Generation of symbolic Hessian matrices in OpenModelica

Sören Kai Möller Karim Kai Abdelhak Bernhard Bachmann

Bielefeld University of Applied Sciences

February 3, 2020

FH Bielefeld
University of
Applied Sciences

Outline

1. Motivation
2. Dynamic optimization in OpenModelica
3. Symbolic Hessian

Motivation

$$H_f(x) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_n} \end{pmatrix}$$

- Hessian matrices play a critical role for dynamic optimization problems
- performance of the optimizer heavily depends on the availability of derivative information
- whole symbolic machinery available in OpenModelica

Dynamic Optimization Problem

Dynamic Optimization Problem

$$\min_{u(t)} \; M(x(t_f), u(t_f), t_f) + \int_{t_0}^{t_f} L(x(t), u(t), t)dt$$

s.t.

$$x(t_0) = x_0 \tag{1}$$

$$\dot{x}(t) = f(x(t), u(t), t) \tag{2}$$

$$g(x(t), u(t), t) \leq 0 \tag{3}$$

$$r(x(t_f)) = 0 \tag{4}$$

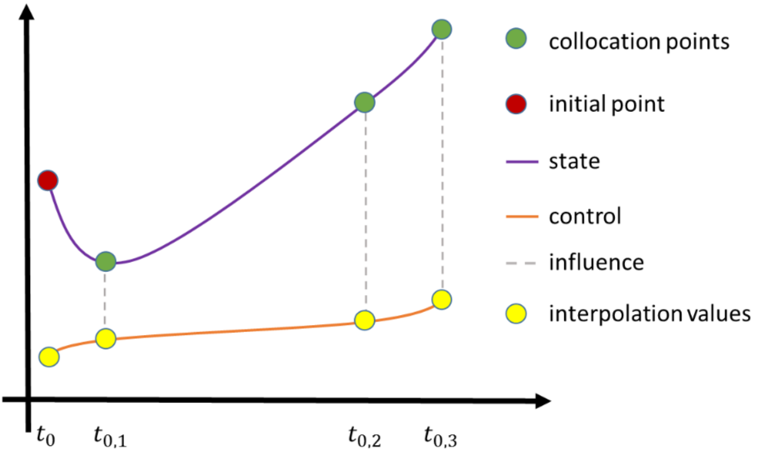
- Mayer term $M(\cdot)$
- Lagrange term $L(\cdot)$
- state vector $x(t)$
- control variable vector $u(t)$
- constraints (1), (2), (3) and (4)

Discretized problem formulation

- collocation methods are highly suitable for discretizing
- collocation with RADAU IIA and LOBATTO IIIA
- approximate Lagrange term with quadrature formulas
- discretized optimization problem can be solved

Discretized problem formulation

Closer look at collocation process:



Discretized problem formulation

- collocation methods are highly suitable for discretizing
- collocation with RADAU IIA and LOBATTO IIIA
- approximate Lagrange term with quadrature formulas
- discretized optimization problem can be solved

Finally the dynamic optimization problem can be discretized...

Discretized problem formulation

Discretized problem

$$\min \;\; M(x_{n,m}, u_{n,m}, t_{n,m}) + \Phi(\mathtt{x}, \mathtt{u}, \mathtt{t})$$

s.t.

$$\begin{aligned} c(\mathtt{x}, \mathtt{u}, \mathtt{s}, \mathtt{t}) &\stackrel{!}{=} 0 \\ \mathcal{U}_{max} \leq \mathtt{u} &\leq \mathcal{U}_{min} \\ \mathcal{X}_{max} \leq \mathtt{x} &\leq \mathcal{X}_{min} \\ 0 &\leq \mathtt{s} \end{aligned}$$

- $\mathtt{x} := [x_{0,1}, \ldots, x_{n,m}]$, $\mathtt{u} := [u_{0,1}, \ldots, u_{n,m}]$ and slack variables $\mathtt{s}$
  - $\Rightarrow$ Constraints: $c(\mathtt{x}, \mathtt{u}, \mathtt{s}, \mathtt{t})$
  - $\Rightarrow$ $\Phi(\mathtt{x}, \mathtt{u}, \mathtt{t}) \approx \int L(x(t), u(t), t)dt$

Nonlinear optimization

- transformed to nonlinear optimization problem
- optimizer need to find optimal discretized control vector
- requires first order derivatives from $M(\cdot)$, $\Phi(\cdot)$ und $\mathsf{c}(\cdot)$
- second order derivatives from the Lagrangian function

Lagrangian function

$$\mathcal{L}(z, \lambda, \mathrm{t}) = M(\cdot) + \Phi(\cdot) + \lambda^T \cdot \mathsf{c}(\cdot),$$
$$z = [\mathrm{x}, \mathrm{u}, \mathrm{s}]$$

Symbolic Hessian

- capabilities to differentiate symbolically a Modelica model
- generates symbolically partial derivatives
- new module *SymbolicHessian.mo*
- at the moment just for dynamic optimization implemented
- flag `--generateSymbolicHessian`

Generate Symbolic Hessian

Idea

Differentiate the system two times under usage of the Jacobian matrix!

1. differentiate objective function, ODE and constraints with respect to $x(t)$ and $u(t)$
2. multiply the Lagrange multipliers with the Jacobian matrix
3. differentiate resulting vector again under usage of Jacobian matrix

Generate Symbolic Hessian

Idea

Differentiate the system two times under usage of the Jacobian matrix!

1. differentiate objective function, ODE and constraints with respect to $x(t)$ and $u(t)$
2. multiply the Lagrange multipliers with the Jacobian matrix
3. differentiate resulting vector again under usage of Jacobian matrix

$$\Rightarrow \nabla^2 \mathcal{L}(\cdot)$$

Short example

Mathematical description of the well known Van der Pol oscillator.

Van der Pol oscillator

$$\min_{u(t)} \int_{t_0}^{t_f} x_1(t)^2 + x_2(t)^2 + u(t)^2 dt$$

s.t.

$$\begin{aligned}
\dot{x}_1(t) &= (1 - x_2(t)^2) \cdot x_1(t) - x_2(t) + u(t) \\
\dot{x}_2(t) &= x_1(t) \\
x_1(t_0) &= 0 \\
x_2(t_0) &= 1
\end{aligned}$$

Short example

Transform objective function in Mayer term.

Van der Pol oscillator

$$\min_{u(t)} \quad cost(t_f)$$

s.t.

$$\begin{aligned}
\dot{cost}(t) &= x_1(t)^2 + x_2(t)^2 + u(t)^2 \\
\dot{x}_1(t) &= (1 - x_2(t)^2) \cdot x_1(t) - x_2(t) + u(t) \\
\dot{x}_2(t) &= x_1(t) \\
x_1(t_0) &= 0 \\
x_2(t_0) &= 1
\end{aligned}$$

Short example

Collect the information.

Van der Pol oscillator

- objective function: $cost(t_f)$
- states: $cost$, x_1 and x_2
- input: u
- initial conditions: $x_1(t_0) = 0$ and $x_2(t_0) = 1$

Short example

Write it as an Modelica Model.

```
model VDP
      Real x1(start = 0, fixed = true);
      Real x2(start = 1, fixed = true);
      input u(max = 1, min = -0.5);
equation
      der(x1)= (1-x2^2)*x1-x2+u;
      der(x2)= x1;
end VDP;
optimization nmpcVDP(objective = cost)
      extends VDP;
      Real cost(start = 10, fixed = true);
equation
      der(cost)= x1^2+x2^2+u^2;
end nmpcVDP;
```

Short example

Well known Jacobian matrix calculates first order derivatives.

Van der Pol oscillator

$$
\begin{array}{cc}
 & \begin{array}{cccc} cost & x_1 & x_2 & u \end{array} \\
\begin{array}{c} \dot{cost} \\ \dot{x_1} \\ \dot{x_2} \end{array} &
\begin{pmatrix}
0 & 2x_1 & 2x_2 & 2u \\
0 & 1 - x_2^2 & -2x_2x_1 - 1 & 1 \\
0 & 1 & 0 & 0
\end{pmatrix}
\end{array}
$$

Short example

Use vector-matrix product, with $\lambda^T = (\lambda_1, \lambda_2, \lambda_3)$

Van der Pol oscillator

$$(\lambda_1, \lambda_2, \lambda_3) \cdot \begin{pmatrix} 0 & 2x_1 & 2x_2 & 2u \\ 0 & 1 - x_2^2 & -2x_2x_1 - 1 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix} =$$

$$\left(\; 0, \quad \lambda_1 2x_1 + \lambda_2(1 - x_2^2) + \lambda_3, \quad \lambda_1 2x_2 + \lambda_2(-2x_2x_1 - 1), \quad \lambda_1 2u + \lambda_2 \; \right)$$

Short example

Use vector-matrix product, with $\lambda^T = (\lambda_1, \lambda_2, \lambda_3)$

Van der Pol oscillator

$$(\lambda_1, \lambda_2, \lambda_3) \cdot \begin{pmatrix} 0 & 2x_1 & 2x_2 & 2u \\ 0 & 1 - x_2^2 & -2x_2x_1 - 1 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix} =$$

$$\left(\; 0, \quad \lambda_1 2x_1 + \lambda_2(1 - x_2^2) + \lambda_3, \quad \lambda_1 2x_2 + \lambda_2(-2x_2x_1 - 1), \quad \lambda_1 2u + \lambda_2 \; \right)$$

For the second order derivatives: Run the Jacobian module again!

Short example

Result: Hessian of the Lagrangian function, with respect to the states and the input

Van der Pol oscillator

$$
\begin{array}{c c}
 & \begin{array}{cccc} cost & x_1 & x_2 & u \end{array} \\
\begin{array}{c} cost \\ x_1 \\ x_2 \\ u \end{array} &
\begin{pmatrix}
0 & 0 & 0 & 0 \\
0 & \lambda_1 2 & \lambda_2(-2x_2) & 0 \\
0 & \lambda_2(-2x_2) & \lambda_1 2 + \lambda_2(-2x_1) & 0 \\
0 & 0 & 0 & \lambda_1 2
\end{pmatrix}
\end{array}
$$

Short example

Van der Pol oscillator

$$
\begin{array}{c c}
 & \begin{array}{cccc} cost & x_1 & x_2 & u \end{array} \\
\begin{array}{c} cost \\ x_1 \\ x_2 \\ u \end{array} &
\begin{pmatrix}
0 & 0 & 0 & 0 \\
0 & \lambda_1 2 & \lambda_2(-2x_2) & 0 \\
0 & \lambda_2(-2x_2) & \lambda_1 2 + \lambda_2(-2x_1) & 0 \\
0 & 0 & 0 & \lambda_1 2
\end{pmatrix}
\end{array}
$$

Representation of the symbolic Hessian in Modelica

```
1/1 (1): $HessianB =2.0*(x1.SeedB1*x1.SeedB+x2.SeedB1*
x2.SeedB+u.SeedB1*u.SeedB)*$lambda[1]+(-2.0)*(x2.SeedB*(x2*
x1.SeedB1+x2.SeedB1*x1)+x2*x2.SeedB1*x1.SeedB)*$lambda[2]
```

Outlook

- possible to generate Hessian matrices with symbolical differentiation techniques as Modelica expression
  - $\rightarrow$ at the moment it does not work with the discretized optimization problem
  - $\rightarrow$ goal: fix the issue and make the symbolic Hessian available for the optimizer
- analyze the influence of the initial guess of Newton-Raphson's algorithm
  - $\rightarrow$ used for the sensitivity of the solution after the first Newton-Raphson iteration

Thank you for your attention!

If you have any questions please feel free to ask