Lund, 8 December 2011

CasADi tutorial – Advanced concepts in CasADi

Joel Andersson   Johan Åkesson

Department of Electrical Engineering (ESAT-SCD) &
Optimization in Engineering Center (OPTEC)
Katholieke Universiteit Leuven

Outline

Recall – SX symbolics

On Tuesday:

- Created symbolic variables using the function: `ssym("name",n,m)`
  - Returns an n-by-m `SXMatrix` with symbolic scalar variables
  - `SXMatrix` is a general sparse matrix type
  - Elements of `SXMatrix` can also be expressions or constants
- We used `SXMatrix` expressions to define `SXFunction` functions
  - `f = SXFunction([x1,x2,x3],[y1,y2])`
  - `SXFunction` was part of a larger family of "functions"
- `SXFunction` can be evaluated (numerically and) *symbolically*
  - `[y1,y2] = f.eval([x1,x2,x3])`

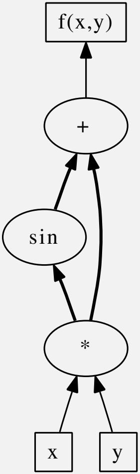
SX symbolics – what was 'cool'?

- *Look-and-feel* of CAS types . . . (cf. Maple, Symbolic Toolbox for Matlab)
- . . . but as "economic" and fast as AD-types (e.g. `adouble` in ADOL-C)
  - ≈ 5 times slower than optimized C code
- Generate complete Jacobians and Hessians
- Sparsity exploitation

Advanced symbolics, the `MX` class

`SX` symbolics – how it works

- Expressions are represented internally as *directed acyclic graphs* of simple unary & binary operations
  - E.g.: $+, -, *, \sin, \cos$, etc.
  - As in AD tools (ADOL-C, CppAD)
- When we evaluate a function symbolically (e.g. with `eval`), we copy all the new nodes to a new expression



Example: $f(x, y) = x\,y + \sin(x\,y)$

SX – limitations

- Expression graphs get very large
  - E.g.: If ODE/DAE requires 10^6 operations and we wish to evaluate it in $4 \cdot 20$ time points, NLP constraint function graph might contain some 10^8 nodes, it's Jacobian 10^{11} etc.
  - Conventional AD-tools tackle this problem with *checkpointing*
- Not all functions can be expanded in elementary operations
  - ODE/DAE integrator
  - External C function

`MX` symbolics

- CasADi's solution: a second, more general graph formulation
- Elementary operations in graph are: *multiple sparse matrix-valued input, multiple sparse matrix-valued output*
  - E.g. Function evaluation, matrix multiplication, ...
  - $\Rightarrow$ graph can contain many *calls* to the same function!

Automatic differentiation with `MX`

Uses chain rule for matrix operations

MX symbolics - syntax

- Create symbolic variables using the function: `msym("name",n,m)`
  - Returns an n-by-m MX symbolic expression
  - MX is a general sparse expression
  - (Almost) the same syntax as SXMatrix
- We can use MX expressions to define MXFunction functions
  - Syntax: `f = MXFunction([x1,x2,x3],[y1,y2])`
  - MXFunction uses the same syntax as SXFunction

Why not only use MX then?

Technically yes, but:

- Extra generality = slower speed
- Less features
- Currently not as stable

You cannot mix SX and MX graphs

But an MX graph can contain *calls* to an `SXFunction`

Advanced symbolics, the MX class

The user decides whether to work with SX or MX

- Idea:
  - SX for low-level operations – called often
  - MX for high-level operations – the "glue"
- Expanding MX $\Rightarrow$ SX
  - An MXFunction which does not contain calls to e.g. ODE integrators, can be automatically converted into an SXFunction.

Exercise, part 1:

- Working with `MX` symbolics
  - $\Rightarrow$ You need it for OCP

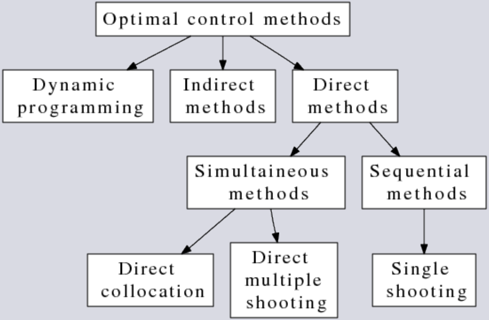
Outline

1. Advanced symbolics, the `MX` class
2. Optimal control with CasADi
3. What else is in CasADi?
4. Help & support
5. Exercise

Recall: Optimal control problem (OCP)

$$
\begin{aligned}
&\text{minimize} && \int_{t=0}^{T} L(x,u,p), dt + E(x(T),p) \\
&\text{subject to} && \\
& && \dot{x}(t) = f(x(t),u(t),p), && t \in [0,T] \\
& && x(0) = x_0(p) && \\
& && x_{\min} \le x(t) \le x_{\max}, && t \in [0,T] \\
& && u_{\min} \le u(t) \le u_{\max}, && t \in [0,T] \\
& && p_{\min} \le p \le p_{\max} &&
\end{aligned}
\tag{1}
$$

Optimal control methods



Optimal control with CasADi

- CasADi can be used efficiently for . . .
  - Direct single-shooting
  - Direct multiple-shooting
  - Direct collocation
  - Pseudospectral methods
  - Indirect methods

  Recommended work flow:
  - Use an existing OCP solver ($\Rightarrow$ JModelica.org) . . .
  - . . . or modify an example in CasADi's examples collection

Outline

What else is in CasADi?

Parallelization

- Normal (serial) function call (x1, x2, x3 $\in$ MX):
  ```
  [y1,y2] = f.call([x1,x2,x3])
  ```
- Parallel function call:
  ```
  [[y11,y12],[y21,y22]] = \
                 f.call([[x11,x12,x13],[x21,x22,x23]])
  ```
- Parallel function evaluation $\Rightarrow$ parallel Jacobian evaluation
- Uses OpenMP or (soon) MPI
- CasADi will ensure that evaluation is thread-safe

C code generation

- Generates very efficient C code
- Currently only for `SX`, `MX` possible extension
- The C code can be compiled and loaded *during execution*

Outline

Help & support

What do I do if . . .

- . . . discover a bug?
  ⇒ check the FAQ, isolate the issue, post a simple script to forum
- . . . my solution is very slow
  ⇒ check the speed-up tricks, isolate the issue, post to forum
- . . . IPOPT does not converge
  ⇒ read the IPOPT docs, mail *their* mailing list

More support

Happy to do joint projects

Other existing interfaces

- **ODE/DAE integrators**: CVODES/IDAS, ACADO Integrators, GSL
- **NLP solvers**: IPOPT, KNITRO, (SNOPT) ...
- **QP solvers**: qpOASES, OOQP, (CPLEX)
- **Newton-method**: KINSOL
- **Linear solvers**: LAPACK, CSparse, (SuperLU), ...

Adding more interfaces

(Usually) not much work

Help & support

Optimal control modelling framework

- Allows symbolic reformulation of OCP:s
  - Sort states & equations
  - Eliminating algebraic states (DAE $\Rightarrow$ ODE)
- Import models from Modelica via XML
  - FMI – standard format, supported by different tools
  - Currently uses the modified FMI format used in JModelica.org

Outline

Exercise

Exercise

- Get some experience using `MX` symbolics
- Implement direct single-shooting and direct multiple-shooting
  - Modify *Van-der-Pol oscillator* example
- If time permits, have a look at direct collocation example