Introduction to CasADi

AWESCO Winter School on Numerical Optimal Control with Differential Algebraic Equations, Freiburg 2016

Joel Andersson,
joel@casadi.org

University of Wisconsin-Madison

15 February 2015

Outline

Methods for calculating derivatives

Derivatives play a central role in nonlinear optimization – how compute them?

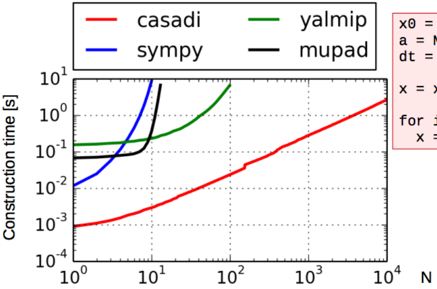
- By hand $\leftarrow$ **Time consuming & error prone!**
- Symbolic differentiation
- Finite differences
- Algorithmic differentiation (AD)

Symbolic differentiation

Obtain derivatives with a computer algebra system (CAS):

- Symbolic Toolbox for MATLAB / MuPAD
- SymPy
- ...

Easy to use but often results in a very long code which is expensive to evaluate.




```
x0 = MX.sym('x0')
a = MX.sym('a')
dt = 1

x = x0

for i in range(N):
  x = x+dt*(a*x+cos(x))
```

Finite differences

Consider a function $F : \mathbb{R}^{n_x} \to \mathbb{R}^{n_y}$ with Jacobian $J(x) = \dfrac{\partial F}{\partial x}$

$$J(x)\,\hat{x} \approx \frac{F(x + t\,\hat{x}) - F(x)}{t}$$

Pros and cons:

+ Easy to implement and relatively fast
− Poor accuracy, need to carefully choose t:
  - Small $t \Rightarrow$ *cancellation errors*
  - Large $t \Rightarrow$ *approximation errors*
− No efficient way to calculate $\hat{y}^\intercal\, J(x)$

Algorithmic differentiation (AD) (e.g. Griewank & Walther, 2008)

Decomposable function: $y = F(x)$

- $F : \mathbb{R}^{n_0} \rightarrow \mathbb{R}^{n_K}$ sufficiently smooth
- Decompose into "atomic operations" with known differentiation rules:

$z_0 \leftarrow x$
for $k = 1, \dots, K$ **do**
  $z_k \leftarrow f_k\left(\{z_i\}_{i \in \mathcal{I}_k}\right)$
end for
$y \leftarrow z_K$
return y

Such a decomposition is always available if F written as a computer program!

Example

$$y = \sin(\sqrt{x})$$

$z_0 \leftarrow x$
$z_1 = \sqrt{z_0}$
$z_2 = \sin z_1$
$y \leftarrow z_2$
return y

- Decomposition can be with simple scalar operations ...
  - $x + y$, $x * y$, $\sin(x)$, x^y
- ...or with higher-level operations for which a chain-rule can be defined
  - $x^\intercal$, $x[i] = y$, XY, e^X
  - E.g. gradient of $\det(X)$: $\det(X)\, X^{-\intercal}$
  - Linear and nonlinear systems of equations
  - Initial-value problems in ODE or DAE

Idea: Differentiate the algorithm!

$$
\begin{array}{l}
z_0 \leftarrow x \\
\textbf{for } k = 1, \ldots, K \textbf{ do} \\
\quad z_k \leftarrow f_k\left(\{z_i\}_{i \in \mathcal{I}_k}\right) \\
\textbf{end for} \\
y \leftarrow z_K \\
\textbf{return } y
\end{array}
\quad \Rightarrow \quad
\begin{array}{l}
z_0 \leftarrow x \\
\dfrac{dz_0}{dx} \leftarrow I \\
\textbf{for } k = 1, \ldots, K \textbf{ do} \\
\quad z_k \leftarrow f_k\left(\{z_i\}_{i \in \mathcal{I}_k}\right) \\
\quad \dfrac{dz_k}{dx} \leftarrow \displaystyle\sum_{i \in \mathcal{I}_k} \frac{\partial f_k}{\partial z_i}\left(\{z_i\}_{i \in \mathcal{I}_k}\right) \frac{dz_i}{dx} \\
\textbf{end for} \\
y \leftarrow z_K \\
J \leftarrow \dfrac{dz_K}{dx} \\
\textbf{return } y, J
\end{array}
$$

Write as a system of linear equations:

$$\frac{dz}{dx} = B + L\,\frac{dz}{dx}, \qquad J = A^{\intercal}\,\frac{dz}{dx},$$

Write as a system of linear equations:

$$\frac{dz}{dx} = B + L\,\frac{dz}{dx}, \qquad J = A^\intercal\,\frac{dz}{dx},$$

with

$$z = \begin{pmatrix} z_0 \\ z_1 \\ \vdots \\ z_K \end{pmatrix}, \quad A = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ I \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} I \\ 0 \\ \vdots \\ 0 \end{pmatrix},$$

with I and 0 of appropriate dimensions, as well as the *extended Jacobian*,

$$L = \begin{pmatrix} 0 & \ldots & \ldots & 0 \\ \frac{\partial f_1}{\partial z_0} & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ \frac{\partial f_K}{\partial z_0} & \ldots & \frac{\partial f_K}{\partial z_{K-1}} & 0 \end{pmatrix},$$

Since $I - L$ is invertible, we can solve for J:

$$J = A^\intercal\,(I - L)^{-1}\,B$$

- Have $J = A^\intercal (I - L)^{-1} B$
- Multiply J from the right: **Forward mode of AD**
  - $\hat{y} := J \hat{x} = A^\intercal (I - L)^{-1} B \hat{x}$
  - Cheap with *forward substitution* of lower triangular $(I - L)$
  - Computational cost: $\approx$ cost of evaluating F
  - Small memory requirements (no storage of L needed)
- Multiply J from the left: **Reverse mode of AD**
  - $\bar{x} := J^\intercal \bar{y} = B^\intercal (I - L)^{-\intercal} A \bar{y}$
  - Cheap with *backward substitution* of upper triangular $(I - L)^\intercal$
  - Computational cost: $\approx$ cost of evaluating F
  - If $F(x)$ is scalar, $\bar{y} = 1$ gives $\nabla_x F(x)$
  - Intermediate operations (or their linearization) must be stored
  - (Can trade storage for extra computation: "checkpointing")

Calculating complete Jacobians and Hessians

- Jacobians can be calculated by multiplying with n_{col} vectors from the right or n_{row} vectors from the left
- Worst-case: $\approx \min(n_{\text{row}}, n_{\text{col}})$ times cost of evaluating F
- *Much cheaper* if J is sparse, e.g. banded
- Hessians can be calculated as Jacobian-of-gradient
- Symmetry can be exploited

Exploiting sparsity: illustration

- $J = \begin{bmatrix} * & & & \\ & * & & \\ & & * & \\ & & & * \end{bmatrix} \quad \Rightarrow \hat{x} = [1, 1, 1, 1]$
- $J = \begin{bmatrix} * & & & \\ * & * & & \\ * & & * & \\ * & & & * \end{bmatrix} \quad \Rightarrow \hat{x}_1 = [0, 1, 1, 1], \hat{x}_2 = [1, 0, 0, 0]$
- $J = \begin{bmatrix} * & * & * & * \\ * & & & \\ * & & & \\ * & & & \end{bmatrix} \quad \Rightarrow \hat{x}_1 = [1, 0, 0, 0], \bar{y}_2 = [1, 0, 0, 0]$

Finding a small set of vectors

- NP hard combinatorial problem!
- Graph coloring techniques usually work well (cf. Gebremedhin et al, 2005)

Outline

CasADi

- Started as an implementation of AD using CAS-like syntax
- Current scope: Numerical optimization general
- In particular: Facilitates the solution of optimal control problems (OCPs)
  - *Facilitates*, not actually *solves* the OCPs
  - Write state-of-the-art OCP algorithms with very little code!
- Free & open-source (LGPL), also for commercial use
- Project started in December 2009, now (almost) at version 3.0
- Main developers: Joel Andersson and Joris Gillis

casadi.org → github.com

Search or type a command
Explore Gist Blog Help
jgillis

PUBLIC casadi / casadi

Pull Request | Unwatch | Unstar 4 | Fork 4

Code | Network | Pull Requests 0 | Issues 120 | Wiki | Graphs | Settings

Home | Pages | Wiki History | Git Access

Home

New Page | Edit Page | Page History

CasADi

Welcome to the CasADi wiki!

CasADi is a symbolic framework for automatic differentiation and numeric optimization. Using the syntax of computer algebra systems, it implements automatic differentiation in forward and adjoint modes by means of a hybrid symbolic/numeric approach. The main purpose of the tool is to be a low-level tool for quick, yet highly efficient implementation of algorithms for numerical optimization. Of particular interest is dynamic optimization, using either a collocation approach, or a shooting-based approach using embedded ODE/DAE-integrators. In either case, CasADi relieves the user from the work of efficiently calculating the relevant derivative or ODE/DAE sensitivity information to an arbitrary degree, as needed by the NLP solver. This together with full-featured Python and Octave front ends, as well as back ends to state-of-the-art codes such as Sundials (CVODES, IDAS and KINSOL), IPOPT and KNITRO, drastically reduces the effort of implementing the methods compared to a pure C/C++/Fortran approach.

Every feature of CasADi (with very few exceptions) is available in C++, Python and Octave, with little to no difference in performance, so the user has the possibility of working completely in C++, Python or Octave or mixing the languages. We recommend new users to try out the Python version first, since it allows interactivity and is more stable and better documented than the Octave front-end.

CasADi is an open-source tool, written in self-contained C++ code, depending only on the Standard Template Library. It is developed by Joel Andersson and Joris Gillis at the Optimization in Engineering Center, OPTEC of the K.U. Leuven under supervision of Moritz Diehl. CasADi is distributed under the LGPL license, meaning the code can be used royalty-free even in commercial applications.

Algorithmic differentiation (AD) in CasADi (Andersson, 2013)

- Decomposes algorithms into a sequence of either **scalar** or **sparse matrix-valued atomic operations**
- New symbolic expressions generated for derivatives: "Source code transformation" approach
  - Forward mode: Jacobian-times-vector products
  - Reverse mode: vector-times-Jacobian products
  - Sparse Jacobians and Hessians via:
    - Automatic detection of sparsity pattern (nontrivial!)
    - Graph coloring techniques to exploit sparsity & symmetry
  - Arbitrary order
- Supports **high-level operations**: matrix-operations, implicit functions, calls to DAE integrators

Outline

Quadratic programs (QPs)

- Exercise 1
- User needs to write the problem in the following standard form:

$$\boxed{\begin{array}{ll} \underset{x}{\text{minimize}} & f(x) \\ \text{subject to} & \underline{g} \le g(x) \le \overline{g} \\ & \underline{x} \le x \le \overline{x} \end{array}}$$

  where $f(x)$ is a convex quadratic function and $g(x)$ is a linear function.
- QP solvers available: qpOASES, OOQP, CPLEX, GUROBI
- Solver "plugins" can be added post-installation
- CasADi automatically generates matrix sparsities
- GUROBI & CPLEX: a subset of x can be integer-valued (mixed-integer QP)

Nonlinear programs (NLPs)

- Exercise 2 (part one)
- User needs to write the problem in the following standard form:

$$\boxed{\begin{array}{ll} \underset{x}{\text{minimize}} & f(x) \\ \text{subject to} & \underline{g} \leq g(x) \leq \overline{g} \\ & \underline{x} \leq x \leq \overline{x} \end{array}}$$

where $f(x)$ and $g(x)$ twice continuously differentiable functions

- NLP solvers available: IPOPT, SNOPT, KNITRO, WORHP, CasADi's own
- Solver "plugins" can be added post-installation
- CasADi automatically generates derivative information

Nonlinear rootfinding problems

- Exercise 2 (part two)
- Nonlinear system of equations:

$$g(z, x_1, x_2, \ldots, x_n) = 0$$

  which implicitly defines z as a function of x_1, ..., x_n according to the *implicit function theorem* (i.e. $\frac{\partial g}{\partial z}$ must be invertible).
- NLP solvers available: KINSOL, CasADi's own
- Solver "plugins" can be added post-installation
- CasADi automatically generates derivative information
- Differentiable object: Derivatives of the rootfinding solver calculated automatically

Integrators

- Solves initial-value problems in ordinary or differential-algebraic equations (ODE/DAE)
- Given a DAE with fixed initial values coupled to another (linear) DAE with fixed terminal value (both with quadratures):

$$
\left\{
\begin{array}{rcl}
\dot{x} & = & f_x(x, z, p, t) \\
0 & = & f_z(x, z, p, t) \\
\dot{q} & = & f_q(x, z, p, t) \\
 & & \\
-\dot{\tilde{x}} & = & \tilde{f}_x(\tilde{x}, \tilde{z}, \tilde{p}, x, z, p, t) \\
0 & = & \tilde{f}_z(\tilde{x}, \tilde{z}, \tilde{p}, x, z, p, t) \\
-\dot{\tilde{q}} & = & \tilde{f}_q(\tilde{x}, \tilde{z}, \tilde{p}, x, z, p, t)
\end{array}
\right.
\qquad
\begin{array}{c}
t \in [0, T] \\
\left\{
\begin{array}{rcl}
x(0) & = & x_0 \\
q(0) & = & 0 \\
 & & \\
\tilde{x}(T) & = & \tilde{x}_0 \\
\tilde{q}(T) & = & 0
\end{array}
\right.
\end{array}
$$

- An *integrator in CasADi* is a mapping from $\{x_0, p, \tilde{x}_0, \tilde{p}\}$ to $\{x(T), q(T), \tilde{x}(0), \tilde{q}(0)\}$
- Enables **automatic forward and adjoint sensitivity analysis**

Outline

Summary

- Algorithmic differentiation (AD)
  - Jacobian-times-vector products cheap using *forward mode AD*
  - Vector-times-Jacobian products cheap using *reverse mode AD*
  - Good heauristics exist for complete sparse Jacobians and Hessians
- CasADi
  - Open-source framework for numerical optimization
  - Central feature I: general-purpose implementation of AD
  - Central feature II: solve standard problems conveniently
    - QPs
    - NLPs
    - Rootfinding problems
    - Initial-value problems in ODE/DAE
  - Currently (CasADi 3.0), relatively mature. Exception: MATLAB

Outline