Aim of this exercise is to learn how to use the ACADO Integrator Suite to generate fast compiled ODE and DAE integrators that compute sensitivities and are callable from MATLAB [3]. Using these integrators, we implement a simple SCP method to solve a classical optimal control problem from the NMPC literature [2]. For this aim, we use the classical direct multiple shooting method from Bock and Plitt [1] in combination with Sequential Convex Programming [4].

1 Exercise Tasks

1. Install the ACADO Integrators Suite for MATLAB, following the instructions in Section 2.

2. Regard the dynamic system described by the ODE

\[ \dot{x} = f(x, u), \quad \text{with} \quad f(x, u) = \begin{bmatrix} x_2 + u(\mu + (1 - \mu)x_1) \\ x_1 + u(\mu - 4(1 - \mu)x_2) \end{bmatrix} \]

and assume \( \mu = 0.5 \). Generate an instance of the integrator that simulates the system for a time of \( \Delta t = 0.1 \) units and uses 3 integrator steps of the implicit Gauss-Legendre scheme of order 2. In mathematical notation, we will call this function \( \Phi(x_k, u_k) \). It generates the discrete time system \( x_{k+1} = \Phi(x_k, u_k) \). To help you with the syntax, here is the framework to do this in ACADO from MATLAB:

```matlab
DifferentialState x1 x2
Control u
mu = 0.5;
f = [ ...; ... ];
h = 0.1;
sim = acado.SIMexport( h );
sim.setModel(f);
sim.set( 'INTEGRATOR_TYPE', 'INT_IRK_GL2' );
% implicit Gauss-Legendre method of order 2
sim.set( 'NUM_INTEGRATOR_STEPS', 3 ); % 3 steps
sim.exportCode('export');
```

The resulting auto generated integrator can then simply be called like “states = integrate(x, u)”.

3. Call your generated function from MATLAB with arbitrary input values \( x_k, u_k \) and make yourself familiar with the outputs that the integrator generates. The result from the integrator is a struct,
with 3 fields: "value", "sensX" and "sensU". These are respectively the state values $\Phi(\bar{x}_k, \bar{u}_k)$, the sensitivities with respect to the states $\frac{\partial \Phi}{\partial x}(\bar{x}_k, \bar{u}_k)$, and the sensitivities with respect to the control inputs $\frac{\partial \Phi}{\partial u}(\bar{x}_k, \bar{u}_k)$.

4. Simulate the ODE starting with the value $x(0) = [-0.683, -0.864]^T$ and with the constant control input $u(t) = 0, t \in [0, T]$ over an interval of length $T = 1.5$. Plot the resulting state trajectories and compare them with this plot:

You can briefly check the accuracy of the integration method as follows:

```matlab
x = [-0.683; -0.864];
u = 0;
states = integrate(x, u);
xs = states.value;
options = odeset('RelTol', 1e-12, 'AbsTol', 1e-12);
[tout exact] = ode45(@(t, y) rhs(t, y, u), [0 h], x, options);
exact = exact(end, :);
abs(xs - exact)./abs(exact)
```

5. Now we want to solve the following optimal control problem from [2].

minimize $\int_0^T L(x(t), u(t)) dt + \frac{1}{2} x(T)^TPx(T)$
subject to $\dot{x}(t) = f(x(t), u(t)), \quad t \in [0, T],$
$-2 \leq u(t) \leq 2, \quad t \in [0, T],$
$x_0 = [-0.683; -0.864]^T,$
$x(T)^TPx(T) \leq \alpha.$
Here,
\[
L(x, u) = \frac{1}{2}(x^T Q x + u^T R u), \quad P = \begin{bmatrix} 16.5926 & 11.5926 \\ 11.5926 & 16.5926 \end{bmatrix}, \\
Q = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}, \quad R = \begin{bmatrix} 1.0 \end{bmatrix}, \quad \text{and} \quad \alpha = 0.7
\]

After the direct multiple shooting parameterization with \( N = 15 \) time steps of length \( \Delta t = 0.1 \), the resulting NLP has the form

\[
\begin{aligned}
\text{minimize} & \quad \Delta t \sum_{k=0}^{N-1} L(x_k, u_k) + \frac{1}{2} x_N^T P x_N \\
\text{subject to} & \quad x_{k+1} = \Phi(x_k, u_k), \quad k = 0, \ldots, N - 1, \\
& \quad -2 \leq u_k \leq 2, \quad k = 0, \ldots, N - 1, \\
& \quad x_0 = [-0.683; -0.864]^T, \\
& \quad x_N^T P x_N \leq \alpha.
\end{aligned}
\]

Note: the linearization of the nonlinear equality constraints at a trajectory \( \bar{x} = (\bar{x}_0, \ldots, \bar{x}_N) \) and \( \bar{u} = (\bar{u}_0, \ldots, \bar{u}_{N-1}) \) is given by:

\[
x_{k+1} = \Phi(\bar{x}_k, \bar{u}_k) + \frac{\partial \Phi}{\partial x_k}(\bar{x}_k, \bar{u}_k)(x_k - \bar{x}_k) + \frac{\partial \Phi}{\partial u_k}(\bar{x}_k, \bar{u}_k)(u_k - \bar{u}_k), \quad k = 0, \ldots, N - 1.
\]

In one step of the SCP loop, which of the variables in this equation will become YALMIP optimization variables, and which will be constants for YALMIP?

6. Now use the environment YALMIP to formulate and solve the convex subproblem that results from linearization at an arbitrary trajectory guess \( \bar{x} \) and \( \bar{u} \). Use no Levenberg-Marquardt regularization.

7. Write the full SCP algorithm, i.e. update the linearization point in each iteration with the most current solution guess \( \bar{x} \) and \( \bar{u} \). Decide on a suitable number of SCP iterations, e.g. 10. Plot the solution trajectory pieces in each iteration and make the infeasibilities \( \bar{x}_{k+1} - \Phi(\bar{x}_k, \bar{u}_k) \) between intervals visible. These infeasibilities are typical for direct multiple shooting, or simultaneous optimal control methods.

8. Produce a semilogarithmic plot of the norm of the infeasibility vector vs. the iteration number, and observe what is the convergence rate. Note that the total infeasibility is given by the residual vector

\[
\begin{bmatrix}
\bar{x}_1 - \Phi(\bar{x}_0, \bar{u}_0) \\
\vdots \\
\bar{x}_N - \Phi(\bar{x}_{N-1}, \bar{u}_{N-1})
\end{bmatrix}
\]

2 Appendix: ACADO

ACADO Toolkit \(^1\) is a software environment and algorithm collection for automatic control and dynamic optimization. It provides a general framework for using a great variety of algorithms for direct optimal control, including model predictive control, state and parameter estimation and robust optimization. ACADO Toolkit is implemented as self-contained C++ code and comes along with a user-friendly MATLAB interface.

We will explain how to download, compile and use ACADO Toolkit from MATLAB in Linux. ACADO can also be used from other operating systems like Windows and OS X. For the compact optimal control course, we however restrict ourselves to Linux systems.

\(^1\)https://sourceforge.net/projects/acado/
Check out ACADO from the SVN: Our suggestion is to always check out trunk for the latest bug fixes. Type the following command in a terminal from the directory in which you want to install ACADO. We will refer to this main ACADO folder as `<ACADO_ROOT>`.

```
svn checkout svn://svn.code.sf.net/p/acado/code/trunk ACADOtoolkit
```

Configuring MATLAB: Once a compiler is installed it needs to be linked to MATLAB. Open MATLAB (a recent version of MATLAB is required) and run in command window:

```
>> mex -setup;
```

MATLAB returns:

```
Please choose your compiler for building external interface (MEX) files:
Would you like mex to locate installed compilers [y]/n?
```

Type “y” and hit enter. MATLAB then shows you a list of installed compilers. Enter the number corresponding to the GCC compiler (in this case 1) and hit enter.

```
The options files available for mex are:

1: /software/matlab/2009b/bin/gccopts.sh : Template Options file for building gcc MEX-files
2: /software/matlab/2009b/bin/mexopts.sh : Template Options file for building MEX-files via the system ANSI compiler
0: Exit with no changes

Enter the number of the compiler (0−2):
```

Confirm the result by writing “y” and hitting enter.

Building the ACADO interface: Open Matlab in the `<ACADO_ROOT>` directory. Then navigate to the MATLAB installation directory by running:

```
cd interfaces/matlab/;
```

You are now ready to compile the ACADO interface. This compilation will take several minutes but needs to be done only once. Run "make" in your command window:

```
make clean all;
```

You will see:

```
Cleaning up all ACADO files...
Removing ACADO folders from Matlab path...
Clean completed.
Making ACADO...
```

and after a while when the compilation is finished:

```
ACADO successfully compiled.
Needed to compile 249 file(s).
```
If you need to restart Matlab, run this make file again
to set all paths or run savepath in your console to
save the current search path for future sessions.

ACADO has now been compiled. As the text indicated, every time you restart MATLAB you need
to run ”make” again to set all paths. When running ”make” again no new files need to be compiled
and the process will only take a few seconds. However, it is easier to save your path for future Matlab
sessions. Do so by running ”savepath” in your command window:

savepath;

Running your first example: We will now run a simple ”getting_started” example that will auto
generate an ACADO integrator with sensitivities. First go to the following examples folder:

cd examples/code_generation/simulation/

Now type ”getting_started” in the command window to run the example:

getting_started

The output in the command window should consist of only the following parts:

1. ACADO translates the Matlab script into ACADO C++ syntax and compiles it:

Writing c++ files ...
Compiling c++ files ...

2. the resulting file is run, which will call ACADO to auto generate the requested code in the folder
”getting_started_export”:

ACADO Toolkit for Matlab – Developed by David Ariens, 2009–2010
Support available at http://www.acadotoolkit.org/matlab
Information: Code generation successful

3. the auto generated code is then compiled into MEX files to be used from MATLAB:

COMPILATION OF MEX FILES...
mex COPTIMFLAGS='-DNDEBUG -O3' -output integrate.mexa64
  getting_started_export/integrate.c
  getting_started_export/integrator.c
mex COPTIMFLAGS='-DNDEBUG -O3' -output rhs.mexa64
  getting_started_export/rhs.c
  getting_started_export/integrator.c

4. the rest of the output reports some accuracy and timing results after comparing a simulation
with the auto generated ACADO integrator and one with the MATLAB integrator ’ode45’

You could also run the example in ”pendulum.m”, which should show the simulation of a pendulum
falling down.
References


