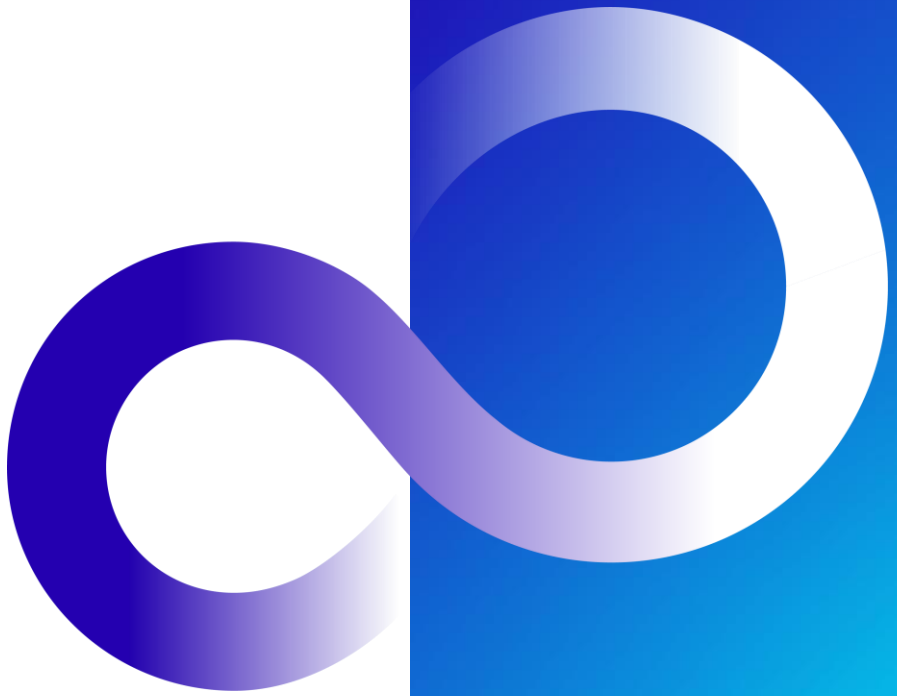
Differentiable Parameter Optimization for DAEs with State-Dependent Events

Ion Matei, Maksym Zhenirovskyy, Hon Yung Wong

Fujitsu Research of America

American Modelica & FMI Conference 2026, Atlanta, October 13, 2026



© 2026 Fujitsu Limited

Background and motivation

Differentiable simulation of Modelica models

© 2026 Fujitsu Limited

Differentiable solvers in ML frameworks

FUJITSU

Until 2026, differentiable solvers in the main ML frameworks targeted ODEs, while our Modelica models were often DAEs.

	TensorFlow TF 1.x	**PyTorch + torchdiffeq** 2018	**JAX + diffrax** 2018, 2021	**TorchDAE** June 2026
Capability	First differentiable ODE solvers in an ML framework	Dynamic graphs; backpropagation or adjoint through ODE solvers	JIT compilation (XLA), vmap, custom derivative rules	Index-1 DAEs, index reduction, adjoints, differentiable events
Our use	ODE models from Modelica; equations parsed from JModelica HTML output	Learning interaction laws of particle systems (2019)	Control design from Modelica DAEs via BLT form (2020); this work	Not evaluated in this work
Limitation	ODE only; unstable integration in our use; recompilation after each model change	ODE only; slow for larger optimization problems	ODE only; DAEs must first be converted to ODEs	Third-party library, not part of PyTorch core

ML: machine learning; ODE: ordinary differential equation; DAE: differential-algebraic equation; JIT: just-in-time; XLA: accelerated linear algebra compiler.
TorchDAE: github.com/yousef-rafat/torchdae, announced on the PyTorch forum on June 3, 2026. JAX: 2018 (Bradbury et al.); diffrax: 2021 (Kidger).

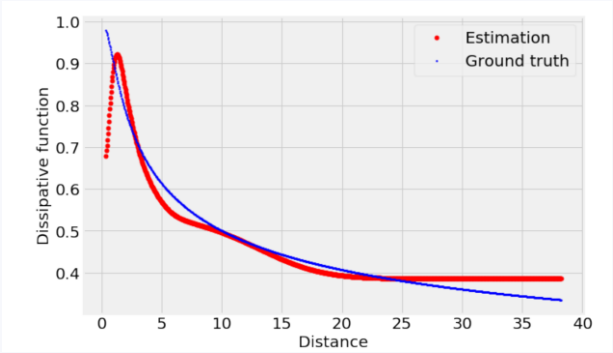
© 2026 Fujitsu Limited

Earlier workarounds for DAEs

FUJITSU

Design refinement cannot guarantee an ODE, so we need gradients through DAEs with events.

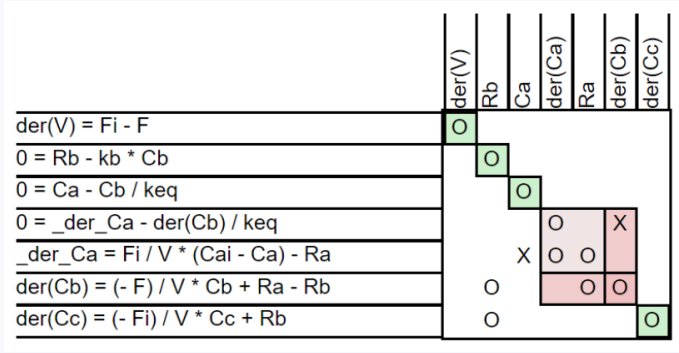
2019 Port-Hamiltonian particle models



- Swarm dynamics as networks of mass-spring-dampers
- Neural networks learn damping and potential functions
- PyTorch; an ODE integration in every training iteration

Lesson: The ODE size grows quadratically with the particle count.

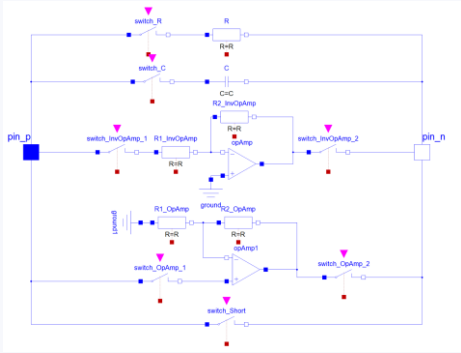
2020 BLT form with a learned algebraic loop



- BLT form parsed from JModelica HTML output
- A regression model replaces the algebraic loop
- The index-2 reactor DAE becomes a 3-state ODE (JAX)

Lesson: The learned causal model adds model uncertainty.

2024 Automated design with FMUs



- Designs are Modelica models refined by an optimizer
- Relaxed switches make the model a DAE
- Powell's method on co-simulation FMUs (PyFMI)

Lesson: No gradients: derivative-free search over up to 343 variables.

Sources: DARPA AIE Phase 1 report (2019); PGAI workshop at AAAI 2020; Modelica Conference 2024 proceedings (Matei et al.).
BLT: block lower triangular form; FMU: Functional Mock-up Unit; JModelica, PyFMI: Modelica compiler and FMU interface for Python.

© 2026 Fujitsu Limited

Two gradient methods for hybrid DAEs

AD through simulation and an explicit discrete adjoint

© 2026 Fujitsu Limited

Overview and contributions

One constrained least-squares problem, two ways to compute its gradient, compared on two benchmarks.

1 Parameter learning as constrained least squares
DAE segments, event guards, and reset maps enter as equality constraints

2 Method A: AD through simulation

- Newton iteration for z, differentiated by the implicit function theorem
- Events by segmented differentiable integration
- JAX and diffrax; one JIT-compiled value_and_grad call

3 Method B: explicit discrete adjoint

- An external solver (SUNDIALS IDA) provides the event-split trajectory
- Trapezoidal and event residuals are equality constraints
- Gradient from Lagrange multipliers (reverse sweep)

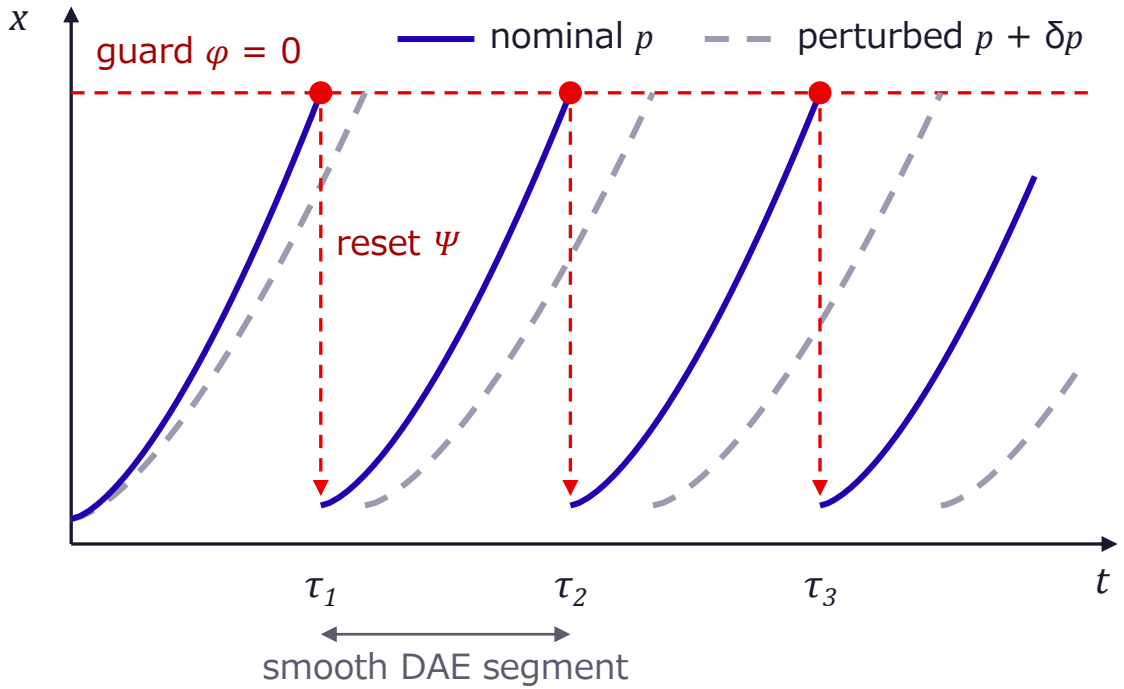
4 Comparison on a Cauer ladder DAE with a reinit event and on N bouncing balls (N = 3, 7, 15), with a PyTorch implementation of Method A

AD: automatic differentiation; DA: discrete adjoint; SUNDIALS IDA: implicit DAE solver of the SUNDIALS suite; N: number of balls.

© 2026 Fujitsu Limited

Hybrid DAE with state-dependent events

Three features make gradients difficult: implicit algebraic variables, parameter-dependent event times, and discontinuous resets.




```
when C3.v > 0.5 then
  reinit(C3.v, 0);
end when;
```

Modelica state event with reinitialization (Cauer benchmark)

On each segment $t \in (\tau_m, \tau_{m+1})$:

$$\dot{x} = f(t, x, z, p), \qquad 0 = g(t, x, z, p)$$

At an event time τ_m: guard and reset

$$\phi_{e_m}(\tau_m, x^-, z^-, p) = 0, \qquad (x^+, z^+) = \Psi_{e_m}(\tau_m, x^-, z^-, p)$$

Parameter estimation

$$\min_{p,\,x,\,z,\,\tau} \ \frac{1}{2} \sum_{i=1}^{N_{\text{obs}}} \left\| h\big(t_i, x(t_i), z(t_i), p\big) - y_i^{\text{data}} \right\|_2^2$$

$$\text{s.t.} \ \ \text{segment DAEs and } E_m(\tau_m, w_m^-, w_m^+, p) = 0, \quad m = 1, \ldots, M$$

x: differential states; z: algebraic variables; p: parameters; $w = (x, z)$; h: output map; E_m: guard, reset, continuity, and algebraic consistency at event m; $-/+$: left/right limits

1. Implicit z
2. τ_m depends on p
3. Jump at reset Ψ

© 2026 Fujitsu Limited

Method A: differentiation of the algebraic part

FUJITSU

Gradients pass through the algebraic constraint with one linear system, not through every Newton iteration.

$$z = \zeta(t, x, p) \quad \text{solves} \quad g\big(t, x, \zeta(t, x, p), p\big) = 0, \qquad \dot{x} = f\big(t, x, \zeta(t, x, p), p\big)$$



Unrolled AD (not used)

- Backpropagate through every Newton iteration
- Memory and time grow with the iteration count
- The gradient depends on the iteration history

Implicit function theorem at the converged z^* (used)

$$g_z\,dz + g_t\,dt + g_x\,dx + g_p\,dp = 0 \quad \Longrightarrow \quad dz = -\,g_z^{-1}\big(g_t\,dt + g_x\,dx + g_p\,dp\big)$$

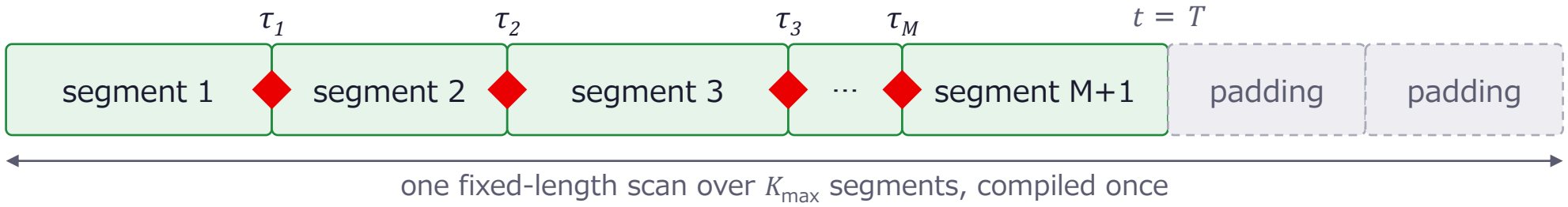
- g_t, g_x, g_z, g_p: partial derivatives of g at (t, x, z^*, p)
- A linear system in g_z is solved; g_z^{-1} is never formed
- Custom JVP (Jacobian-vector product) in JAX; reverse mode by transposition
- Requires g_z nonsingular (index-1 algebraic part)

© 2026 Fujitsu Limited

Method A: segmented event integration

FUJITSU

A fixed number of event-bounded segments makes the hybrid simulation one JIT-compiled, differentiable program.



At each segment boundary

1. Exclude guards already at zero ($\varphi_e \le \varepsilon_\varphi$) at the segment start
2. Locate the first zero of the composite guard Φ by root finding
3. Apply the reset map Ψ_e of the triggered event e
4. Re-solve z for post-event algebraic consistency

$$\Phi(t, x, p) = \min_{e \in \mathcal{A}} \phi_e\big(t, x, \zeta(t, x, p), p\big)$$

$$\hat{y}(\tau_m) = h\big(\tau_m,\, x(\tau_m^+),\, z(\tau_m^+),\, p\big)$$

$\mathcal{A}$: guards with $\varphi_e > \varepsilon_\varphi$ at the segment start ($\varphi_e > 0$ before its event, ≤ 0 after; ε_φ: small threshold); outputs at events are right-continuous

One JIT-compiled value_and_grad call differentiates through algebraic equations, integration, event times, resets, and output reconstruction.

Gradients are local to the event sequence of the forward simulation.

© 2026 Fujitsu Limited

Method A in JAX and in PyTorch

FUJITSU

The same AD-through-simulation method used 5, 16, and 37 times less total wall-clock time in JIT-compiled JAX than in PyTorch.

	JAX (ours)	PyTorch
Integrator	diffrax	torchdiffeq odeint
Event times	Root finding on the composite guard in a fixed-length scan	Hand-written root finding
Gradient	value_and_grad; implicit-function JVP for z	Backpropagation through the full tape (no odeint_adjoint)
Execution	JIT-compiled (XLA): traced once, run as one compiled program	Eager mode: each operation dispatched from Python

Total wall-clock time of PyTorch AD divided by that of JAX AD (bouncing balls)

5×	16×	37×
N = 3 balls	N = 7 balls	N = 15 balls

- Same model, targets, step size, and tolerance; the implementations differ in gradient computation and in the framework that holds the tape.
- At N = 15, PyTorch AD attains a lower loss (3.9×10^{-8} versus 1.4×10^{-5}) at 37 times the wall-clock time.

CPU runs; one run per setting; GPU execution and odeint_adjoint were not benchmarked. N: number of balls in Benchmark 2.

© 2026 Fujitsu Limited

Method B: discretize, then optimize

FUJITSU

An external DAE solver supplies the event-split trajectory; residuals turn it into a finite-dimensional constrained problem.

External DAE solver
SUNDIALS IDA with event detection

→

Event-split trajectory
segments; τ_m, active event, w^-, w^+

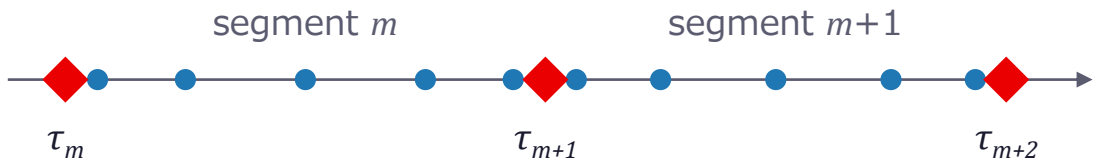
→

Residual system
$R_{m,k} = 0$ per step, $E_m = 0$ per event

→

Constrained problem
unknowns W, τ, p

Mesh of one segment between two events



$$t_{m,k} = \tau_m + \eta_{m,k}\,(\tau_{m+1} - \tau_m), \quad 0 = \eta_{m,0} < \cdots < \eta_{m,N_m-1} = 1$$

Event times move the mesh nodes, so the residuals depend on τ.

The residuals are equality constraints of the same estimation problem, not penalty terms.

Trapezoidal residual, step k of segment m

$$R_{m,k} = \begin{bmatrix} -x_{m,k+1} + x_{m,k} + \frac{h_{m,k}}{2}\left(f_{m,k} + f_{m,k+1}\right) \\ g_{m,k+1} \end{bmatrix} = 0$$

Finite-dimensional problem

$$\min_{W,\tau,p} \; \mathcal{J}_h(W,\tau,p) \quad \text{s.t.} \quad R_{m,k}(W,\tau,p) = 0, \quad E_m(W,\tau,p) = 0$$

$f_{m,k}$, $g_{m,k}$: f and g at node (m, k); $h_{m,k} = t_{m,k+1} - t_{m,k}$; W: values of x and z at all nodes; $\mathcal{J}_h$: discrete objective

w=(x,z), W=collection of w at all mesh nodes

© 2026 Fujitsu Limited

Method B: gradient from Lagrange multipliers

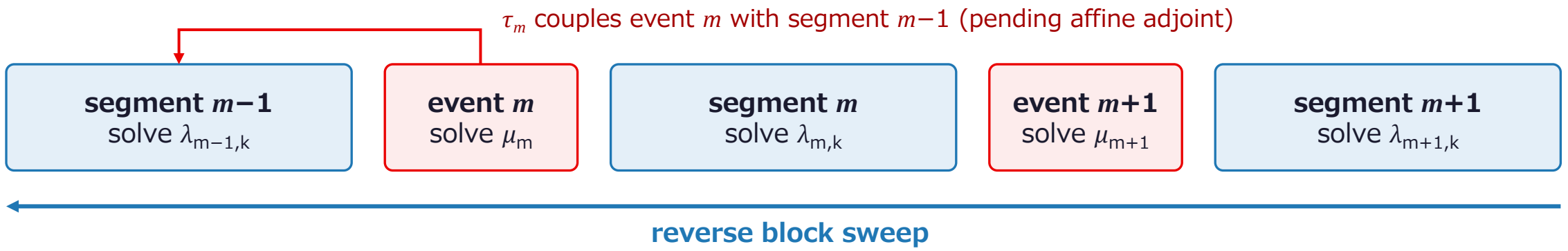
FUJITSU

Multipliers remove the need for trajectory and event-time sensitivities; one reverse block sweep gives the parameter gradient.

$$\mathcal{L}_h = \mathcal{J}_h + \sum_{m,k} \lambda_{m,k}^\top R_{m,k} + \sum_m \mu_m^\top E_m$$

$$\frac{\partial \mathcal{L}_h}{\partial W} = 0, \quad \frac{\partial \mathcal{L}_h}{\partial \tau} = 0 \implies \frac{d\mathcal{J}_h}{dp} = \frac{\partial \mathcal{L}_h}{\partial p}$$

On feasible trajectories $R = 0$ and $E = 0$, so $\mathcal{L}_h = \mathcal{J}_h$ for any multipliers λ, μ.



Advantages

- Uses a mature external DAE solver; no AD through solver internals
- Smaller linear systems when tearing is available

Cost

- Accuracy is set by the adjoint mesh, not by the forward solver tolerance
- An accurate forward simulation does not imply an accurate gradient

© 2026 Fujitsu Limited

Relation to hybrid optimal control

FUJITSU

Parameter estimation is optimal control with a constant control p; Method B is a discretized costate with jump conditions.

Continuous-time conditions (ODE, one event)

$$H = \ell(x) + \lambda^\top f(x,p), \qquad \dot\lambda = -\frac{\partial H}{\partial x}, \qquad \lambda(T) = 0$$

$$\lambda(\tau^-) = \Psi_x^\top \lambda(\tau^+) + \nu \nabla_x \phi$$

$$H(\tau^-) = H(\tau^+) \implies \nu = \frac{H(\tau^+) - \ell(x^-) - \lambda(\tau^+)^\top \Psi_x f^-}{\underbrace{\nabla_x \phi^\top f^-}_{\dot\phi(\tau^-)\neq 0}}$$

$$\frac{dJ}{dp} = \int_0^T \lambda^\top f_p \, dt + \lambda(\tau^+)^\top \Psi_p + \nu \, \phi_p$$

ℓ: running data-fit cost; λ: costate; ν: guard multiplier; $f^- = f(x(\tau^-), p)$; φ, Ψ: time-invariant guard and reset; subscripts denote partial derivatives

The denominator of c corresponds to $d\varphi/dt$ at τ^-; both methods assume it is nonzero (transversal crossing). For DAEs, the costate has an algebraic part.

Method B (appendix)	Hybrid optimal control
$\lambda_{m,k}$: step multipliers	costate $\lambda(t)$
$\mu = \mu_0 + c\,\nu$, with $A^\top\mu_0 = -a^+$ and $A^\top\nu = 0$	event multiplier: reset part plus guard part
$a^- = B^\top\mu_0 + c\,B^\top\nu$	$\lambda(\tau^-) = \Psi_x^\top\lambda(\tau^+) + \nu\nabla\varphi$
c: guard component of μ	guard multiplier ν
$q_\tau^E(c) + q_\tau^S(c) = 0$ fixes c	$H(\tau^-) = H(\tau^+)$ fixes ν
regularized denominator of c	$d\varphi/dt$ at τ^-; nonzero if transversal
reverse block sweep	backward costate equation

A, B: Jacobians of the event residual E in w^+, w^-; a^+, a^-: adjoint loads after and before the event; q_τ^E, q_τ^S: event and segment terms of the event-time derivative

Notation: appendix of arXiv:2605.05395. Interior-point constraints: Bryson and Ho (1975). Adjoint DAEs: Cao et al. (2003).

© 2026 Fujitsu Limited

FUJITSU

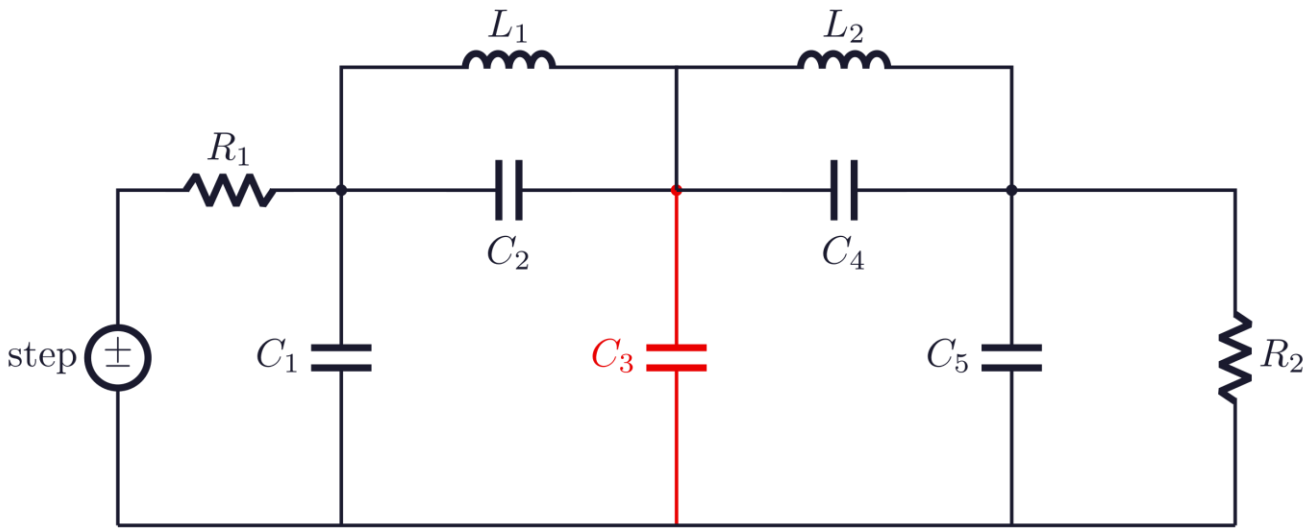
Experiments

A Cauer ladder DAE and N bouncing balls

© 2026 Fujitsu Limited

Benchmark 1: Cauer ladder with a reinit event

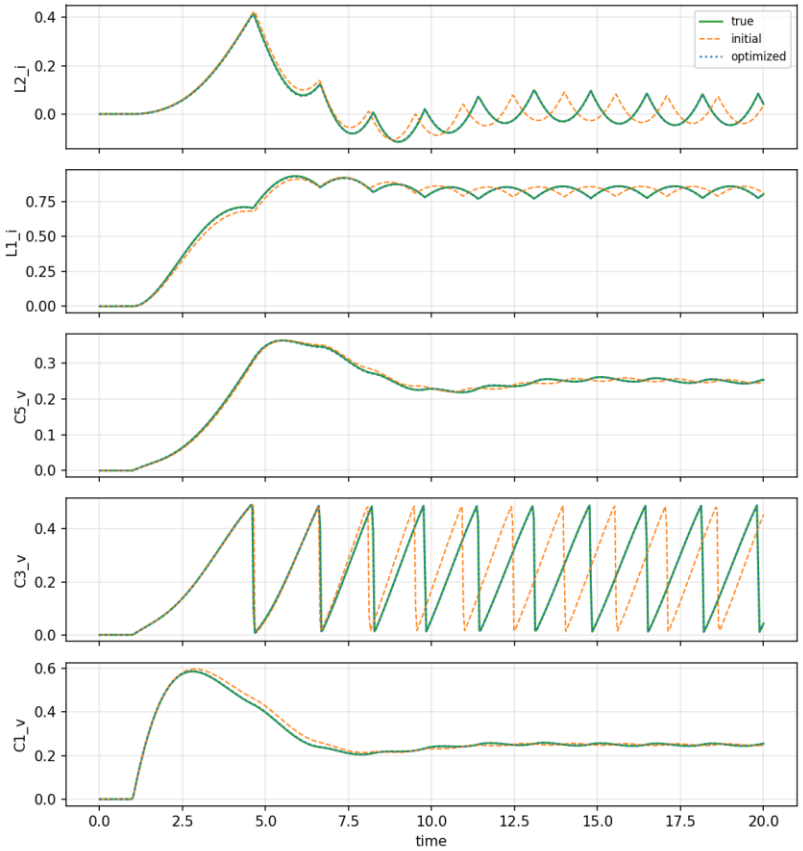
A stiff electrical DAE with 5 states, 3 algebraic variables, 7 parameters, and 10 state events in 20 s.




```
when C3.v > 0.5 then
  reinit(C3.v, 0);
end when;
```

Model: Modelica Standard Library CauerLowPassAnalog plus the event

- States: i_{L1}, i_{L2}, v_{C1}, v_{C3}, v_{C5}
- Algebraic: dv/dt of C_1, C_3, C_5
- Unknown: C_1 to C_5, L_1, L_2
- Start: ±10% bias; Adam, 500 iterations, 500 targets



Method A: true (solid), initial (dashed), optimized (dotted)

© 2026 Fujitsu Limited

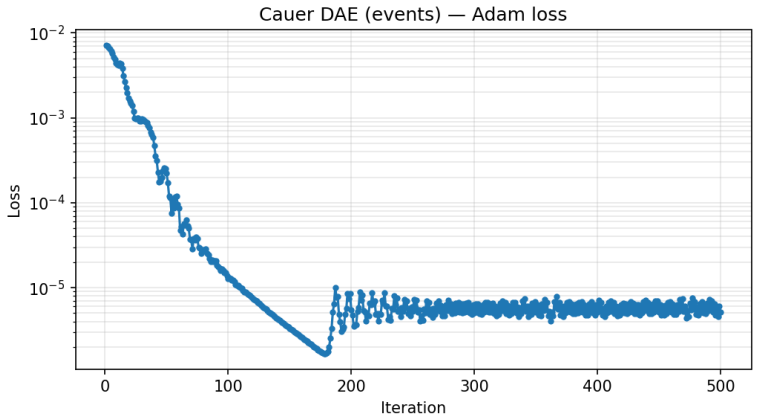
Cauer ladder: parameter recovery and loss

Method A attains a hard-selection loss of 1.3×10^{-6}, against 4.0×10^{-4} for Method B, at a similar time per iteration.

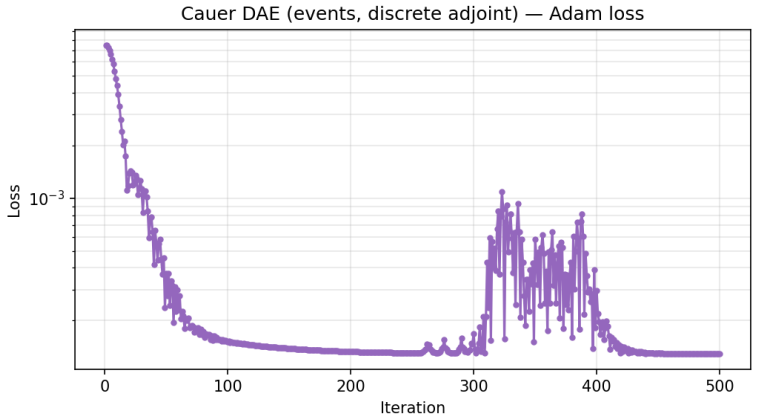
Method	C_1	C_2	C_3	C_4	C_5	L_1	L_2	Loss	Time/iter
Initial	2.78	4.50	8.77	9.22	6.47	8.61	2.16	7.59e-3	n/a
Method A (AD)	0.29	1.84	0.01	0.09	3.28	1.13	0.04	**1.32e-6**	0.030 s
Method B (DA)	2.97	1.80	0.02	0.03	5.44	0.53	1.09	4.01e-4	0.042 s

Cells: relative parameter error in % with respect to the true value. Loss: output loss with hard segment selection, re-evaluated by a common IDA simulation at the iterate with the smallest blended training loss. One run, seed 0.

Method A (AD): training loss



Method B (DA): training loss

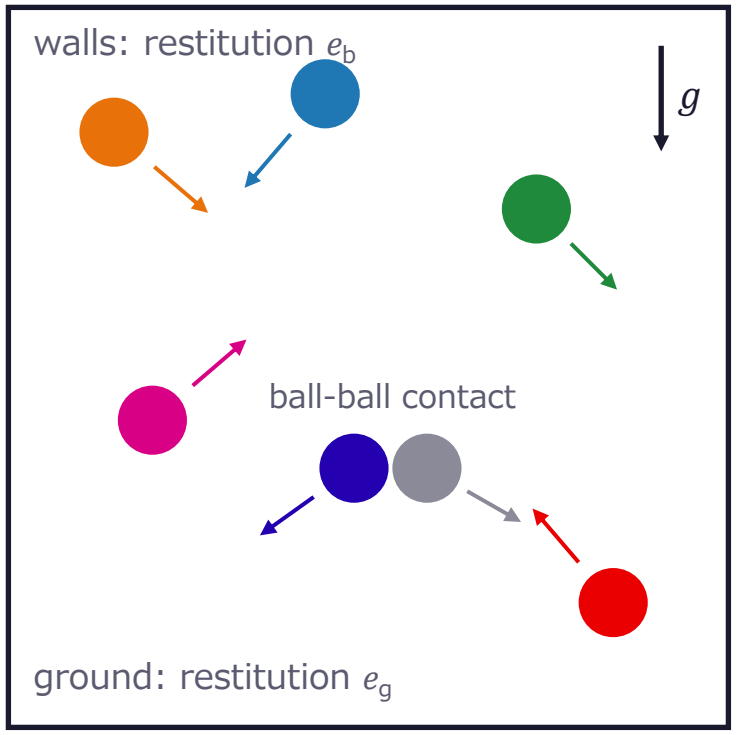


- Method A recovers four of seven parameters to below 0.3% error.
- Method B gradient accuracy is bounded by its trapezoidal adjoint mesh.
- C_5 is weakly identifiable: different parameters give nearly equal outputs.

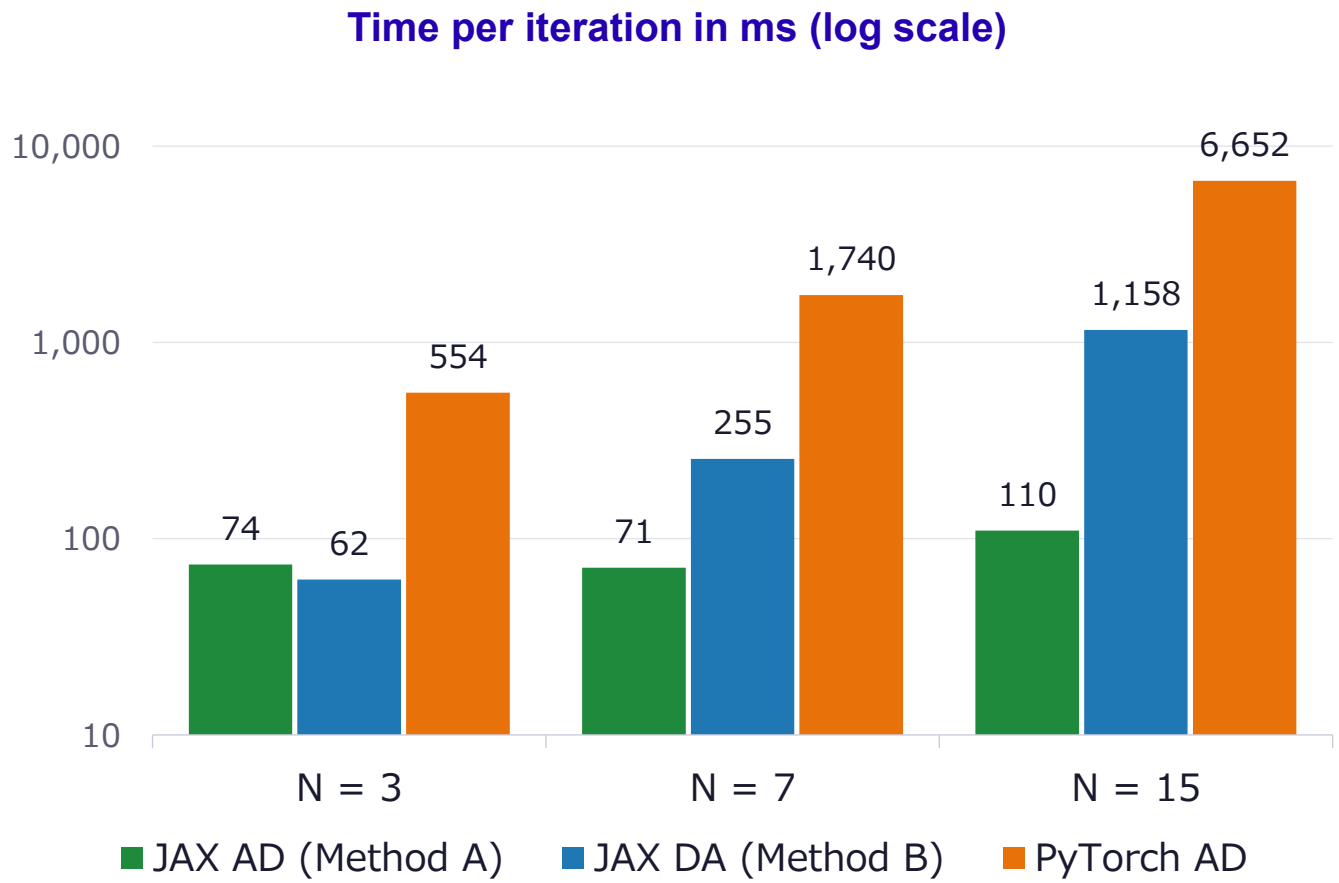
© 2026 Fujitsu Limited

Benchmark 2: N bouncing balls

An ODE with many impact events: $4N + N(N-1)/2$ event clauses, that is 15, 49, and 165 for N = 3, 7, and 15.



- Unknown $\theta = (g, e_g, e_b)$; start: ±10% bias
- Adam; 500 target points; CPU runs

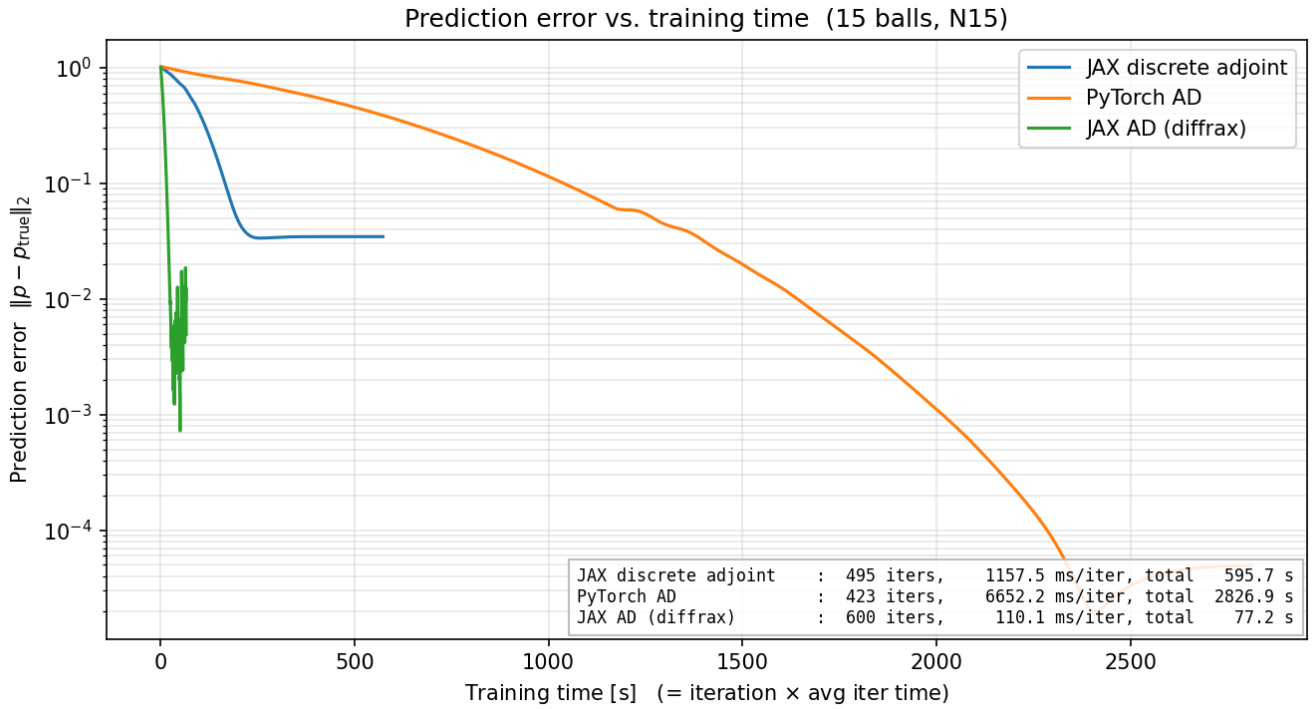


© 2026 Fujitsu Limited

Bouncing balls: accuracy and wall-clock time

FUJITSU

JAX AD keeps the time per iteration nearly constant; PyTorch AD attains a lower loss at $N = 15$ at 37 times the wall-clock time.



$N = 15$: parameter error $\| p - p_{\text{true}} \|_2$ versus training time (iterations × mean time per iteration)

$N = 15$	Validation loss	Total time
JAX AD	1.35e-5	77 s
JAX DA	9.28e-4	596 s
PyTorch AD	3.88e-8	2,827 s

1. Time per iteration from $N = 3$ to 15: JAX AD 74 to 110 ms; PyTorch AD 554 to 6,652 ms.

2. At $N = 3$ and 7, JAX AD attains the lowest loss (1.4×10^{-9} and 1.6×10^{-9}).

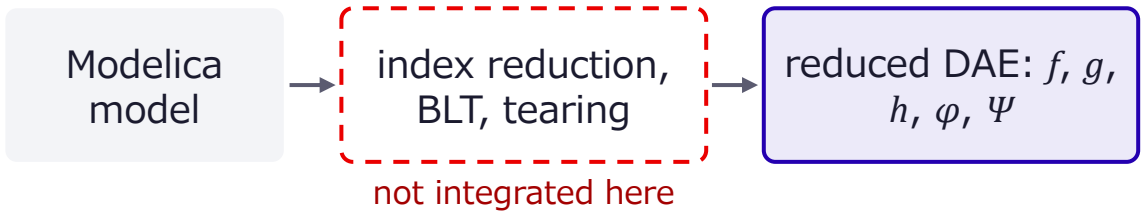
3. At $N = 15$, Method B shows a bias of about 4% on e_b from its fixed trapezoidal mesh.

© 2026 Fujitsu Limited

Limitations and open problems

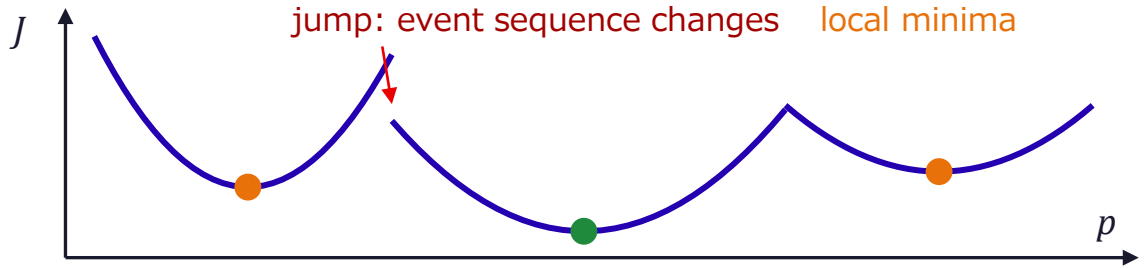
Both methods give local gradients for a fixed event sequence, and the reduced DAE is not yet generated by the Modelica toolchain.

1 Generation of the reduced DAE



- We supply the semi-explicit index-1 DAE: f, g, h, guards, and resets.
- Our algebraic iteration treats all of z jointly; tearing would reduce both the forward and the implicit linear systems.
- Integration paths: compiler-exported residuals and Jacobians, directional and adjoint derivatives of FMI 3.0 (Functional Mock-up Interface), symbolic adjoints.

2 Optimization through events



- Gradients hold only for a fixed, transversal event sequence.
- Grazing, simultaneous, or reordered events violate the assumptions.
- The loss is nonconvex; the result depends on the initialization.
- Output blending widens the basin and may bias the optimum.

© 2026 Fujitsu Limited

Summary and next steps

1. Gradients through DAEs with state events can be computed with two methods: AD through a JIT-compiled simulation, and an explicit discrete adjoint around an external solver.

2. Implicit differentiation of the Newton iteration and segmented integration make Method A practical; it attained the lowest loss on the Cauer DAE.

3. Method B reuses mature DAE solvers and corresponds to hybrid optimal control; its accuracy depends on the adjoint mesh.

Next steps

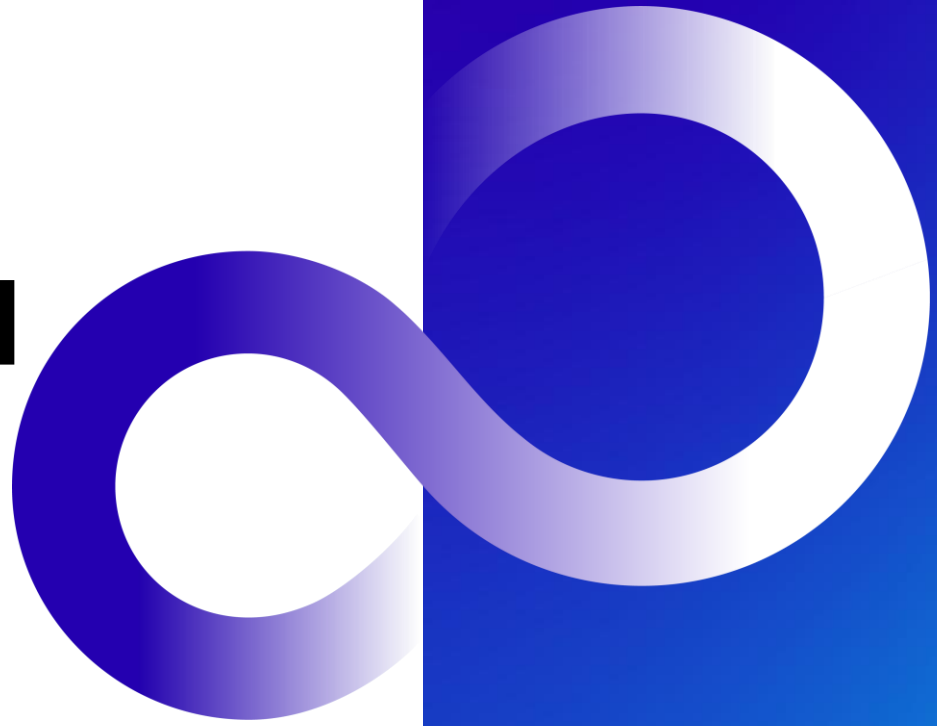
Reduced-DAE generation from Modelica and FMI; adaptive adjoint mesh; larger DAEs; multi-seed studies; comparison with IDAS adjoints and TorchDAE.

© 2026 Fujitsu Limited

Thank you

Questions

imatei@fujitsu.com



© 2026 Fujitsu Limited

Backup slides

Technical details for questions

© 2026 Fujitsu Limited

Backup: derivative rules for the algebraic part

FUJITSU

Forward: chord Newton with a frozen Jacobian $\bar{G} \approx g_z$

$$z^{k+1} = z^k - \bar{G}^{-1}\, g\big(t, x, z^k, p\big), \qquad k = 0, 1, \ldots$$

Tangent (JVP) rule at the converged point z^*

$$dz = -\,g_z^{-1}\big(g_t\, dt + g_x\, dx + g_p\, dp\big)$$

Adjoint rule (vector-Jacobian product), obtained by transposition

$$g_z^\top v = \bar{z}, \qquad \bar{t} \mathrel{+}= -g_t^\top v, \quad \bar{x} \mathrel{+}= -g_x^\top v, \quad \bar{p} \mathrel{+}= -g_p^\top v$$

$\bar{z}$, $\bar{x}$, $\bar{p}$, $\bar{t}$: cotangents (adjoint variables) of z, x, p, t; g_t, g_x, g_z, g_p: partial derivatives of g

- The derivative uses g_z at the converged z^*, not the frozen $\bar{G}$ of the chord iterations.
- Cost per evaluation: one linear system with g_z or its transpose, independent of the number of Newton iterations.
- The linear system is solved; g_z^{-1} is never formed.
- Assumes g_z nonsingular at the evaluated point (index-1 algebraic part).

© 2026 Fujitsu Limited

Backup: differentiable target-time blending

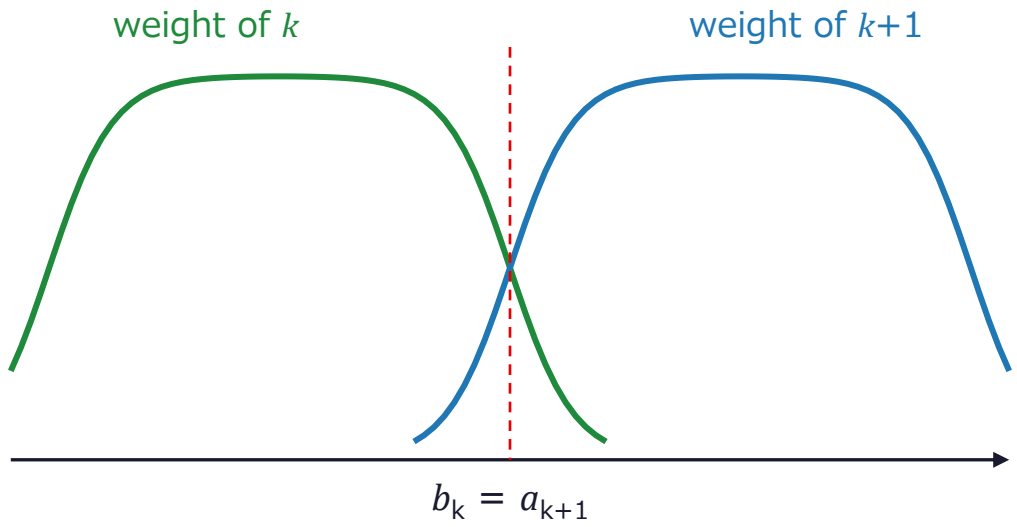
Only the output reconstruction is blended; the dynamics, guards, and resets are not smoothed.

$$\omega_{k,i} = \sigma\big(\beta(t_i - a_k)\big)\,\sigma\big(\beta(b_k - t_i)\big)\,r_k$$

$$\bar{\omega}_{k,i} = \frac{\omega_{k,i}}{\sum_\ell \omega_{\ell,i} + \epsilon_\omega}, \qquad \hat{x}_\beta(t_i;p) = \sum_k \bar{\omega}_{k,i} X_{k,i}$$

$$\hat{z}_\beta(t_i;p) = \zeta\big(t_i, \hat{x}_\beta(t_i;p), p\big), \qquad \hat{y}_\beta = h\big(t_i, \hat{x}_\beta, \hat{z}_\beta, p\big)$$

a_k, b_k: bounds of segment k; $r_k \in \{0, 1\}$: physical or padded segment; $X_{k,i}$: segment k evaluated at t_i clipped to $[a_k, b_k]$; $\sigma(s) = 1/(1 + e^{-s})$; $\beta > 0$: sharpness; $\varepsilon_\omega > 0$



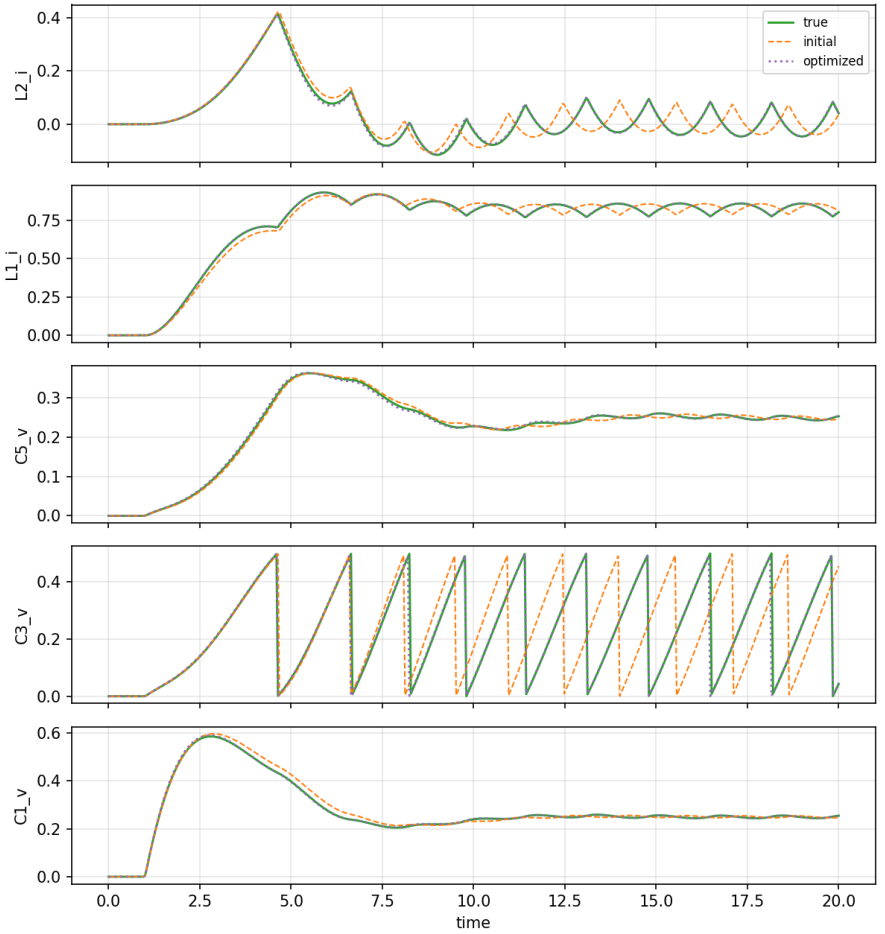
- β = 150 in the Cauer runs; reported losses use hard selection.
- Hard selection is recovered away from events as $\beta \to \infty$.
- Finite β may bias the optimum; the bias was not quantified.

© 2026 Fujitsu Limited

Backup: Cauer event residual

$$E_C(\tau_C, w^-, w^+, p) = \begin{bmatrix} 0.5 - v_{C_3}^- \\ v_{C_3}^+ \\ P_{\neg 3}\left(x^+ - x^-\right) \\ g(\tau_C, x^+, z^+, p) \end{bmatrix} = 0$$

- Row 1: guard; row 2: reset (reinit to 0)
- Row 3: $P_{\neg 3}$ selects the four states other than v_{C3}; they are continuous
- Row 4: post-event algebraic consistency
- x = (i_{L2}, i_{L1}, v_{C5}, v_{C3}, v_{C1}); z = time derivatives of v_{C5}, v_{C3}, v_{C1}
- Nominal 20 s trajectory: 10 events, 11 smooth segments



Method B: true (solid), initial (dashed), optimized (dotted)

© 2026 Fujitsu Limited

Backup: accuracy of the discrete adjoint

FUJITSU

The adjoint mesh, not the forward solver tolerance, sets the accuracy of the Method B gradient.

Mesh, not tolerance

Forward IDA tolerances control the stored trajectory and the event localization, not the trapezoidal adjoint mesh.

Bias along weak directions

With a uniform target grid, the trapezoidal Jacobians accumulate a small coherent bias along weakly identifiable directions (C_1 and C_5 in the Cauer case).

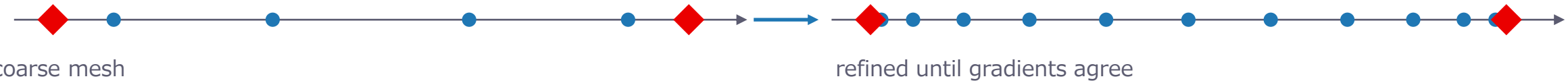
Growth with the event count

For the fixed mesh tested, the bias reached about 4% on e_b at $N = 15$.

Planned extension (untested)

Keep event times as segment boundaries and refine each segment's interior mesh until successive parameter gradients agree.

Adaptive refinement of one segment (planned)



coarse mesh

refined until gradients agree

© 2026 Fujitsu Limited

Backup: bouncing balls, full results

FUJITSU

N	Method	$\lvert\Delta g\rvert/g$	$\lvert\Delta e_g\rvert/e_g$	$\lvert\Delta e_b\rvert/e_b$	Loss	Iter	Total time	ms/iter
3	JAX AD	0.002	0.000	0.001	1.40e-9	300	33.4 s	73.9
3	JAX DA	0.27	1.16	0.16	9.38e-5	300	22.1 s	61.9
3	PyTorch AD	0.003	0.001	0.003	3.89e-9	300	168.6 s	554.4
7	JAX AD	0.002	0.001	0.001	1.63e-9	300	33.4 s	71.2
7	JAX DA	0.15	0.09	0.10	7.76e-6	300	84.0 s	254.8
7	PyTorch AD	0.013	0.010	0.010	6.61e-8	300	525.5 s	1740.4
15	JAX AD	0.07	0.19	0.42	1.35e-5	600	77.2 s	110.1
15	JAX DA	0.21	2.67	4.04	9.28e-4	495	595.7 s	1157.5
15	PyTorch AD	0.000	0.005	0.001	3.88e-8	423	2826.9 s	6652.2

Relative errors in %; loss: final validation loss on a denser held-out grid. Budget: 300 iterations for N = 3 and 7, 600 for N = 15. CPU runs; one run per entry. PyTorch uses odeint backpropagation, not odeint_adjoint.

© 2026 Fujitsu Limited

Backup: comparison with existing approaches

FUJITSU

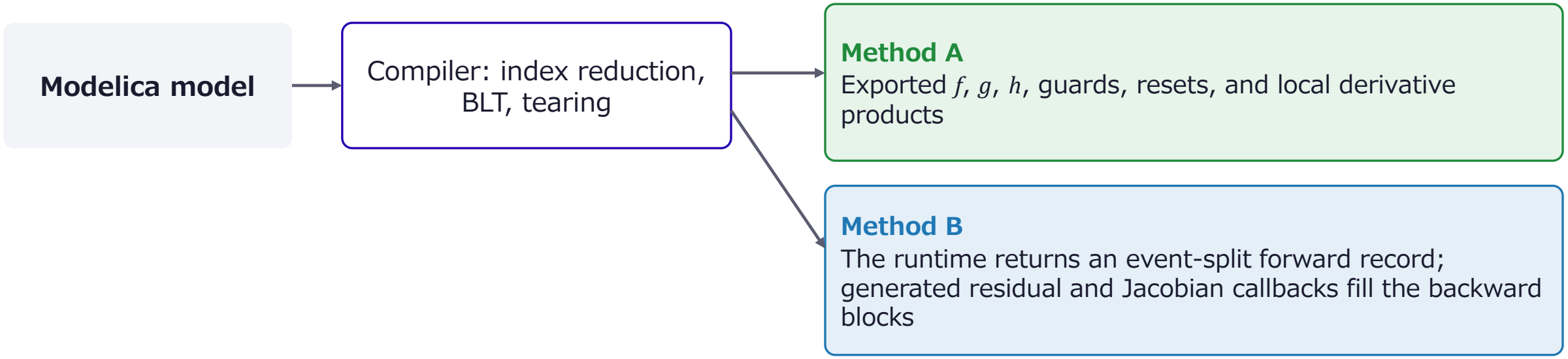
Approach	Model class	Event treatment	Gradient mechanism
Neural ODE, differentiable ODE solvers	ODEs	Usually continuous; events in some solvers	Backpropagation through the solver or continuous adjoint
Neural event ODEs	ODEs with event functions and resets	Events end integration and trigger state updates	AD through event time, terminal state, and reset; locally fixed event sequence
Method A: AD through simulation	Semi-explicit DAEs with state events	Guard-triggered resets; z re-solved after events	AD through reduced simulation, implicit algebraic solution, event time, and reset
Method B: explicit discrete adjoint	Discretized hybrid DAE residual system	Smooth-segment residuals coupled with event residuals	Backward computation of multipliers; explicit event-time stationarity
IDAS continuous adjoint	DAEs with user-defined backward problem	Forward rootfinding; reset and backward jump conditions left to the application	Checkpointed forward trajectory and backward adjoint DAE
Symbolic adjoint compilation	Compiler-transformed equation blocks	OpenModelica prototype targets continuous strong components	Compile-time reverse equations and sparse Jacobian products

© 2026 Fujitsu Limited

Backup: Modelica and FMI integration path

FUJITSU

A proposed integration; it is not standardized FMI functionality and was not implemented in this work.



FMI 3.0 Model Exchange

An importer records event indicators and Event Mode updates, then combines fmi3GetDirectionalDerivative or fmi3GetAdjointDerivative products with event-time and reset terms.

Both paths retain the index reduction and tearing of the Modelica tool.

© 2026 Fujitsu Limited

Backup: IDAS adjoints and symbolic adjoints

FUJITSU

SUNDIALS IDAS adjoint

- IDASolveB integrates a user-defined backward DAE from forward checkpoints.
- IDASolveB performs no backward rootfinding.
- Hybrid use still needs event splitting with guard, event-time, reset, and consistency jump terms.
- We did not implement a native IDAS-adjoint baseline, and we do not make claims about runtime or accuracy against it.

Symbolic adjoints in OpenModelica

- Prototypes generate symbolic adjoints after BLT transformation (Brandt et al., 2026).
- They target continuous strong components and compact sparse Jacobian products.
- Such products could replace local derivatives in our implicit and discrete-adjoint blocks, with our event-time and reset coupling retained.
- Not implemented here; not an equivalent hybrid-event baseline.

© 2026 Fujitsu Limited

Backup: TorchDAE (2026) and this work

FUJITSU

TorchDAE was announced after submission; the paper does not compare against it.

TorchDAE (June 2026)

- Index-1 DAEs; automatic index reduction (Pantelides, dummy derivatives)
- Stiff solvers: BDF1, BDF2, SDIRK TR-BDF2, Radau IIA
- Continuous adjoint sensitivities
- Differentiable events and resets with Hermite interpolation
- vmap and batched GPU execution

This work

- Parameter learning as constrained least squares for hybrid DAEs
- Method A: AD through a JIT-compiled segmented simulation (JAX)
- Method B: explicit discrete adjoint around an external solver (IDA)
- Event-split residual structure with event-time stationarity
- Relation to hybrid optimal control

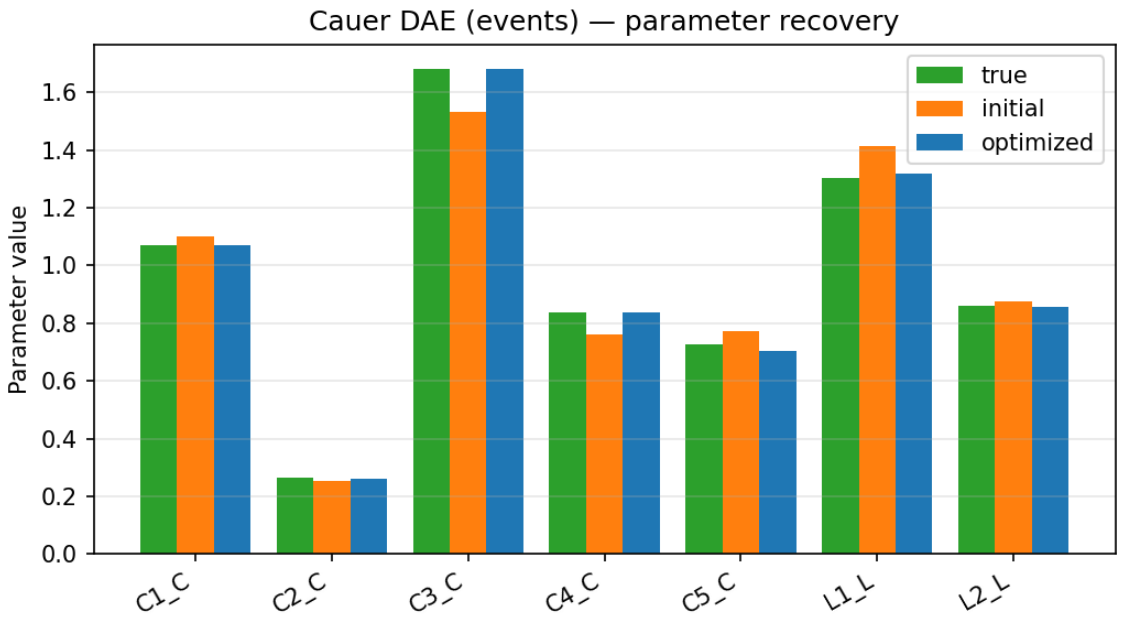
TorchDAE is a third-party library in the PyTorch ecosystem; features as stated in its README: github.com/yousef-rafat/torchdae.
BDF: backward differentiation formula; SDIRK: singly diagonally implicit Runge-Kutta; TR-BDF2: trapezoidal rule with BDF2.

© 2026 Fujitsu Limited

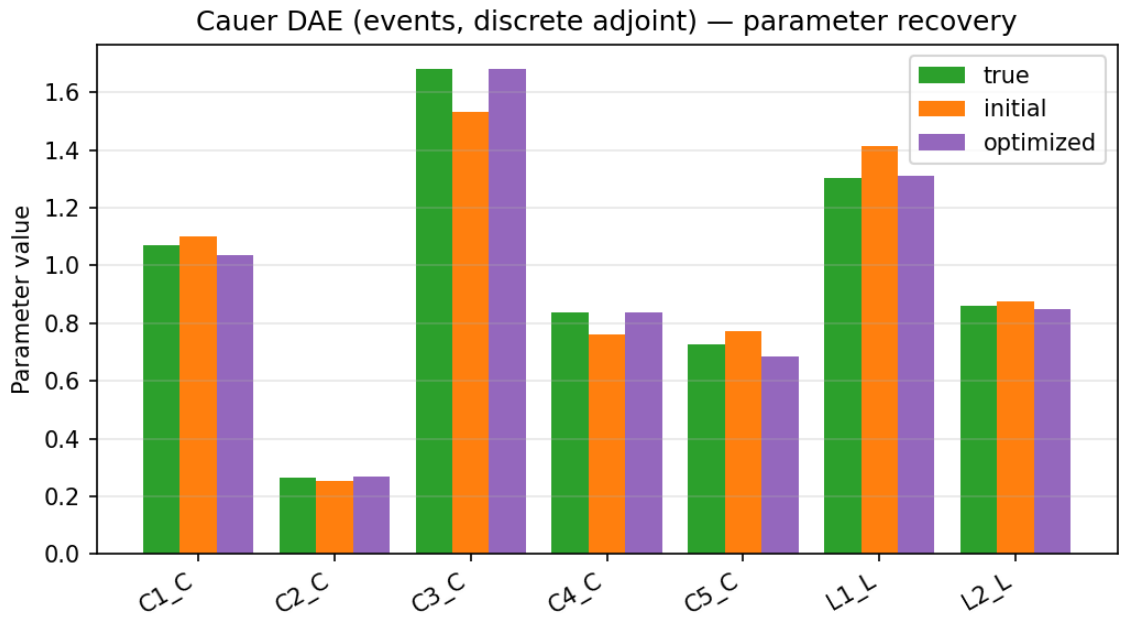
Backup: Cauer parameter recovery

Different parameter combinations give nearly indistinguishable outputs; the output loss is the primary objective.

Method A (AD)



Method B (DA)



Bars: true, initial, and optimized parameter values (figure colors are those of the paper).

© 2026 Fujitsu Limited