the OCP can be found in Section 3.2 and in the example provided in the folder `<grampc_root>/examples/Double_Integrator`.

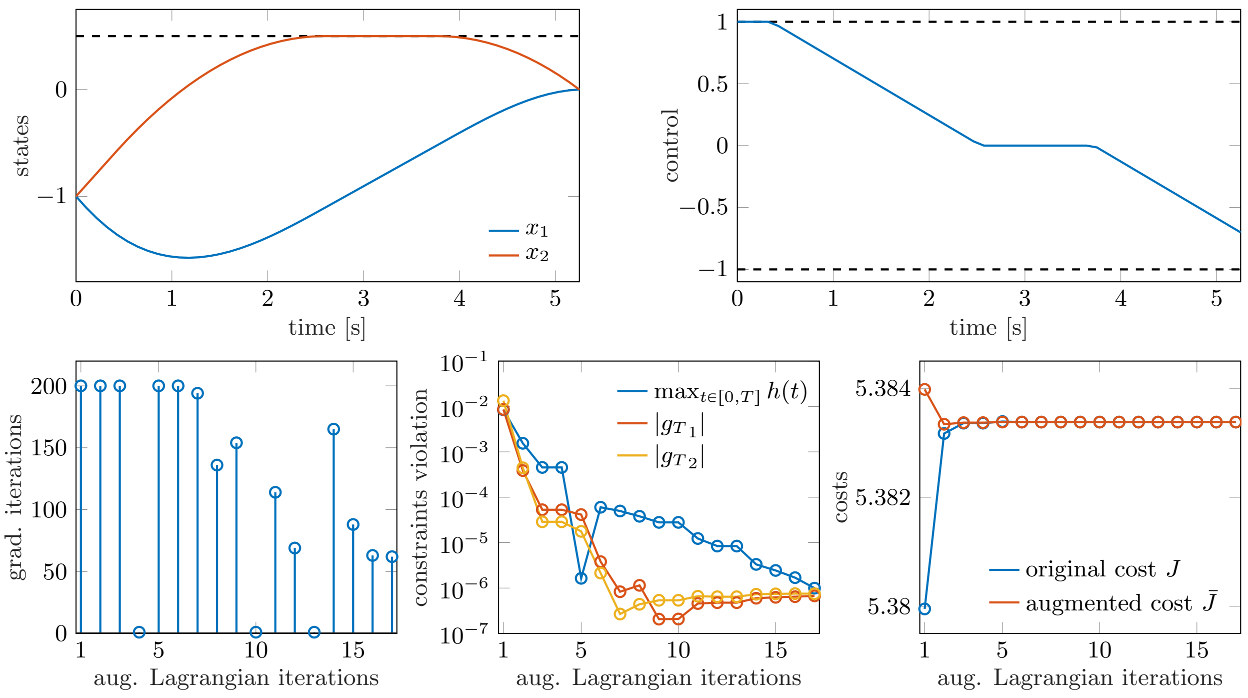

Figure 6.5: Numerical OCP solution of the double integrator problem with fixed end time and state constraint.

The options `OptimControl` and `OptimTime` are activated to optimize not only the control variable u but also the end time T. The lower and upper bounds of the optimzation variables are set to $u \in [-1, 1]$ and $T \in [1, 10]$ using the parameters `umin`, `umax`, `Tmin`, and `Tmax`, cf. Section 3.1. Note that the option `ShiftControl` is deactivated, as the shifting of the control trajectory typically only applies to MPC problems.

The option `ConvergenceCheck` is activated to terminate the augmented Lagrangian algorithm as soon as the state constraints are fulfilled with sufficient accuracy and the optimization variable has converged to an optimal value. To this end, the convergence criteria (4.32) and (4.33) are evaluated after each gradient and augmented Lagrangian step using the thresholds $\boldsymbol{\varepsilon}_{\boldsymbol{g_T}} = [1 \times 10^{-6}, 1 \times 10^{-6}]^\mathsf{T}$ and $\varepsilon_{\boldsymbol{h}} = 1 \times 10^{-6}$, cf. the option `ConstraintsAbsTol`. The threshold for checking convergence of the optimization variable $\varepsilon_{\text{rel,c}}$ is set to 1×10^{-9} via the option `ConvergenceGradientRelTol`. Note that the activation of the convergence check using the option `ConvergenceCheck` causes the gradient method to be aborted when the convergence condition $\eta^{i|j+1} \leq \varepsilon_{\text{rel,c}}$ defined by Equation (4.33) is reached.

6.2.2 Optimization with fixed end time

In a first scenario, the OCP (6.7) is numerically solved with the fixed end time $T = 4\,\text{s}$, i. e., the option `OptimTime` is set `off`, and without the inequality constraint (6.7e), i. e., the option `Inequality-Constraints` is also set `off`. The time interval $[0, T]$ is discretized using `Nhor`$= 50$ discretization points. During the augmented Lagrangian iterations, a decrease of the penalty parameters is prevented by setting the adaptation factor $\beta_{\text{de}} = 1$ (see option `PenaltyDecreaseFactor`). The increase factor β_{in} of the penalty parameter update (4.27) is set to the value 1.25. These settings ensure a fast convergence, since the very low tolerances $\boldsymbol{\varepsilon}_{\boldsymbol{g_T}}$ require high penalty parameters. However, starting with high penalties can lead to instabilities, as high penalties distort the optimization problem.

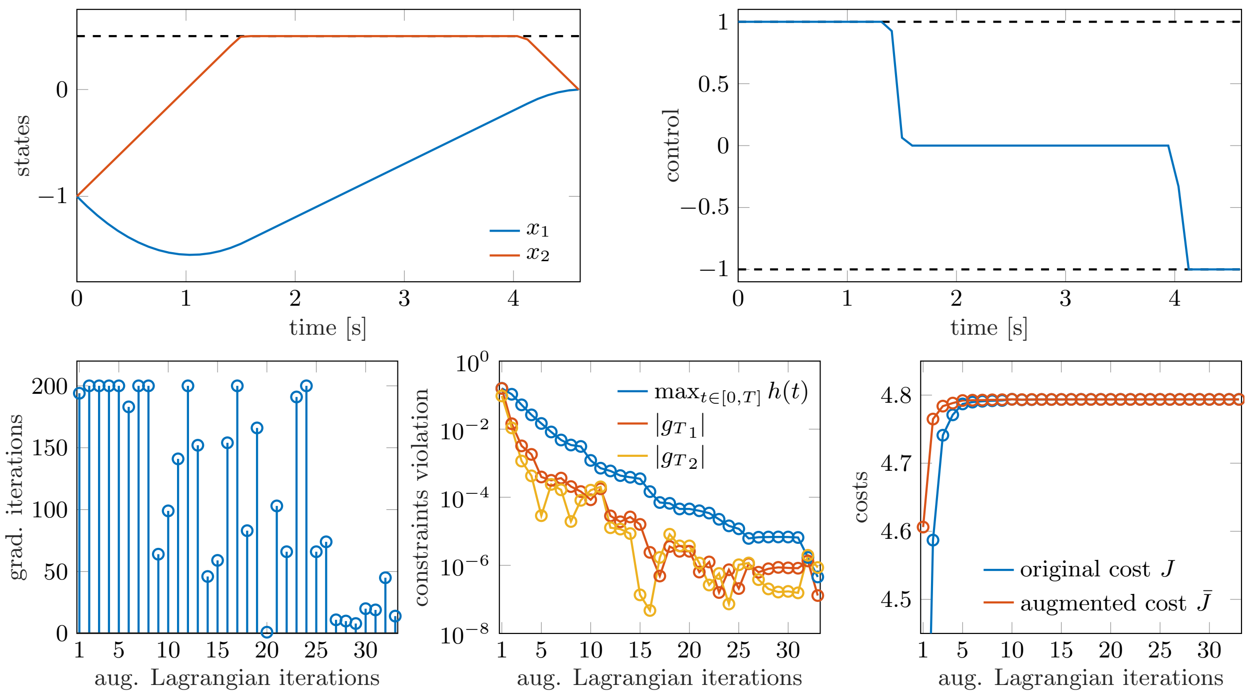

Figure 6.6: Numerical OCP solution of the double integrator problem with free end time and state constraint.

As shown in Figure 6.4, the state variables $\boldsymbol{x}(t)$ are transferred to the origin as specified by the terminal state constraints (6.7d). Note that only 5 augmented Lagrangian steps are required in total for numerically solving the state constrained optimization problem in accordance with the formulated convergence criterion for the terminal equality constraints and the optimization variable u. The number of gradient iterations varies in each augmented Lagrangian step as shown in Figure 6.4. The violation of the formulated terminal equality constraints (6.7d) continuously decreases below the specified thresholds $\boldsymbol{\varepsilon}_{\boldsymbol{g_T}} = [1\times10^{-6}, 1\times10^{-6}]^{\mathsf{T}}$. As a result, the augmented cost functional and the original cost functional converge to the same value, i. e. the so-called duality gap is zero.

In a second scenario, Figure 6.5 shows the optimal solution of OCP (6.7) with activated inequality constraint (6.7e) using the fixed end time $T = 5.25\,\text{s}$. Further problem settings are identical to the first simulation scenario. Again, the terminal equality constraints (6.7d) are satisfied by the optimal solution and the state variables $\boldsymbol{x}(t)$ are transferred to the origin. The control variable u is slightly adapted compared to the first simulation scenario in Figure 6.4 in order to comply with the inequality constraint (6.7e). In view of the additional inequality constraint, 17 augmented Lagrangian steps are required to be able to solve the optimization problem with sufficient accuracy.

The number of gradient iterations varies in each augmented Lagrangian step as shown in Figure .6.5. The violation of the state constraints is almost continuously decreased below the specified thresholds $\boldsymbol{\varepsilon}_{\boldsymbol{g_T}} = [1\times10^{-6}, 1\times10^{-6}]^{\mathsf{T}}$ and $\varepsilon_{\boldsymbol{h}} = 1\times10^{-6}$, respectively. As before, the augmented cost functional and the original cost functional converge to the same value. The computation time for solving the problem on a Windows 10 machine with an Intel(R) Core(TM) i5-5300U CPU running at 2.3 GHz using the Microsoft Visual C++ 2013 Professional (C) compiler amounts to 1.1 ms and 14.6 ms, respectively.

6.2.3 Optimization with free end time

In a third simulation scenario, the OCP (6.7) is numerically solved in the free end time setting. The initial end time is set to $T = 5.25\,\text{s}$ as given before. To weight the update of the end time T against the control update when updating the optimization variables according to Equation (4.12), the adaptation factor γ_T is set using the option `OptimTimeLineSearchFactor`. The value $\gamma_T = 1.75$ is used in the scenario. Note, however, that values $\gamma_T < 1$ typically increase the algorithmic stability at the expense of the calculation time and vice versa.

The numerical results for the free end time case are shown in Figure 6.6. In contrast to the first two simulation scenarios, the reduction of the end time below 4.60 s allows one to carry out the setpoint transition with a significantly more aggressive control trajectory u. The free end time optimization is a more challenging problem than before and is accompanied by a higher computational effort. This can be observed both in the larger number of augmented Lagrangian steps and gradient steps as well as in terms of the slower improvement of the violation of the state constraints.

Nevertheless, the state constraints are fulfilled at the last augmented Lagrangian step in accordance with the thresholds $\boldsymbol{\varepsilon}_{\boldsymbol{g_T}} = [1 \times 10^{-6}, 1 \times 10^{-6}]^\mathsf{T}$ and $\varepsilon_{\boldsymbol{h}} = 1 \times 10^{-6}$. The improvement of the control performance when optimizing the end time compared to a fixed end time can be specified by the lower value of the cost functional, cf. Figure 6.5 and 6.6. However, this results in a slightly increased computation time of 21.96 ms compared to 14.66 ms with a fixed end time on the same Windows 10 machine.

6.3 Moving horizon estimation of a CSTR

Continuous stirred tank reactors (CSTR) are a popular class of systems when it comes to the implementation of advanced nonlinear control methods. In this subsection, a CSTR model is used for an example implementation of a moving horizon estimation (MHE) used in closed loop with MPC. Corresponding m and C files can be found in the folder `<grampc_root>/examples/Reactor_CSTR`.

6.3.1 Problem formulation

The system dynamic are given by [13]

$$\begin{aligned}
\dot{c}_\mathrm{A} &= -k_1(T)c_\mathrm{A} - k_2(T)c_\mathrm{A}^2 + (c_\mathrm{in} - c_\mathrm{A})u_1 && \text{(6.8a)}\\
\dot{c}_\mathrm{B} &= k_1(T)c_\mathrm{A} - k_1(T)c_\mathrm{B} - c_\mathrm{B}u_1 && \text{(6.8b)}\\
\dot{T} &= -\delta(k_1(T)c_\mathrm{A}\Delta H_\mathrm{AB} + k_1(T)\Delta H_\mathrm{BC} + k_2(T)c_\mathrm{A}^2\Delta H_\mathrm{AD})\\
&\quad + \alpha(T_\mathrm{C} - T) + (T_\mathrm{in} - T)u_1 && \text{(6.8c)}\\
\dot{T}_\mathrm{C} &= \beta(T - T_\mathrm{C}) + \gamma u_2\,, && \text{(6.8d)}
\end{aligned}$$

where the monomer and product concentrations c_A and c_B, respectively, as well as the reactor and cooling temperature T and T_C form the state vector $\boldsymbol{x} = [c_\mathrm{A}, c_\mathrm{B}, T, T_\mathrm{C}]$. The two functions $k_1(T)$ and $k_2(T)$ are of Arrhenius type

$$k_i(T) = k_{i0}\exp\left(\frac{-E_i}{T/^\circ\mathrm{C} + 273.15}\right), \quad i = 1, 2\,. \tag{6.9}$$

The measured quantities are the two temperatures $\boldsymbol{y} = [T, T_C]^\mathsf{T}$. The controls $\boldsymbol{u} = [u_1, u_2]$, i.e. the normalized flow rate and the cooling power, are assumed to be known as well. Table 6.1 gives the parameters of the system that are passed to the GRAMPC problem functions via `userparam`. A more detailed description can be found in [13]. The control task at hand is the setpoint change between the two stationary points

$$\boldsymbol{x}_{\mathrm{des},1} = [1370\,\frac{\mathrm{kmol}}{\mathrm{m}^3}, 950\,\frac{\mathrm{kmol}}{\mathrm{m}^3}, 110.0\,^\circ\mathrm{C}, 108.6\,^\circ\mathrm{C}]^\mathsf{T}, \qquad \boldsymbol{u}_{\mathrm{des},1} = [5.0\,\mathrm{h}^{-1}, -1190\,\mathrm{kJ\,h}^{-1}]^\mathsf{T} \tag{6.10}$$

Table 6.1: Parameters of the CSTR mode [13].

Parameter	Value	Unit	Parameter	Value	Unit
α	30.828	h^{-1}	k_{20}	9.043×10^{6}	$\mathrm{m}^{3}\,\mathrm{mol}^{-1}\,\mathrm{h}^{-1}$
β	86.688	h^{-1}	E_1	9785.3	
δ	3.522×10^{-4}	$\mathrm{K\,kJ}^{-1}$	E_2	8560.0	
γ	0.1	kh^{-1}	ΔH_{AB}	4.2	$\mathrm{kJ\,mol}^{-1}$
T_{in}	104.9	°C	ΔH_{BC}	−11.0	$\mathrm{kJ\,mol}^{-1}$
c_{in}	5.1×10^{3}	$\mathrm{mol\,m}^{-3}$	ΔH_{AD}	−41.85	$\mathrm{kJ\,mol}^{-1}$
k_{10}	1.287×10^{12}	h^{-1}			

and

$$\boldsymbol{x}_{\mathrm{des},2} = [2020\,\frac{\mathrm{kmol}}{\mathrm{m}^3}, 1070\,\frac{\mathrm{kmol}}{\mathrm{m}^3}, 100.0\,°\mathrm{C}, 97.1\,°\mathrm{C}]^{\mathsf{T}}, \qquad \boldsymbol{u}_{\mathrm{des},2} = [5.0\,\mathrm{h}^{-1}, -2540\,\mathrm{kJ\,h}^{-1}]^{\mathsf{T}}. \tag{6.11}$$

The cost functional is designed quadratically

$$J(\boldsymbol{u}, \boldsymbol{x}_k) := \Delta\boldsymbol{x}(T)^{\mathsf{T}}\boldsymbol{P}\Delta\boldsymbol{x}(T) + \int_0^T \Delta\boldsymbol{x}(t)^{\mathsf{T}}\boldsymbol{Q}\Delta\boldsymbol{x}(t)\Delta\boldsymbol{u}(t)^{\mathsf{T}}\boldsymbol{R}\Delta\boldsymbol{u}(t) \tag{6.12}$$

in order to penalize the deviation of the state and control from the desired setpoint $(\boldsymbol{x}_{\mathrm{des},1}, \boldsymbol{u}_{\mathrm{des},1})$ with $\Delta\boldsymbol{x} = \boldsymbol{x} - \boldsymbol{x}_{\mathrm{des}}$ and $\Delta\boldsymbol{u} = \boldsymbol{u} - \boldsymbol{u}_{\mathrm{des}}$. The weight matrices are chosen as

$$\boldsymbol{P} = \mathrm{diag}(0.2, 1.0, 0.5, 0.2), \ \boldsymbol{R} = \mathrm{diag}(0.5, 5.0 \times 10^{-3}), \ \boldsymbol{Q} = \mathrm{diag}(0.2, 1.0, 0.5, 0.2). \tag{6.13}$$

The control task will be tackled by MPC using GRAMPC. In addition, an MHE using GRAMPC is designed to estimate the current state $\hat{\boldsymbol{x}}_k$ w.r.t. the measured temperatures $\boldsymbol{y} = [T, T_C]^{\mathsf{T}}$.

In analogy to the MPC formulation (3.1), moving horizon estimation is typically based on the online solution of a dynamic optimization problem

$$\min_{\hat{\boldsymbol{x}}_k} \quad J(\hat{\boldsymbol{x}}_k; \boldsymbol{u}, \boldsymbol{y}) = \int_{t_k - T}^{t_k} \|\hat{\boldsymbol{y}}(t) - \boldsymbol{y}(t)\|^2 \,\mathrm{d}t \tag{6.14a}$$

$$\text{s.t.} \quad \boldsymbol{M}\dot{\hat{\boldsymbol{x}}}(t) = \boldsymbol{f}(\hat{\boldsymbol{x}}(t), \boldsymbol{u}(t), t)\,, \quad \hat{\boldsymbol{x}}(t_k) = \hat{\boldsymbol{x}}_k \tag{6.14b}$$

$$\hat{\boldsymbol{y}}(t) = \boldsymbol{\sigma}(\hat{\boldsymbol{x}}(t)) \tag{6.14c}$$

that depends on the history of the measured outputs $\boldsymbol{y}(t)$ and controls $\boldsymbol{u}(t)$ in the past time window $[t_k - T, t_k]$. The solution of (6.14) yields the estimate of the state $\hat{\boldsymbol{x}}_k$ such that the difference between the measured output $\boldsymbol{y}(t)$ and the estimated output function (6.14c) is minimal in the sense of (6.14). GRAMPC solves this MHE problem by means of parameter optimization. To this end, the state at the beginning of the optimization horizon is defined as optimization variable, i.e. $\boldsymbol{p} = \hat{\boldsymbol{x}}(t_k - T)$.

Both MHE and MPC use a sampling rate of $\Delta t = 1\,\mathrm{s}$. A prediction horizon of $T = 20\,\mathrm{min}$ with 40 discretization points is used for the MPC, while a prediction horizon of $T = 10\,\mathrm{s}$ with 10 discretization points is used for the MHE. The MPC implementation uses three gradient iterations per sampling step, i.e. $(i_{\max}, j_{\max}) = (1, 3)$, while the implementation of the MHE uses only a single gradient iteration, i.e. $(i_{\max}, j_{\max}) = (1, 1)$. Note that because the MHE and MPC problems are defined without state constraints, the outer augmented Lagrangian loop causes no computational overhead, as GRAMPC skips the multiplier and penalty update. As the implementation of the MHE is not quite as straightforward as the MPC case, the next subsection describes the implementation process in more detail.

6.3.2 Implementation aspects

The following lines describe the implementation of the MHE problem with GRAMPC, the corresponding simulation results are shown in the next subsection. In a first step, the MHE problem (6.14) has to be transformed in a more suitable representation that can be tackled with the parameter optimization functionality of GRAMPC. To this end, a coordinate transformation

$$\tilde{\boldsymbol{x}}(\tau) = \hat{\boldsymbol{x}}(t_k - T + \tau) - \boldsymbol{p}\,, \quad \tilde{\boldsymbol{u}}(\tau) = \boldsymbol{u}(t_k - T + \tau)\,, \quad \tilde{\boldsymbol{y}}(\tau) = \boldsymbol{y}(t_k - T + \tau) \tag{6.15}$$

is used together with the corresponding time transformation from $t \in [t_k - T, t_k]^\mathsf{T}$ to the new time coordinate $\tau \in [0, T]$. In combination with the optimization variable $\boldsymbol{p} = \hat{\boldsymbol{x}}(t_k - T)$ and the homogeneous initial state $\tilde{\boldsymbol{x}}(0) = \boldsymbol{0}$, the optimization problem can be cast into the form

$$\min_{\boldsymbol{p}} \quad J(\boldsymbol{p}; \tilde{\boldsymbol{u}}, \tilde{\boldsymbol{y}}) = \int_0^T \|\hat{\boldsymbol{y}}(\tau) - \tilde{\boldsymbol{y}}(\tau)\|^2 \,\mathrm{d}\tau \tag{6.16a}$$

$$\text{s.t.} \quad \dot{\tilde{\boldsymbol{x}}}(\tau) = \boldsymbol{f}(\tilde{\boldsymbol{x}}(\tau) + \boldsymbol{p}, \tilde{\boldsymbol{u}}(\tau), t_k - T + \tau)\,, \quad \tilde{\boldsymbol{x}}(0) = \boldsymbol{0} \tag{6.16b}$$

$$\hat{\boldsymbol{y}}(\tau) = \boldsymbol{\sigma}(\tilde{\boldsymbol{x}}(\tau) + \boldsymbol{p})\,. \tag{6.16c}$$

The implementation of this optimization problem still requires to access the measurements $\tilde{\boldsymbol{y}}$ in the integral cost term. This is achieved by appending the measurements to `userparam` (see `startMHE.m` in `<grampc_root>/examples/Reactor_CSTR`)

```
% init array of last MHE-Nhor measurements of the two temperatures
xMeas_array = repmat(grampcMPC.param.x0(3:4), 1, grampcMHE.opt.Nhor);
grampcMHE.userparam(end-2*grampcMHE.opt.Nhor+1:end) = xMeas_array;
```

The measurements are updated in each iteration of the MPC/MHE loop, e.g.

```
% set values of last MHE-Nhor measurements of the two temperatures
xMeas_temp = xtemp(end,3:4) + randn(1,2)*4; % measurement noise
xMeas_array = [xMeas_array(3:end), xMeas_temp];
grampcMHE.userparam(end-2*grampcMHE.opt.Nhor+1:end) = xMeas_array;
```

When the number of discretization points and the horizon length is known, the measurements can easily be accessed in the problem description file `probfct_REACTOR_CSTR_MHE.c` in the following way:

```
typeRNum* pSys = (typeRNum*)userparam;
typeRNum* pCost = &pSys[14];
typeRNum* pMeas = &pSys[20];
typeInt index = (int)floor(t / 2.777777777777778e-04 + 0.00001);
typeRNum meas1 = pMeas[2 * index];
typeRNum meas2 = pMeas[1 + 2 * index];
```

The pointer `pMeas` is set to the 20th element of the `userparam` vector, since the first 14 are the system parameters given in Table 6.1 and the next six elements are the weights of the cost function. Also note that $2.778 \times 10^{-4}\,\mathrm{h} \approx \Delta t$. Since the order of magnitude of the individual states and controls differs a lot, the scaling option `ScaleProblem` with

$$\begin{aligned}
\boldsymbol{x}_{\text{scale}} &= [500\,\frac{\text{kmol}}{\text{m}^3}, 500\,\frac{\text{kmol}}{\text{m}^3}, 50\,^\circ\text{C}, 50\,^\circ\text{C}]^\mathsf{T} & \boldsymbol{x}_{\text{offset}} &= [500\,\frac{\text{kmol}}{\text{m}^3}, 500\,\frac{\text{kmol}}{\text{m}^3}, 50\,^\circ\text{C}, 50\,^\circ\text{C}]^\mathsf{T} \\
\boldsymbol{p}_{\text{scale}} &= [500\,\frac{\text{kmol}}{\text{m}^3}, 500\,\frac{\text{kmol}}{\text{m}^3}, 50\,^\circ\text{C}, 50\,^\circ\text{C}]^\mathsf{T} & \boldsymbol{p}_{\text{offset}} &= [500\,\frac{\text{kmol}}{\text{m}^3}, 500\,\frac{\text{kmol}}{\text{m}^3}, 50\,^\circ\text{C}, 50\,^\circ\text{C}]^\mathsf{T} \\
\boldsymbol{u}_{\text{scale}} &= [16\,\text{h}^{-1}, 4500\,\text{kJ}\,\text{h}^{-1}]^\mathsf{T} & \boldsymbol{u}_{\text{offset}} &= [19\,\text{h}^{-1}, -4500\,\text{kJ}\,\text{h}^{-1}]^\mathsf{T}
\end{aligned} \tag{6.17}$$

is activated.

6.3.3 Evaluation

The moving horizon estimator is evaluated in conjunction with the MPC. The state estimates are initialized with an initial disturbance $\delta \boldsymbol{p} = [100\,\mathrm{kmol\,m^{-3}},\ 100\,\mathrm{kmol\,m^{-3}},\ 5\,°\mathrm{C},\ 7\,°\mathrm{C}]^\mathsf{T}$. For a more realistic setting, white Gaussian noise with zero mean and a standard deviation of 4 °C is added to the measurements $\boldsymbol{y} = [T, T_C]^\mathsf{T}$.

Figure 6.7 shows the simulation results from the closed loop simulation of the MHE in conjunction with the MPC. The estimates $\hat{\boldsymbol{x}}_k$ quickly converge to the actual states (ground truth), as e.g. can be seen in the upper right plot. Furthermore, the cost of the MPC quickly converges to zero after each setpoint change at $t = 0\,\mathrm{h}$ and $t = 1.5\,\mathrm{h}$, respectively, which shows the good performance of the combined MPC/MHE problem.

The corresponding computation times of GRAMPC amount to 58 µs and 11 µs per MPC and MHE step, respectively, on a Windows 10 machine with Intel(R) Core(TM) i5-5300U CPU running at 2.3 GHz using the Microsoft Visual C++ 2013 Professional (C) compiler.

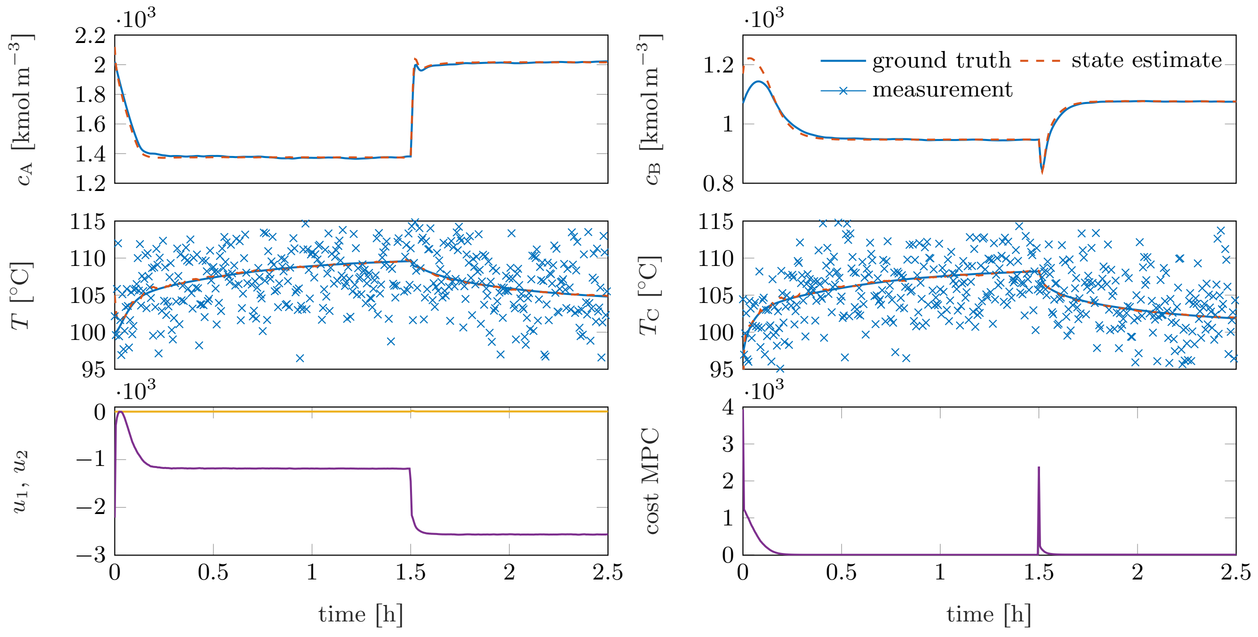

Figure 6.7: Simulated MHE/MPC trajectories for the CSTR reactor example.

6.4 Differential algebraic equations

This section first introduces the DAE-system, before implementation aspects regarding the dedicated DAE-solver RODAS are considered. Finally, the example is evaluated.

6.4.1 Problem formulation

The problem at hand is a toy example to illustrate the functionality of GRAMPC with regard to the solution of DAEs. It consists of two differential integrator states and one algebraic state. In addition, this algebraic state is subject to an equality constraint. The corresponding MPC problem is given by

$$\begin{aligned}
\min_{u} \quad & J(\boldsymbol{x},\boldsymbol{u}) = \int_0^T \frac{1}{2}\left(\Delta\boldsymbol{x}\boldsymbol{Q}\Delta\boldsymbol{x}^\mathsf{T} + \boldsymbol{u}\boldsymbol{R}\boldsymbol{u}^\mathsf{T}\right)\,\mathrm{d}t && \text{(6.18a)}\\
\text{s.t.} \quad & \dot{x}_1(t) = u_1(t)\,, \qquad x_1(0) = x_{1,0} && \text{(6.18b)}\\
& \dot{x}_2(t) = u_2(t)\,, \qquad x_2(0) = x_{2,0} && \text{(6.18c)}\\
& \quad 0 = x_1(t) + x_2(t) - x_3(t) && \text{(6.18d)}\\
& g(\boldsymbol{x}(t))) = x_3(t) - 1 = 0 && \text{(6.18e)}\\
& \boldsymbol{u}(t) \in [\boldsymbol{u}_{\min}, \boldsymbol{u}_{\max}]\,, && \text{(6.18f)}
\end{aligned}$$

where $\Delta\boldsymbol{x} = \boldsymbol{x} - \boldsymbol{x}_{\text{des}}$ and the weight matrices are chosen as $\boldsymbol{Q} = \text{diag}(500, 0, 0)$ and $\boldsymbol{R} = \text{diag}(1, 1)$, respectively. The target of the MPC formulation is to steer the first differential state to the desired value, while remaining on the manifold defined by $x_1(t) + x_2(t) = 1$. Note that this equation results from substituting the algebraic equation (6.18d) into the constraint (6.18e). Even though it would be possible to do this substitution and solve the resulting problem, the purpose of this example is to illustrate the solution of a DAE.

The DAE given in (6.18) can be rewritten with a mass matrix $\boldsymbol{M}$, i.e.

$$\underbrace{\begin{pmatrix} 1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 0 \end{pmatrix}}_{\boldsymbol{M}} \dot{\boldsymbol{x}} = \underbrace{\begin{pmatrix} u_1\\ u_2\\ x_1 + x_2 - x_3 \end{pmatrix}}_{\boldsymbol{f}(\boldsymbol{x},\boldsymbol{u})}. \tag{6.19}$$

Clearly, the mass matrix is different from the identity matrix and singular, i.e. the inverse does not exist and therefore the solver RODAS is used to integrate the system dynamics as well as the corresponding adjoint dynamics.

6.4.2 Implementation aspects

To solve the MPC problem for a DAE, the integrator RODAS has to be used and some additional options have to be set. Furthermore, some additional functions have to be implemented in the `probfct`-file.

The options are described in Section 4.2.2. In the example at hand, the right hand side of the system dynamics is not explicitly dependent on the time t and therefore `IFCN` is set to zero. The next option concerns the calculation of $\frac{\partial f}{\partial t}$ and $\frac{\partial^2 H}{\partial x \partial t}$. It can either be set to zero (i.e. `IDFX` $= 0$) and finite differences are utilized or set to one (i.e. `IDFX` $= 1$) and the analytical solutions implemented in the functions `dfdt` and `dHdxdt` are called. The third option `IJAC` determines if the numerical (i.e. finite differences) or the analytical solution (i.e. `dfdx` and `dfdxtrans`) is used to compute the Jacobians $\frac{\partial f}{\partial x}$ and $(\frac{\partial f}{\partial x})^\mathrm{T} = \frac{\partial^2 H}{\partial x \partial \lambda}$. The next option (`IMAS`) determines if the the mass matrix is equal to the identity matrix (i.e. `IMAS` $= 0$) or if it is specified by the functions `Mfct` and `Mtrans` (i.e. `IMAS` $= 1$). In the current example, the mass matrix is singular (not the identity matrix) and therefore the option is set to one. The remaining options regard the size of the Jacobian and the mass matrix. The number of non-zero lower and upper diagonals of the Jacobian are given by `MLJAC` and `MUJAC`, respectively. In our case, we have a full matrix and therefore set both options to the system dimension, i.e. $N_{\boldsymbol{x}}$. The only non-zero entries of the mass matrix lie on the main diagonal. Thus, the corresponding options (i.e. `MLMAS` and `MUMAS`) are set to zero. Note that one has to be careful, if the Jacobian or mass matrix are sparse, since the lower and upper diagonals are padded with zeros. This is shown for the example matrix in Figure 6.8 with the corresponding code in the following Example.

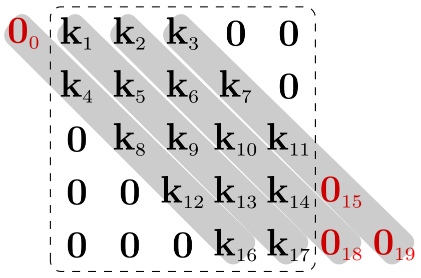
0_0 k_1 k_2 k_3 0 0
k_4 k_5 k_6 k_7 0
0 k_8 k_9 k_{10} k_{11}
0 0 k_{12} k_{13} k_{14} 0_{15}
0 0 0 k_{16} k_{17} 0_{18} 0_{19}

Figure 6.8: Example mass matrix with the index i showing at which position in the output array (i.e. out[i]) the corresponding value has to be written, cf. the example above.

Example (C-Code for the mass function illustrated in Figure 6.8)

```
void Mfct(typeRNum *out, typeUSERPARAM *userparam)
{
    /* row 1 */
    out[0]  = 0;
    out[1]  = k;
    out[2]  = k;
    out[3]  = k;
    /* row 2 */
    out[4]  = k;
    out[5]  = k;
    out[6]  = k;
    out[7]  = k;
    /* row 3 */
    out[8]  = k;
    out[9]  = k;
    out[10] = k;
    out[11] = k;
    /* row 4 */
    out[12] = k;
    out[13] = k;
    out[14] = k;
    out[15] = 0;
    /* row 5 */
    out[16] = k;
    out[17] = k;
    out[18] = 0;
    out[19] = 0;
};
```

6.4.3 Evaluation

Figure 6.9 shows the simulated trajectories for three set point changes. The setpoint of the first state x_1 changes from 1 to 0 at 0 s, from 0 to 0.5 after 1 s, and finally from 0.5 to 1 after 2 s. Due to the algebraic state and the equality constraint, the trajectory of the second state x_2 has to be the mirror image of x_1 around 0.5, which can be observed in the upper left plot. The corresponding controls in the upper right plot are also mirrored. The constraint violation during the simulation is shown in the lower left plot of Figure 6.9. The allowed constraint violation was set to 1×10^{-4}, which is approximately met. Lastly, the lower right plot shows the original costs, i.e. (6.18a) and the augmented costs. Both of which quickly converge to zero after each set point change (after 0, 1 and 2 s, respectively).

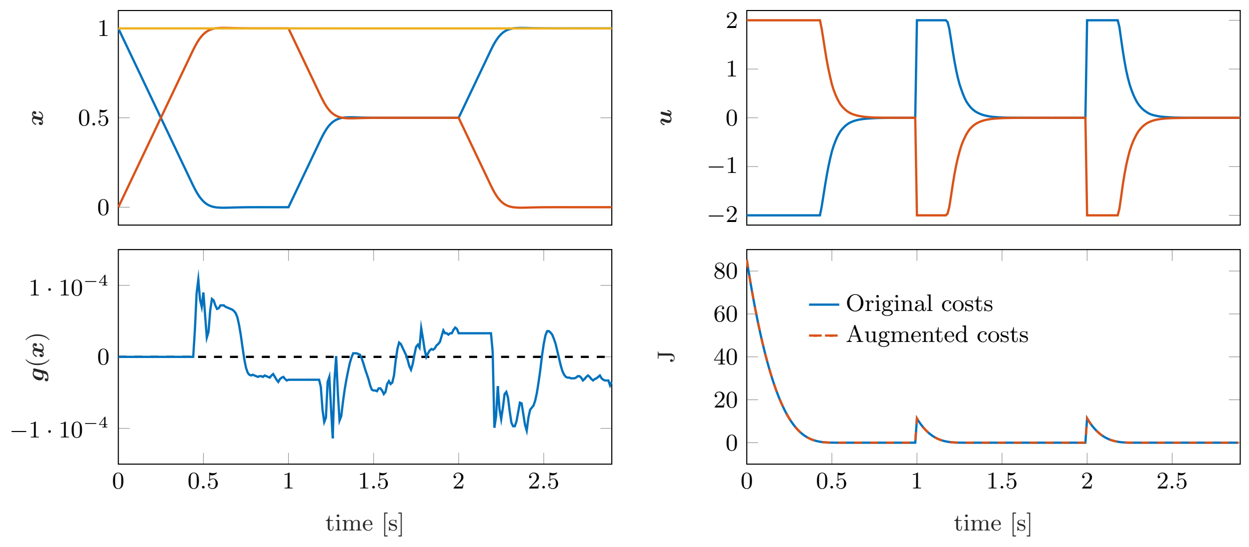

Figure 6.9: Simulated MPC trajectories for the DAE-example (6.18).

6.5 Constraint Tuning

This section shows how the options of GRAMPC can be adjusted in such a way that the performance in terms of computation time and optimality is improved. To this end, a specific OCP problem is considered. However, the approach can serve as template for different problems.

6.5.1 Problem formulation

The system at hand is a double integrator with one inequality constraint and a terminal equality constraint for each state [2]. The OCP problem is then defined as

$$\min_{u} \quad J(\boldsymbol{x}, u) = \int_0^T r u^2 \,\mathrm{d}t \tag{6.20a}$$

$$\text{s.t.} \quad \dot{x}_1(t) = x_2(t)\,, \qquad x_1(0) = x_{1,0} \tag{6.20b}$$

$$\dot{x}_2(t) = u(t)\,, \qquad x_2(0) = x_{2,0} \tag{6.20c}$$

$$h(\boldsymbol{x}(t)) = x_1(t) - 0.1 \leq 0 \tag{6.20d}$$

$$g_{\mathrm{T},1}(\boldsymbol{x}(T))) = x_1(T) = 0 \tag{6.20e}$$

$$g_{\mathrm{T},2}(\boldsymbol{x}(T))) = x_2(T) + 1 = 0, \tag{6.20f}$$

with the weight $r = 0.5$ and the initial state $\boldsymbol{x}(0) = [0, 1]^{\mathsf{T}}$. The target of the problem is to steer the double integrator states to the terminal state $\boldsymbol{x}(T) = [0, -1]^{\mathsf{T}}$ without violating the inequality constraint (6.20d). The time in which the set point change should be executed is set to $T = 1\,\mathrm{s}$.

6.5.2 Tuning approach

In the listing below, the step by step approach of tuning the parameters of the augmented Lagrangian algorithm are detailed. The computation time as well as the number of outer and inner iterations are shown in Table 6.2 along with the corresponding option that was added or changed in each step.

1. The choice of the initial penalty parameter is crucial for numerical conditioning and therefore convergence. A value that is too high will initially put an unnecessary amount of weight on

Table 6.2: Computation time and outer / inner iteration count for the different settings. The last column shows which additional parameter was set different from the initial values in each step.

Step	Time	MultIter	GradIter (mean)	Additional option
0.	343 ms	3883	10.3	Default settings
1.	129 ms	637	22.4	`grampc_estim_penmin`, cf. Section 4.4.3
2.	59 ms	171	42.0	`PenaltyIncreaseFactor` = 1.5
3.	36 ms	98	46.6	`PenaltyIncreaseThreshold` = 0.75
4.	22 ms	42	61.2	`LineSearchInit` = 5e-7
5.	19 ms	101	22.8	`PenaltyMin` = 2e4

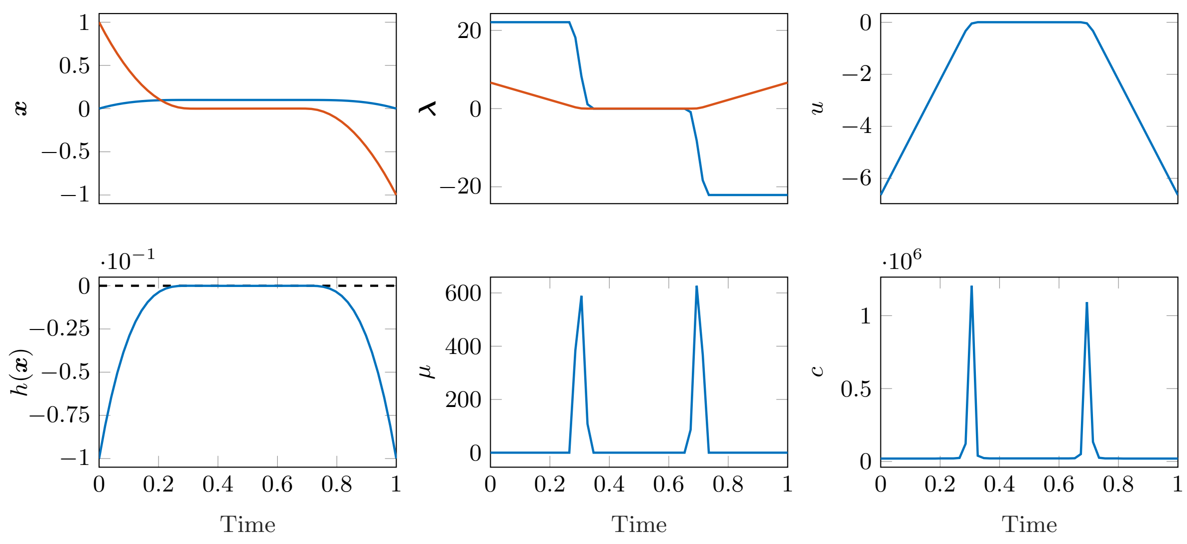

Figure 6.10: The prediction plot of GRAMPC for the double integrator example (6.20).

constraint satisfaction and mostly ignore the optimality. If the value is too low, the opposite will happen, i.e. at first the cost function is decreased at the cost of constraint violation. This will be especially detrimental in MPC applications. If no prior knowledge is available, it is recommended to use the function `grampc_estim_penmin` (or the corresponding Cmex interface `grampc_estim_penmin_Cmex`). For the example at hand, this reduces the computation time by approximately a third. Figure 6.10 shows the simulation results using the estimated value for the `PenaltyMin`.

2. While the convergence speed is significantly increased, the penalty parameters at 0.3 s and 0.7 s are several magnitudes greater than the initial value. Since this huge penalty parameter occurs only at two points during the simulation, it is advisable to set the `PenaltyIncreaseFactor` to a bigger value. This again reduces the computation time by more than half, since fewer increases of the penalty parameter during the outer iterations are necessary. The value should not be chosen too big, as this will have an adverse effect on the numerical conditioning and convergence.

3. In accordance with the previous step, the threshold to increase the penalty parameter, i.e. `PenaltyIncreaseThreshold` is lowered in order to increase the penalty parameter more aggressively. This step almost doubles the convergence speed. Note that this step and the previous step are interchangeable.

4. Another common tuning possibility is the initial step size of the line search, especially if one of the explicit methods is used, cf. Section 4.3.2. Note that the initial value `LineSearchInit` is used in the case that the explicit formula results in a negative step size. One approach is to use the `LineSearchExpAutoFallback` option. However, problem specific tuning (mostly trial and error) can result in a significant performance boost. In the example at hand, this results in approximately 40 % faster convergence speed.

5. To further optimize the parameters, the estimation function for the minimal penalty parameter can be deactivated again and a better value for `PenaltyMin` be used (note that the parameter is increased until there is no further improvement or an decrease in performance). This results in an additional 15 % decrease of computation time.

A Appendix

In addition to a list of all parameters and algorithmic options of GRAMPC, this appendix contains a short description of the structure variable `grampc`. Essential C functions of the GRAMPC project are also listed.

A.1 List of parameters

Table A.1 gives an overview of the problem-specific parameters of GRAMPC in terms of the parameter name and type as well as the admissible range and default values (if available), whereby the symbol $\mathrm{e}_n = [1, \dots, 1]^\mathsf{T} \in \mathbb{R}^n$ denotes an n-dimensional column vector.

Table A.1: Problem-specific parameters.

Parameter name	**Type**	**Allowed values**	**Default**
`x0`	`typeRNum*`	$(-\infty, \infty)$	$0 \cdot \mathrm{e}_{\mathrm{Nx}}$
`xdes`	`typeRNum*`	$(-\infty, \infty)$	$0 \cdot \mathrm{e}_{\mathrm{Nx}}$
`u0`	`typeRNum*`	$(-\infty, \infty)$	$0 \cdot \mathrm{e}_{\mathrm{Nu}}$
`udes`	`typeRNum*`	$(-\infty, \infty)$	$0 \cdot \mathrm{e}_{\mathrm{Nu}}$
`umax`	`typeRNum*`	$(-\infty, \infty)$	$+\infty \cdot \mathrm{e}_{\mathrm{Nu}}$
`umin`	`typeRNum*`	$(-\infty, \infty)$	$-\infty \cdot \mathrm{e}_{\mathrm{Nu}}$
`p0`	`typeRNum*`	$(-\infty, \infty)$	$0 \cdot \mathrm{e}_{\mathrm{Np}}$
`pmax`	`typeRNum*`	$(-\infty, \infty)$	$+\infty \cdot \mathrm{e}_{\mathrm{Np}}$
`pmin`	`typeRNum*`	$(-\infty, \infty)$	$-\infty \cdot \mathrm{e}_{\mathrm{Np}}$
`Thor`	`typeRNum`	$(0, \infty)$	To be provided
`Tmax`	`typeRNum`	$(0, \infty)$	10^8
`Tmin`	`typeRNum`	$(0, \infty)$	10^{-8}
`dt`	`typeRNum`	$(0, \infty)$	To be provided
`t0`	`typeRNum`	$(-\infty, \infty)$	0
`userparam` (in C)	`void*`	User-defined	`NULL`
`userparam` (in MATLAB)	`typeRNum*`	$(-\infty, \infty)$	`[]`

A description of all parameters is as follows (see also Section 3.1):

- `x0`: Initial state vector $\boldsymbol{x}(t_0) = \boldsymbol{x}_0$ at the corresponding sampling time t_0..
- `xdes`: Desired (constant) setpoint vector for the state variables $\boldsymbol{x}$.
- `u0`: Initial value of the control vector $\boldsymbol{u}(t) = \boldsymbol{u}_0 = \text{const.}$, $t \in [0, T]$ that is used in the first iteration of GRAMPC.
- `udes`: Desired (constant) setpoint vector for the control variables $\boldsymbol{u}$.

- `umin`, `umax`: Lower and upper bounds for the control variables $\boldsymbol{u}$.
- `p0`: Initial value of the parameter vector $\boldsymbol{p} = \boldsymbol{p}_0$ that is used in the first iteration of GRAMPC.
- `pmin`, `pmax`: Lower and upper bounds for the parameters $\boldsymbol{p}$.
- `Thor`: Prediction horizon T or initial value if the end time is optimized.
- `Tmin`, `Tmax`: Lower and upper bound for the prediction horizon T.
- `dt`: Sampling time Δt of the considered system for model predictive control or moving horizon estimation. Required for prediction of next state `grampc.sol.xnext` and for the control shift, see Section 4.7.
- `t0`: Current sampling instance t_0 that is provided to the time-varying system dynamics (3.1b).
- `userparam`: Further problem-specific parameters, e.g. system parameters or weights in the cost functions that are passed to the problem functions via a `void`-pointer in C or `typeRNum` array in MATLAB.

A.2 List of options

Table A.2 gives an overview of the algorithmic options of GRAMPC in terms of the option name and type as well as the allowed and the default values (if available).

Table A.2: Algorithmic options.

Option name	**Type**	**Allowed values**	**Default**
`Nhor`	`typeInt`	$[2, \infty)$	30
`MaxGradIter`	`typeInt`	$[1, \infty)$	2
`MaxMultIter`	`typeInt`	$[1, \infty)$	1
`ShiftControl`	`typeChar*`	`on` / `off`	`on`
`IntegralCost`	`typeChar*`	`on` / `off`	`on`
`TerminalCost`	`typeChar*`	`on` / `off`	`on`
`IntegratorCost`	`typeChar*`	`trapezodial` / `simpson`	`trapezodial`
`Integrator`	`typeChar*`	`euler` / `modeuler` / `heun` / `ruku45` / `rodas`	`heun`
`IntegratorRelTol`	`typeRNum`	$(0, \infty)$	10^{-6}
`IntegratorAbsTol`	`typeRNum`	$(0, \infty)$	10^{-8}
`IntegratorMinStepSize`	`typeRNum`	$(0, \infty)$	*eps*
`IntegratorMaxSteps`	`typeInt`	$[1, \infty)$	10^{8}
`FlagsRodas`	`typeInt*`	see description	see description
`LineSearchType`	`typeChar*`	`adaptive` / `explicit1` / `explicit2`	`explicit2`
`LineSearchExpAutoFallback`	`typeChar*`	`on` / `off`	`on`
`LineSearchMax`	`typeRNum`	$(0, \infty)$	0.75
`LineSearchMin`	`typeRNum`	$(0, \infty)$	10^{-10}
`LineSearchInit`	`typeRNum`	$(0, \infty)$	10^{-4}
`LineSearchAdaptAbsTol`	`typeRNum`	$[0, \infty)$	10^{-6}
`LineSearchAdaptFactor`	`typeRNum`	$(1, \infty)$	3/2

Table A.2: Algorithmic options.

Option name	**Type**	**Allowed values**	**Default value**
`LineSearchIntervalTol`	`typeRNum`	(0, 0.5)	0.1
`LineSearchIntervalFactor`	`typeRNum`	(0, 1)	0.85
`OptimControl`	`typeChar*`	`on` / `off`	`on`
`OptimParam`	`typeChar*`	`on` / `off`	`off`
`OptimParamLineSearchFactor`	`typeRNum`	$(0, \infty)$	1.0
`OptimTime`	`typeChar*`	`on` / `off`	`off`
`OptimTimeLineSearchFactor`	`typeRNum`	$(0, \infty)$	1.0
`ScaleProblem`	`typeChar*`	`on` / `off`	`off`
`xScale`	`typeRNum*`	$(-\infty, \infty)$	e_{Nx}
`xOffset`	`typeRNum*`	$(-\infty, \infty)$	$0 \cdot \mathrm{e}_{\mathrm{Nx}}$
`uScale`	`typeRNum*`	$(-\infty, \infty)$	e_{Nu}
`uOffset`	`typeRNum*`	$(-\infty, \infty)$	$0 \cdot \mathrm{e}_{\mathrm{Nu}}$
`pScale`	`typeRNum*`	$(-\infty, \infty)$	e_{Np}
`pOffset`	`typeRNum*`	$(-\infty, \infty)$	$0 \cdot \mathrm{e}_{\mathrm{Np}}$
`TScale`	`typeRNum`	$(0, \infty)$	1.0
`TOffset`	`typeRNum`	$(-\infty, \infty)$	0.0
`JScale`	`typeRNum`	$(0, \infty)$	1.0
`cScale`	`typeRNum*`	$(-\infty, \infty)$	e_{Nc}
`EqualityConstraints`	`typeChar*`	`on` / `off`	`on`
`InequalityConstraints`	`typeChar*`	`on` / `off`	`on`
`TerminalEqualityConstraints`	`typeChar*`	`on` / `off`	`on`
`TerminalInequalityConstraints`	`typeChar*`	`on` / `off`	`on`
`ConstraintsHandling`	`typeChar*`	`extpen` / `auglag`	`auglag`
`ConstraintsAbsTol`	`typeRNum*`	$(0, \infty)$	$10^{-4} \cdot \mathrm{e}_{\mathrm{Nc}}$
`MultiplierMax`	`typeRNum`	$(0, \infty)$	10^{6}
`MultiplierDampingFactor`	`typeRNum`	[0, 1]	0.0
`PenaltyMax`	`typeRNum`	$(0, \infty)$	10^{6}
`PenaltyMin`	`typeRNum`	$(0, \infty)$	10^{0}
`PenaltyIncreaseFactor`	`typeRNum`	$[1, \infty)$	1.05
`PenaltyDecreaseFactor`	`typeRNum`	[0, 1]	0.95
`PenaltyIncreaseThreshold`	`typeRNum`	$[0, \infty)$	1.0
`AugLagUpdateGradientRelTol`	`typeRNum`	[0, 1]	10^{-2}
`ConvergenceCheck`	`typeChar*`	`on` / `off`	`off`
`ConvergenceGradientRelTol`	`typeRNum`	[0, 1]	10^{-6}

A description of all options is as follows:

- `Nhor`: Number of discretization points within the time interval $[0, T]$.
- `MaxMultIter`: Sets the maximum number of augmented Lagrangian iterations $i_{\max} \geq 1$. If the option `ConvergenceCheck` is activated, the algorithm evaluates the convergence criterion and terminates if the inner minimization converged and all constraints are satisfied within the tolerance defined by `ConstraintsAbsTol`.

- `MaxGradIter`: If the option `ConvergenceCheck` is activated, the algorithm terminates the inner loop as soon as the convergence criterion is fulfilled.
- `ShiftControl`: Activates or deactivates the shifting of the control trajectory and the adaptation of T in case of a free end time, i.e., if `OptimTime` is active.

- `IntegralCost`, `TerminalCost`: Indicate if the integral and/or terminal cost functions are defined.
- `IntegratorCost`: This option specifies the integration scheme for the cost functionals. Possible values are `trapezodial` and `simpson`.

- `Integrator`: This option specifies the integration scheme for the system and adjoint dynamics. Possible values are `euler`, `modeuler` and `heun` with fixed step size and `ruku45` and `rodas` with variable step size.
- `IntegratorMinStepSize`: Minimum step size for RODAS and the Runge-Kutta integrator.
- `IntegratorMaxSteps`: Maximum number of steps for RODAS and the Runge-Kutta integrator.
- `IntegratorRelTol`: Relative tolerance for RODAS and the Runge-Kutta integrator with variable step size. Note that this option may be insignificant if the minimum step size is chosen too high or the maximum number of steps is set too low.
- `IntegratorAbsTol`: Absolute tolerance for RODAS and the Runge-Kutta integrator with variable step size. Note that this option may be insignificant if the minimum step size is chosen too high or the maximum number of steps is set too low.
- `FlagsRodas`: Vector with the elements [`IFCN`, `IDFX`, `IJAC`, `IMAS`, `MLJAC`, `MUJAC`, `MLMAS`, `MUMAS`] that is passed to the integrator RODAS, see Section 4.2.2 for a description of the single entries.

- `LineSearchType`: This option selects either the adaptive line search strategy (value `adaptive`) or the explicit approach (value `explicit1` or `explicit2`).
- `LineSearchExpAutoFallback`: If this option is activated, the automatic fallback strategy is used in the case that the explicit formulas result in negative step sizes.
- `LineSearchMax`: This option sets the maximum value $\alpha_{\max}$ of the step size α.
- `LineSearchMin`: This option sets the minimum value $\alpha_{\min}$ of the step size α.
- `LineSearchInit`: Indicates the initial value $\alpha_{\text{init}} > 0$ for the step size α. If the adaptive line search is used, the sample point α_2 is set to$\alpha_2 = \alpha_{\text{init}}$.
- `LineSearchAdaptAbsTol`: This option sets the absolute tolerance ε_ϕ of the difference in costs at the interval bounds α_1 and α_2. If the difference in the (scaled) costs on these bounds falls below ε_ϕ, the adaption of the interval is stopped in order to avoid oscillations.
- `LineSearchAdaptFactor`: This option sets the adaptation factor $\kappa > 1$ in (4.14) that determines how much the line search interval can be adapted from one gradient iteration to the next.

- `LineSearchIntervalTol`: This option sets the interval tolerance $\varepsilon_\alpha \in (0, 0.5)$ in (4.14) that determines for which values of α the adaption is performed.
- `LineSearchIntervalFactor`: This option sets the interval factor $\beta \in (0, 1)$ that specifies the interval bounds $[\alpha_1, \alpha_3]$ according to $\alpha_1 = \alpha_2(1 - \beta)$ and $\alpha_3 = \alpha_2(1 + \beta)$, whereby the mid sample point is initialized as $\alpha_2 = \alpha_{\text{init}}$.

- `OptimControl`: Specifies whether the cost functional should be minimized with respect to the control variable $\boldsymbol{u}$.
- `OptimParam`: Specifies whether the cost functional should be minimized with respect to the optimization parameters $\boldsymbol{p}$.
- `OptimParamLineSearchFactor`: This option sets the adaptation factor $\gamma_{\boldsymbol{p}}$ that weights the update of the parameter vector $\boldsymbol{p}$ against the update of the control $\boldsymbol{u}$.
- `OptimTime`: Specifies whether the cost functional should be minimized with respect the horizon length T (free end time problem) or if T is kept constant.
- `OptimTimeLineSearchFactor`: This option sets the adaptation factor γ_T that weights the update of the end time T against the update of the control $\boldsymbol{u}$.
- `ScaleProblem`: Activates or deactivates scaling. Note that GRAMPC requires more computation time if scaling is active.
- `xScale`, `xOffset`: Scaling factors $\boldsymbol{x}_{\text{scale}}$ and offsets $\boldsymbol{x}_{\text{offset}}$ for each state variable.
- `uScale`, `uOffset`: Scaling factors $\boldsymbol{u}_{\text{scale}}$ and offsets $\boldsymbol{u}_{\text{offset}}$ for each control variable.
- `pScale`, `pOffset`: Scaling factors $\boldsymbol{p}_{\text{scale}}$ and offsets $\boldsymbol{p}_{\text{offset}}$ for each parameter.
- `TScale`, `TOffset`: Scaling factor $\boldsymbol{T}_{\text{scale}}$ and offset $\boldsymbol{T}_{\text{offset}}$ for the horizon length.
- `JScale`: Scaling factor J_{scale} for the cost functional.
- `cScale`: Scaling factors $\boldsymbol{c}_{\text{scale}}$ for each state constraint. The elements of the vector refer to the equality, inequality, terminal equality and terminal inequality constraints.

- `EqualityConstraints`: Equality constraints $\boldsymbol{g}(\boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{p}, t) = \boldsymbol{0}$ can be disabled by the option value `off`.
- `InequalityConstraints`: To disable inequality constraints $\boldsymbol{h}(\boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{p}, t) \leq \boldsymbol{0}$, set this option to `off`.
- `TerminalEqualityConstraints`: To disable terminal equality constraints $\boldsymbol{g}_T(\boldsymbol{x}(T), \boldsymbol{p}, T) = \boldsymbol{0}$, set this option to `off`.
- `TerminalInequalityConstraints`: To disable terminal inequality constraints $\boldsymbol{h}_T(\boldsymbol{x}(T), \boldsymbol{p}, T) \leq \boldsymbol{0}$, set this option to `off`.
- `ConstraintsHandling`: State constraints are handled either by means of the augmented Lagrangian approach (option value `auglag`) or as soft constraints by outer penalty functions (option value `extpen`).

- `ConstraintsAbsTol`: Thresholds $(\varepsilon_{\boldsymbol{g}}, \varepsilon_{\boldsymbol{h}}, \varepsilon_{\boldsymbol{g_T}}, \varepsilon_{\boldsymbol{h_T}}) \in \mathbb{R}^{N_c}$ for the equality, inequality, terminal equality, and terminal inequality constraints.

- `MultiplierMax`: Upper bound $\mu_{\max}$ and lower bound $-\mu_{\max}$ for the Lagrangian multpliers.
- `MultiplierDampingFactor`:Damping factor $\rho \in [0, 1)$ for the multiplier update.
- `PenaltyMax`: This option sets the upper bound $c_{\max}$ of the penalty parameters.
- `PenaltyMin`: This option sets the lower bound $c_{\min}$ of the penalty parameters.
- `PenaltyIncreaseFactor`: This option sets the factor β_{in} by which penalties are increased.
- `PenaltyDecreaseFactor`: This option sets the factor β_{de} by which penalties are decreased.
- `PenaltyIncreaseThreshold`: This option sets the factor γ_{in} that rates the progress in the constraints between the last two iterates.
- `AugLagUpdateGradientRelTol`: Threshold $\varepsilon_{\text{rel,u}}$ for the maximum relative gradient of the inner minimization problem.

- `ConvergenceCheck`: This option activates the convergence criterion. Otherwise, the inner and outer loops always perform the maximum number of iterations, see the options `MaxGradIter` and `MaxMultIter`.
- `ConvergenceGradientRelTol`: This option sets the threshold $\varepsilon_{\text{rel,c}}$ for the maximum relative gradient of the inner minimization problem that is used in the convergence criterion. Note that this threshold is different from the one that is used in the update of multipliers and penalties.

A.3 GRAMPC data types

The following lines list the data types of GRAMPC. Note that these data types define the structure variable `grampc` including the substructures `sol`, `param`, `opt`, `rws`, and `userparam`, also see Section 5.1.2.

GRAMPC **main structure:**

```
typedef struct
{
  typeGRAMPCparam *param;
  typeGRAMPCopt *opt;
  typeGRAMPCsol *sol;
  typeGRAMPCrws *rws;
  typeUSERPARAM *userparam;
} typeGRAMPC;
```

GRAMPC **parameter structure:**

```
typedef struct
{
  typeInt Nx;
  typeInt Nu;
  typeInt Np;
  typeInt Ng;
  typeInt Nh;
  typeInt NgT;
  typeInt NhT;
  typeInt Nc;

  typeRNum *x0;
  typeRNum *xdes;

  typeRNum *u0;
  typeRNum *udes;
  typeRNum *umax;
  typeRNum *umin;

  typeRNum *p0;
  typeRNum *pmax;
  typeRNum *pmin;

  typeRNum Thor;
  typeRNum Tmax;
  typeRNum Tmin;

  typeRNum dt;
  typeRNum t0;
} typeGRAMPCparam;
```

GRAMPC **option structure:**

```
typedef struct
{
  typeInt Nhor;
  typeInt MaxGradIter;
  typeInt MaxMultIter;
  typeInt ShiftControl;

  typeInt TimeDiscretization;

  typeInt IntegralCost;
  typeInt TerminalCost;
  typeInt IntegratorCost;

  typeInt  Integrator;
  typeRNum IntegratorRelTol;
  typeRNum IntegratorAbsTol;
  typeRNum IntegratorMinStepSize;
  typeInt  IntegratorMaxSteps;
  typeInt  *FlagsRodas;

  typeInt  LineSearchType;
  typeInt  LineSearchExpAutoFallback;
  typeRNum LineSearchMax;
  typeRNum LineSearchMin;
  typeRNum LineSearchInit;
  typeRNum LineSearchIntervalFactor;
  typeRNum LineSearchAdaptFactor;
  typeRNum LineSearchIntervalTol;

  typeInt  OptimControl;
  typeInt  OptimParam;
  typeRNum OptimParamLineSearchFactor;
  typeInt  OptimTime;
  typeRNum OptimTimeLineSearchFactor;
  typeInt  ScaleProblem;
  typeRNum *xScale;
  typeRNum *xOffset;
  typeRNum *uScale;
  typeRNum *uOffset;
  typeRNum *pScale;
  typeRNum *pOffset;
  typeRNum TScale;
  typeRNum TOffset;
  typeRNum JScale;
  typeRNum *cScale;

  typeInt  EqualityConstraints;
  typeInt  InequalityConstraints;
  typeInt  TerminalEqualityConstraints;
  typeInt  TerminalInequalityConstraints;
  typeInt  ConstraintsHandling;
  typeRNum *ConstraintsAbsTol;

  typeRNum MultiplierMax;
  typeRNum MultiplierDampingFactor;
  typeRNum PenaltyMax;
  typeRNum PenaltyMin;
  typeRNum PenaltyIncreaseFactor;
  typeRNum PenaltyDecreaseFactor;
  typeRNum PenaltyIncreaseThreshold;
  typeRNum AugLagUpdateGradientRelTol;

  typeInt  ConvergenceCheck;
  typeRNum ConvergenceGradientRelTol;

} typeGRAMPCopt;
```

GRAMPC **solution structure:**

```
typedef struct
{
  typeRNum *xnext;
  typeRNum *unext;
  typeRNum *pnext;
  typeRNum Tnext;
  typeRNum *J;
  typeRNum cfct;
  typeRNum pen;
  typeInt *iter;
  typeInt status;
} typeGRAMPCsol;
```

GRAMPC **real workspace structure:**

```
typedef struct
{
  typeRNum *t;
  typeRNum *tls;

  typeRNum *x;
  typeRNum *adj;
  typeRNum *dcdx;

  typeRNum *u;
  typeRNum *uls;
  typeRNum *uprev;
  typeRNum *gradu;
  typeRNum *graduprev;
  typeRNum *dcdu;

  typeRNum *p;
  typeRNum *pls;
  typeRNum *pprev;
  typeRNum *gradp;
  typeRNum *gradpprev;
  typeRNum *dcdp;

  typeRNum T;
  typeRNum Tprev;
  typeRNum gradT;
  typeRNum gradTprev;
  typeRNum dcdt;

  typeRNum *mult;
  typeRNum *pen;
  typeRNum *cfct;
  typeRNum *cfctprev;
  typeRNum *cfctAbsTol;

  typeRNum *lsAdapt;
  typeRNum *lsExplicit;
  typeRNum *rwsScale;
  typeInt  lrwsGeneral;
  typeRNum *rwsGeneral;

  typeInt  lworkRodas;
  typeInt  liworkRodas;
  typeRNum *rparRodas;
  typeInt  *iparRodas;
  typeRNum *workRodas;
  typeInt  *iworkRodas;
} typeGRAMPCrws;
```

A.4 GRAMPC **function interface**

The main C functions for the usage of GRAMPC are listed below.

File grampc_init**:**

```
void grampc_init(typeGRAMPC **grampc, typeUSERPARAM *userparam);
void grampc_free(typeGRAMPC **grampc);
```

File grampc_run**:**

```
void grampc_run(const typeGRAMPC *grampc);
```

File grampc_setparam**:**

```
void grampc_setparam_real(const typeGRAMPC *grampc, const typeChar *paramName,
                          ctypeRNum paramValue);

void grampc_setparam_real_vector(const typeGRAMPC *grampc,
                                 const typeChar *paramName, ctypeRNum *paramValue);

void grampc_printparam(const typeGRAMPC *grampc);
```

File grampc_setopt**:**

```
void grampc_setopt_real(const typeGRAMPC *grampc, const typeChar *optName,
                        ctypeRNum optValue);

void grampc_setopt_int(const typeGRAMPC *grampc, const typeChar *optName,
                       ctypeInt optValue);

void grampc_setopt_string(const typeGRAMPC *grampc, const typeChar *optName,
                          const typeChar *optValue);

void grampc_setopt_real_vector(const typeGRAMPC *grampc, const typeChar *optName,
                               ctypeRNum *optValue);

void grampc_setopt_int_vector(const typeGRAMPC *grampc, const typeChar *optName,
                              ctypeInt *optValue);

void grampc_printopt(const typeGRAMPC *grampc);
```

File grampc_mess**:**

```
/** estimates PenaltyMin on basis of the first MPC iteration **/
typeInt grampc_printstatus(ctypeInt status, ctypeInt level);
```

File grampc_util**:**

```
/* estimates PenaltyMin on basis of the first MPC iteration */
typeInt grampc_estim_penmin(typeGRAMPC *grampc, ctypeInt rungrampc);
```

File `probfct`**:**

```
/** Dimensions of the states (Nx), controls (Nu), parameters (Np), equalities (Ng),
    inequalities (Nh), terminal equalities (NgT), and terminal inequalities (NhT) **/
  void ocp_dim(typeInt *Nx, typeInt *Nu, typeInt *Np, typeInt *Ng,
               typeInt *Nh, typeInt *NgT, typeInt *NhT, typeUSERPARAM *userparam);


/** System model Mfct dx/dt = f(t,x,u,p,userparam) **/
  void ffct(typeRNum *out, ctypeRNum t, ctypeRNum *x, ctypeRNum *u,
            ctypeRNum *p, typeUSERPARAM *userparam);
  void dfdx_vec(typeRNum *out, ctypeRNum t, ctypeRNum *x, ctypeRNum *vec,
                ctypeRNum *u, ctypeRNum *p, typeUSERPARAM *userparam);
  void dfdu_vec(typeRNum *out, ctypeRNum t, ctypeRNum *x, ctypeRNum *vec,
                ctypeRNum *u, ctypeRNum *p, typeUSERPARAM *userparam);
  void dfdp_vec(typeRNum *out, ctypeRNum t, ctypeRNum *x, ctypeRNum *vec,
                ctypeRNum *u, ctypeRNum *p, typeUSERPARAM *userparam);


/** Integral cost l(t,x(t),u(t),p,xdes,udes,userparam) **/
  void lfct(typeRNum *out, ctypeRNum t, ctypeRNum *x, ctypeRNum *u, ctypeRNum *p,
            ctypeRNum *xdes, ctypeRNum *udes, typeUSERPARAM *userparam);
  void dldx(typeRNum *out, ctypeRNum t, ctypeRNum *x, ctypeRNum *u, ctypeRNum *p,
            ctypeRNum *xdes, ctypeRNum *udes, typeUSERPARAM *userparam);
  void dldu(typeRNum *out, ctypeRNum t, ctypeRNum *x, ctypeRNum *u, ctypeRNum *p,
            ctypeRNum *xdes, ctypeRNum *udes, typeUSERPARAM *userparam);
  void dldp(typeRNum *out, ctypeRNum t, ctypeRNum *x, ctypeRNum *u, ctypeRNum *p,
            ctypeRNum *xdes, ctypeRNum *udes, typeUSERPARAM *userparam);


/** Terminal cost V(T,x(T),p,xdes,userparam) **/
  void Vfct(typeRNum *out, ctypeRNum T, ctypeRNum *x, ctypeRNum *p,
            ctypeRNum *xdes, typeUSERPARAM *userparam);
  void dVdx(typeRNum *out, ctypeRNum T, ctypeRNum *x, ctypeRNum *p,
            ctypeRNum *xdes, typeUSERPARAM *userparam);
  void dVdp(typeRNum *out, ctypeRNum T, ctypeRNum *x, ctypeRNum *p,
            ctypeRNum *xdes, typeUSERPARAM *userparam);
  void dVdT(typeRNum *out, ctypeRNum T, ctypeRNum *x, ctypeRNum *p,
            ctypeRNum *xdes, typeUSERPARAM *userparam);


/** Equality constraints g(t,x(t),u(t),p,uperparam) = 0 **/
  void gfct(typeRNum *out, ctypeRNum t, ctypeRNum *x, ctypeRNum *u,
            ctypeRNum *p, typeUSERPARAM *userparam);
  void dgdx_vec(typeRNum *out, ctypeRNum t, ctypeRNum *x, ctypeRNum *u,
                ctypeRNum *p, ctypeRNum *vec, typeUSERPARAM *userparam);
  void dgdu_vec(typeRNum *out, ctypeRNum t, ctypeRNum *x, ctypeRNum *u,
                ctypeRNum *p, ctypeRNum *vec, typeUSERPARAM *userparam);
  void dgdp_vec(typeRNum *out, ctypeRNum t, ctypeRNum *x, ctypeRNum *u,
                ctypeRNum *p, ctypeRNum *vec, typeUSERPARAM *userparam);


/** Inequality constraints h(t,x(t),u(t),p,uperparam) <= 0 **/
  void hfct(typeRNum *out, ctypeRNum t, ctypeRNum *x, ctypeRNum *u,
            ctypeRNum *p, typeUSERPARAM *userparam);
  void dhdx_vec(typeRNum *out, ctypeRNum t, ctypeRNum *x, ctypeRNum *u,
                ctypeRNum *p, ctypeRNum *vec, typeUSERPARAM *userparam);
  void dhdu_vec(typeRNum *out, ctypeRNum t, ctypeRNum *x, ctypeRNum *u,
                ctypeRNum *p, ctypeRNum *vec, typeUSERPARAM *userparam);
  void dhdp_vec(typeRNum *out, ctypeRNum t, ctypeRNum *x, ctypeRNum *u,
                ctypeRNum *p, ctypeRNum *vec, typeUSERPARAM *userparam);
```

```
/** Terminal equality constraints gT(T,x(T),p,uperparam) = 0 **/
  void gTfct(typeRNum *out, ctypeRNum T, ctypeRNum *x, ctypeRNum *p,
             typeUSERPARAM *userparam);
  void dgTdx_vec(typeRNum *out, ctypeRNum T, ctypeRNum *x, ctypeRNum *p,
                 ctypeRNum *vec, typeUSERPARAM *userparam);
  void dgTdp_vec(typeRNum *out, ctypeRNum T, ctypeRNum *x, ctypeRNum *p,
                 ctypeRNum *vec, typeUSERPARAM *userparam);
  void dgTdT_vec(typeRNum *out, ctypeRNum T, ctypeRNum *x ctypeRNum *p,
                 ctypeRNum *vec, typeUSERPARAM *userparam);


/** Terminal inequality constraints hT(T,x(T),p,uperparam) <= 0 **/
  void hTfct(typeRNum *out, ctypeRNum T, ctypeRNum *x, ctypeRNum *p,
             typeUSERPARAM *userparam);
  void dhTdx_vec(typeRNum *out, ctypeRNum T, ctypeRNum *x, ctypeRNum *p,
                 ctypeRNum *vec, typeUSERPARAM *userparam);
  void dhTdp_vec(typeRNum *out, ctypeRNum T, ctypeRNum *x, ctypeRNum *p,
                 ctypeRNum *vec, typeUSERPARAM *userparam);
  void dhTdT_vec(typeRNum *out, ctypeRNum T, ctypeRNum *x, ctypeRNum *p,
                 ctypeRNum *vec, typeUSERPARAM *userparam);


/*rodas specific functions*/
  void dfdx(typeRNum *out, ctypeRNum t, ctypeRNum *x, ctypeRNum *u,
            ctypeRNum *p, typeUSERPARAM *userparam);
  void dfdxtrans(typeRNum *out, ctypeRNum t, ctypeRNum *x,
                 ctypeRNum *u, ctypeRNum *p, typeUSERPARAM *userparam);

  void dfdt(typeRNum *out, ctypeRNum t, ctypeRNum *x, ctypeRNum *u,
            ctypeRNum *p, typeUSERPARAM *userparam);
  void dHdxdt(typeRNum *out, ctypeRNum t, ctypeRNum *x, ctypeRNum *u,
              ctypeRNum *adj, ctypeRNum *p, typeUSERPARAM *userparam);

  void Mfct(typeRNum *out, typeUSERPARAM *userparam);
  void Mtrans(typeRNum *out, typeUSERPARAM *userparam);
```

Bibliography

[1] J. Barzilai and J. M. Borwein. Two-point step size gradient methods. *SIAM Journal on Numerical Analysis*, 8(1):141–148, 1988.

[2] A. E. Bryson and Y.-C. Ho. *Applied Optimal Control.* Blaisdell, New York, USA, 1969.

[3] A. R. Conn, G.I.M. Gould, and P. L. Toint. *LANCELOT: A Fortran Package for Large-Scale Nonlinear Optimization (Release A).* Springer-Verlag, Berlin, Germany, 1992.

[4] T. Englert and K. Graichen. Nonlinear model predictive torque control of PMSMs for high performance applications. *Control Engineering Practice*, 81:43 – 54, 2018.

[5] T. Englert, A. Völz, F. Mesmer, S. Rhein, and K. Graichen. A software framework for embedded nonlinear model predictive control using a gradient-based augmented Lagrangian approach (GRAMPC). *Optimization and Engineering*, 20(3):769–809, 2019. `doi.org/10.1007/s11081-018-9417-2`.

[6] K. Graichen and B. Käpernick. A real-time gradient method for nonlinear model predictive control. In T. Zheng, editor, *Frontiers of Model Predictive Control*, pages 9–28. InTech, 2012. http://www.intechopen.com/books/frontiers-of-model-predictive-control.

[7] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations: Stiff and Differential-Algebraic Problems.* Springer, Heidelberg, Germany, 1996.

[8] B. Käpernick and K. Graichen. Model predictive control of an overhead crane using constraint substitution. In *Proceedings of the American Control Conference (ACC)*, pages 3973–3978, 2013.

[9] B. Käpernick and K. Graichen. The gradient based nonlinear model predictive control software GRAMPC. In *Proceedings of the European Control Conference (ECC)*, pages 1170–1175, Strasbourg (France), 2014.

[10] J. Nocedal and S. Wright. *Numerical Optimization.* Springer Science & Business Media, New York, USA, 2006.

[11] S. Richter. *Computational Complexity Certification of Gradient Methods for Real-Time Model Predictive Control.* Ph.D. thesis ETH Zürich. ETH, 2012.

[12] RODAS. Webpage. http://www.unige.ch/~hairer/software.html, Accessed 01-December-2018.

[13] R. Rothfuss, J. Rudolph, and M. Zeitz. Flatness based control of a nonlinear chemical reactor model. *Automatica*, 32(10):1433–1439, 1996.