Technical Report
Advanced Practical in Optimal Control

Horea-Alexandru Cărămizaru
MSc. Scientific Computing
Heidelberg University
Heidelberg, Germany
horea.caramizaru@stud.uni-heidelberg.de

20/6/2021

Abstract

The main scope of this practical is to expose the interfaces of **IDAS/CVODES** integrators from the **SUNDIALS suite** [3] to MATLAB. To this end, the implementation provides the mean to define a Dynamical System, to integrate it over an interval of time and to compute the first and second order sensitivity.

Alongside with this report, a full implementation can be found here: SUNDIALS2Matlab

1 Introduction

The main task of this practical was to research 2 ways of integrating **SUNDIALS suite** as part of **MLI** project and provide the necessary backend calls.

The 2 possible libraries considered for this task were: CasADi [1] and AMICI [2] both of them providing an automatic way of exposing SUNDIALS c/c++ code integrators to dynamical languages like Python and Matlab.

The final decision of using CasADi was taken after multiple testing on each of them and was based on the fact that is a better integrated project with a comprehensive documentation and a wider community.

In section 2 we will start by defining the kind of problems one should be able to solve based on the API we are going to provide.

In section 3 we will continue by introducing CasADi's framework and the interface particularities of this library.

Section 4 will define the C/C++ code generation interface and the API for defining the dynamical system, computing the integration, jacobian of the ODE w.r.t. initial conditions and parameters of the ODE as well as the hessian wrt the parameters.

2 Problem Definition

The problem that needs to be solved regards the following OCP (Optimal Control problem):

$$
\begin{aligned}
\min_{x(\cdot),u(\cdot)} \quad & \Phi(x(t_f)) && \text{(1a)}\\
\text{s.t.} \quad & \dot{x}(t) = f(x(t), u(t), p), && \text{(1b)}\\
& x(t_0) = x_0 && \text{(1c)}\\
& x^{lo} \le x(t) \le x^{up}, && \text{(1d)}\\
& u^{lo} \le u(t) \le u^{up} \qquad \forall t \in [t_0, t_f] && \text{(1e)}
\end{aligned}
$$

To solve 1 numerically, we are using a discretized version by introducing the following multiple shooting variables: $s_0, \cdots, s_N$ $q_0, \cdots, q_N$

$$
\begin{aligned}
\min_{x(\cdot),u(\cdot)} \quad & \Phi(S_N) && && \text{(2a)}\\
\text{s.t.} \quad & s_{i+1} = x(t_{i+1}; t_i, s_i, q_i, p) && i = 0, ..., N-1 && \text{(2b)}\\
& s_0 = x_0 && && \text{(2c)}\\
& x^{lo} \le s_i \le x^{up}, && i = 0, ..., N && \text{(2d)}\\
& u^{lo} \le q_i \le u^{up} && i = 0, ..., N && \text{(2e)}
\end{aligned}
$$

where $x(t; t_0, s, q, p)$ is the solution of 3

$$
\begin{aligned}
\dot{x}(t) &= f(x(t), q, p) && \text{(3a)}\\
x(t_0) &= s && \text{(3b)}
\end{aligned}
$$

Next, we define the primal variables as $w = (s, q)$ and we introduce the following functions for equality and inequality constraints:

$$a(w) = \begin{bmatrix} x_0 - s_0 \\ x(t_1; t_0, s_0, q_0, p) - s_1 \\ \vdots \\ x(t_N; t_{N-1}, s_{N-1}, q_{N-1}, p) - s_N \end{bmatrix} \tag{4}$$

$$b(w) = \begin{bmatrix} x^{lo} - s \\ s - x^{up} \\ q^{lo} - q \\ q - q^{up} \end{bmatrix} \tag{5}$$

Based on 4 and 5 one can write the OCP in a more compact form:

$$\min_{w} \quad \Phi(w) \tag{6a}$$

$$\text{s.t.} \quad a(w) = 0 \tag{6b}$$

$$b(w) \leq 0 \tag{6c}$$

For 6 the Lagrange function and its derivatives at point (w, λ, μ) are defined as follows:

$$\mathcal{L}(w, \lambda, \mu) = \Phi(w) - \lambda^\top a(w) - \mu^\top b(w) \tag{7}$$

$$\nabla \mathcal{L}(w, \lambda, \mu) = \begin{bmatrix} \nabla_w \Phi(w) - \nabla_w a(w)\lambda - \nabla_w b(w)\mu \\ a(w) \\ b(w) \end{bmatrix} \tag{8}$$

$$\nabla^2 \mathcal{L}(w, \lambda, \mu) = \begin{bmatrix} \nabla_w^2 \Phi(w) - \nabla_w^2 a(w)\lambda & \nabla_w a(w) & \nabla_w b(w) \\ \nabla_w a(w)^\top & & \\ \nabla_w b(w)^\top & & \end{bmatrix} \tag{9}$$

We want to be able to solve:

$$\nabla \mathcal{L}(w, \lambda, \mu) = 0. \tag{10}$$

We apply Newton's method and we have to solve for (w_i, λ_i, μ_i) :

$$\nabla^2 \mathcal{L}(w_i, \lambda_i, \mu_i)\Delta w + \nabla \mathcal{L}(w_i, \lambda_i, \mu_i) = 0. \tag{11}$$

Equivalently, the following QP (Quadratic Programming) needs to be solved:

$$\begin{aligned}
\min_{\Delta w} \quad & \Delta w^\top \nabla_w^2 \mathcal{L}(w_i, \lambda_i, \mu_i)\Delta w + \nabla \Phi(w_i)\Delta w && \text{(12a)}\\
\text{s.t.} \quad & a(w_i) + \nabla a(w_i)\Delta w = 0 && \text{(12b)}\\
& b(w_i) + \nabla b(w_i)\Delta w \leq 0 && \text{(12c)}
\end{aligned}$$

To solve 12, the following terms, which include the evaluation of the dynamical system, must be evaluated: $a(w_i), \nabla_w a(w_i), \nabla_w^2 a(w) \cdot \lambda$

The process of evaluation of $\boldsymbol{a(w_i), \nabla_w a(w_i), \nabla_w^2 a(w) \cdot \lambda}$ requires the implementation of the following functions, as part of integration of **SUNDIALS**:

- $\boldsymbol{x(t; t, s, q, p)}$ Standard forward integration.
- $\boldsymbol{\nabla_w x(t; t, s, q, p) \cdot d}$ This is the directional derivative of $\boldsymbol{x(t; t, s, q, p)}$ in the direction $\boldsymbol{d}$. Multiple directions can be evaluated at the same time. The complete Jacobian can be computed by computing directional derivatives in all unit directions.
- $\boldsymbol{\nabla_w^2 x(t; t, s, q, p) \cdot \lambda}$ Hessian of $\boldsymbol{\lambda^\top \cdot x(t; t, s, q, p)}$ with respect $\boldsymbol{w}$.

Where the dimensions are:

$$\begin{aligned}
\boldsymbol{t} &\in \mathbb{R} && \text{(13a)}\\
\boldsymbol{x} &\in \mathbb{R}^{\boldsymbol{n_x}} && \text{(13b)}\\
\boldsymbol{q} &\in \mathbb{R}^{\boldsymbol{n_q}} && \text{(13c)}\\
\boldsymbol{p} &\in \mathbb{R}^{\boldsymbol{n_p}} && \text{(13d)}\\
& && \text{(13e)}
\end{aligned}$$

3 CasADi

CasADi is an open-source software tool for numerical optimization in general and optimal control (i.e. optimization involving differential equations) in particular. [1]

The main scope of CasADi is automatic differentiation. Besides that, it has also support for ODE/DAE integration and sensitivity analysis, nonlinear programming and interfaces to other numerical tools (**SUNDIALS suite**)

At the core of CasADi is a self-contained symbolic framework that allows the user to construct symbolic expressions using a MATLAB inspired everything-is-a-matrix syntax, i.e. vectors are treated as n-by-1 matrices and scalars as 1-by-1 matrices. Further on, the constructed symbolical expression is used by numerical means.

The **SX** data type is used to represent matrices whose elements consist of symbolic expressions made up by a sequence of unary and binary operations. Below are some examples of CasADi's API for defining symbolical expressions.

```
%Defining 2 symbolical variables 'a' and 'b':
a = SX.sym('a');
b = SX.sym('b');

%Computing the Jacobian of 'sin(a)' with respect to 'a'.
J = jacobian(sin(a),a);
%Computing the Hessian
H = hessian([a;b],[a;b]);

% Function with two scalar inputs, one output.
x = a^2+b^2;
f = Function('f',{a,b},{x});
x_res = f(2,3);

% Function with one vector input, one output.
x = a^2+b^2;
f = Function('f',{[a;b]},{x});
x_res = f([2;3]);
```

```
% Solving a QP.
y = a^2+b^2;
solver = qpsol('solver','qpoases',struct('x',[a;b],'f',y));
res = solver('x0',[0.1;0.2]);
full(res.x)

% Solving NLP.
y = a^2+b^2;
solver = nlpsol('solver','ipopt',struct('x',[a;b],'f',y));
res = solver('x0',[0.1;0.2]);
full(res.x)

% Defining an ODE
%    dot(a) = 1,
%    dot(b) = a^2+b^2 = y
%
y = a^2+b^2;
intg = integrator('intg','cvodes',struct('x',[a;b],'ode',[1;y]));
res = intg('x0',[0.1;0.2]);
full(res.xf)
```

A comprehensive documentation for CasADi can be accessed here: CasADi.

4 Implementation

4.1 General application flow

First, one must define a Dynamical System. This most be done in a separate file as part of the folder *DynamicalSystems* and it must follow the previously introduced CasADi's convention of **ODE/DAE** definition.

An example for Lotka-Volterra ODE would be:

```
import casadi.*

%define the states
x = SX.sym('x',2);

%define the parameters
```

```
a = SX.sym('a',1);
b = SX.sym('b',1);
c = SX.sym('c',1);
d = SX.sym('d',1);

%define the control
u = SX.sym('u',1);

%building the dynamical system
sys = struct;
%states
sys.x = x;
%parameters
sys.p = [u;a;b;c;d];
%defining the ODE/DAE
sys.ode = [
    a * x(1) - b * x(1) * x(2) - x(1) * u ;
    c * x(1) * x(2) - d * x(2) - x(2) * u
    ];
```

Based on the above definition of the dynamical system, the user needs to call one time, a function that generates on the fly the customized C/C++ code for the corresponding integrator, it compiles the newly generated code (**JIT compiler**) and returns a **Functor** that provides future access to it.

The corresponding call for *Lotka-Volterra* ODE would look like as follows:

```
InitODE('lotka_volterraCasADi', tStart, OneCallTimeStepSize);
```

The corresponding call dor DAE looks like as follows:

```
InitDAE('DinamicalSystem',tStart,OneCallTimeStepSize);
```

The parameters of **InitODE()/InitDAE()** are as follows:

- *lotka_volterraCasADi* : Name of the file that contain the dynamical system
- *tStart* : The start time (most of the time, it's **0**)
- *OneCallTimeStepSize* : Length of one call step done by the integrator. The current CasADi's API implementation is limited to fix time-step for **SUNDIALS suite** which are build from C/C++ code generation.

At this point, all the prerequisites for the future calls of *integrator*, *sensitivity* and *hessian* ($\boldsymbol{integrate(inp)}$, $\boldsymbol{integrateWSensitivies(inp)}$ and $\boldsymbol{integrateWSensitiviesAndHessian(inp)}$) w.r.t the the ODE are satisfied.

Each of the calls must contain as parameter an object that contains a subset of the variables defined below:

- $\boldsymbol{inp.N}$: Number of integration steps in each interval from multiple shooting.
- $\boldsymbol{inp.M}$: Number of intervals.
- $\boldsymbol{inp.sd}$: Initial value for differential states for each multiple shooting interval.
- $\boldsymbol{inp.q}$: A vector of controls used by each integrator call with size: $\boldsymbol{inp.N \cdot inp.M}$
- $\boldsymbol{inp.p}$: The values of the parameters in the same order it was defined previously in the dynamical system.
- $\boldsymbol{inp.nx}$: The size of state X.
- $\boldsymbol{inp.nq}$: The number of control parameters.
- $\boldsymbol{inp.np}$: The number of parameters.
- $\boldsymbol{inp.sensdirs}$: The sensitivity directions in form of matrix.
- $\boldsymbol{inp.lambda}$: The adjoint sensitivity direction
- $\boldsymbol{inp.threads}$: The number of threads for the thread pool used by the integrator

For complete examples please check the following files: $\boldsymbol{test\_integrate.m}$, $\boldsymbol{test\_integrateWSensitivies.m}$ and $\boldsymbol{test\_integrateWSensitiviesAndHessia.m}$

The computation process is vectorized and can also be done in parallel by defining the number of threads. For this, special attention must be given to the way the initialization process is handled.

The Figure 1 offers a better perspective of the computational process.

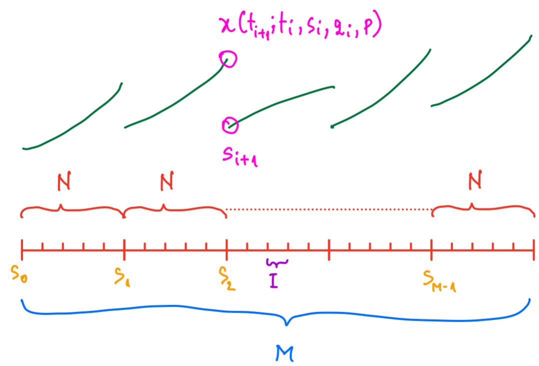

Figure 1: Multiple Shooting $\boldsymbol{I}$ - is a fix time-step integrator call, $\boldsymbol{M}$ - Number of intervals, $\boldsymbol{N}$ - Number of integration steps.

4.2 API

Dynamical system description

$$\dot{\boldsymbol{x}}(\boldsymbol{t}) = \boldsymbol{f}(\boldsymbol{t}, \boldsymbol{x}(\boldsymbol{t}), \boldsymbol{q}, \boldsymbol{p}) \tag{14a}$$

$$\boldsymbol{x}(\boldsymbol{t_0}) = \boldsymbol{x_0} \tag{14b}$$

Where: $\boldsymbol{t}$ is the time, $\boldsymbol{x}$ is the differential states, $\boldsymbol{q}$ is the control(constant), $\boldsymbol{p}$ is the parameter.
The calling Convention from Matlab for
$\boldsymbol{x(t; t, s, q, p)}$ $\boldsymbol{\nabla_w x(t; t, s, q, p) \cdot d}$ $\boldsymbol{\nabla_w^2 x(t; t, s, q, p) \cdot \lambda}$

The functions are called with on input argument **inp** and return one output argument **outp**. So all function calls have the header 'function outp = functionname(inp)'. Not all input attributes are always needed and not alle output attributes are always computed

4.3 Building the integrator

- `inp.thoriz` Integration horizon in the form of a $\mathbf{2 \times 1}$ matrix. At the beginning this can be assumed to be [0,1]. Later time transformation.
- `inp.sd` Initial value for differential states
- `inp.sa` Initial value for algebraic states $\rightarrow$ ignore atm

- `inp.q` Control parameter
- `inp.p` Model parameter
- `inp.sensdirs` Matrix of directional derivates of dimension $1 + n_x + n_q + n_p \times n_{sens}$, where n_{sens} is the number of derivates. Order of directions: [t, x, q, p]. t can be assumed to be 0.
- `inp.lambda` Adjoint directional derivatives. Matrix of dimension $n_x \times n_{adj}$, where n_{adj} is the number of adjoint derivates.

Output → see readme

5 Proposed Solutions

6 Criteria for Assessing Solutions

This may be a modified version of your proposal depending on previously carried out research or any feedback received.

7 Research Methodology

8 Analysis and Interpretation

9 Conclusions and Recommendations

Explanation for example in `howToComputeMultipleJacobiansAtOnce.m`.

The following function is an object, which represents the mathematical

$$x(x_0, q). \tag{15}$$

It maps $\mathbb{R}^2 \times \mathbb{R}^2 \mapsto \mathbb{R}^2$.

```
I = casadi.integrator('I','cvodes',ode_struct,opts);
```

The object

```
I_fwd = I.factory('I_fwd',{'x0','p','fwd:x0','fwd:p'},{'fwd:xf'})
```

represents the mathematical function

$$\boldsymbol{f}(\boldsymbol{x_0}, \boldsymbol{q_0}, \boldsymbol{d_{x_0}}, \boldsymbol{d_q}) = \boldsymbol{G_x}(\boldsymbol{x_0}, \boldsymbol{q})\boldsymbol{d_{x_0}} + \boldsymbol{G_p}(\boldsymbol{x_0}, \boldsymbol{q})\boldsymbol{d_q}. \tag{16}$$

It maps $\mathbb{R}^{\mathbf{2}} \times \mathbb{R}^{\mathbf{2}} \times \mathbb{R}^{\mathbf{2}} \times \mathbb{R}^{\mathbf{2}} \mapsto \mathbb{R}^{\mathbf{2}}$. If we want to compute the full Jacobian, we need to evaluate the function four times. For every direction once. Casadi allows to do this by one function call with

$$\boldsymbol{f}(\boldsymbol{x_0}, \boldsymbol{q_0}, \begin{bmatrix} \mathbf{1000} \\ \mathbf{0100} \end{bmatrix}, \begin{bmatrix} \mathbf{0010} \\ \mathbf{0001} \end{bmatrix}). \tag{17}$$

This is equivalent to

$$\boldsymbol{f}(\begin{bmatrix} \boldsymbol{x_0 x_0 x_0 x_0} \end{bmatrix}, \begin{bmatrix} \boldsymbol{q_0 q_0 q_0 q_0} \end{bmatrix}, \begin{bmatrix} \mathbf{1000} \\ \mathbf{0100} \end{bmatrix}, \begin{bmatrix} \mathbf{0010} \\ \mathbf{0001} \end{bmatrix}). \tag{18}$$

because internally $\boldsymbol{x_0}$ and $\boldsymbol{q_0}$ gets duplicated and the function $\boldsymbol{f}$ is evaluated column wise. In order to compute the Jacobian for multiple values, we need to duplicate the initial values and the directions as well. To evaluate the Jacobian for $\boldsymbol{x_0^1}, \boldsymbol{x_0^2 x_0^3}$ and $\boldsymbol{q_0^1}, \boldsymbol{q_0^2}, \boldsymbol{q_0^3}$ we have to do the function call

$$\begin{aligned} \boldsymbol{f}(& \begin{bmatrix} \boldsymbol{x_0^1 x_0^1 x_0^1 x_0^1 x_0^2 x_0^2 x_0^2 x_0^2 x_0^3 x_0^3 x_0^3 x_0^3} \end{bmatrix}, && (19) \\ & \begin{bmatrix} \boldsymbol{q_0^1 q_0^1 q_0^1 q_0^1 q_0^2 q_0^2 q_0^2 q_0^2 q_0^3 q_0^3 q_0^3 q_0^3} \end{bmatrix}, && (20) \\ & \begin{bmatrix} \mathbf{100010001000} \\ \mathbf{010001000100} \end{bmatrix}, && (21) \\ & \begin{bmatrix} \mathbf{001000100010} \\ \mathbf{000100010001} \end{bmatrix}). && (22) \end{aligned}$$

and split up the result in the individual Jacobians afterwards.

References

[1] Joel A E Andersson, Joris Gillis, Greg Horn, James B Rawlings, and Moritz Diehl. CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, In Press, 2018.

[2] Fabian Fröhlich, Daniel Weindl, Yannik Schälte, Dilan Pathirana, Łukasz Paszkowski, Glenn Terje Lines, Paul Stapor, and Jan Hasenauer. Amici: High-performance sensitivity analysis for large ordinary differential equation models. *Bioinformatics*, 04 2021. btab227.

[3] Alan C Hindmarsh, Peter N Brown, Keith E Grant, Steven L Lee, Radu Serban, Dan E Shumaker, and Carol S Woodward. SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software (TOMS)*, 31(3):363–396, 2005.