Slides: https://t1p.de/Gozo2026

IFDIFF – A MATLAB Toolbox for

Filippov-Type ODEs and DAEs (with State-Dependent Switches)

Andreas Sommer
Interdisciplinary Center for Scientific Computing (IWR)
Heidelberg University

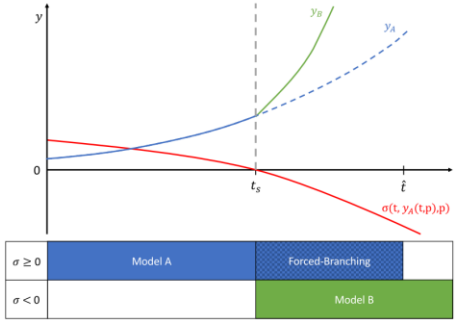
andreas.sommer@iwr.uni-heidelberg.de

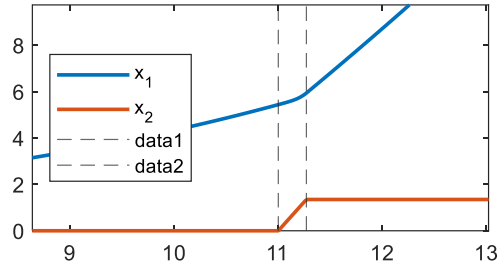
Go20 Conference 2026, May 18-22, Marsalforn, Gozo, Malta

2026-05-18

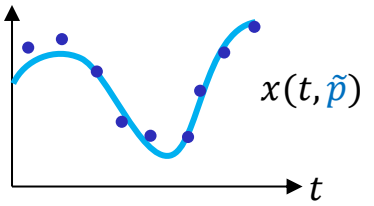
Outline

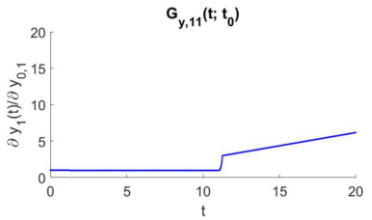
- ODEs with implicit (state-dependent) switches
  - A Canonical Example
  - State Jumps
  - A Glimpse on Switching Function Theory
  - Determination of Switching Time Points



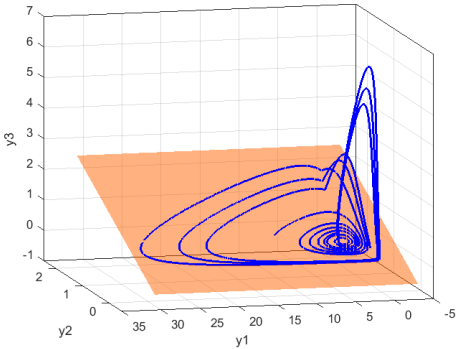


- Filippov Systems
  - A glimpse on Filippov's Theory for Discontinuous ODE
  - Filippov Sliding Mode





- Examples



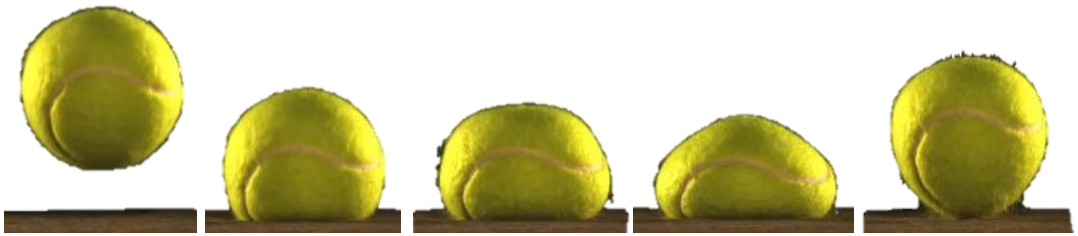
- (Sensitivities and their Propagation through Switches)

Motivation of Implicitly Switched ODEs

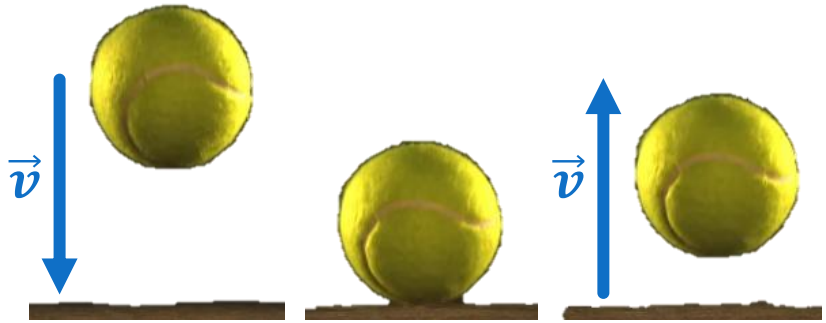
Switched Systems

- **Switched systems** often **arise as simplifications** of complicated continuous processes
- Example: **bouncing ball**

Modelling and simulation of complex material deformation is **very complicated**



Simplification: change in the velocity vector at the contact point



Switching event/model change when the ball hits the ground

Motivation

- Setting: ODE/IVP $\dot{x} = f(t, x, p)$ with rhs function f coded as Matlab file **rhs_f.m**
- **Rapid prototyping** – during model development or model extension
  – quick testing of ideas

state-dependent model switching

```
function dx = rhs_f(t,x,p)
  ...
  if x(2) > vsonic
    dx(3) = submodel_supersonic(t,x,p)
  else
    dx(3) = submodel_subsonic(t,x,p);
  end
  ...
end
```

nonnegativity enforcement:

```
function dx = rhs_f(t,x,p)
  ...
  k1 = x(2)*x(3) - x(4);
  k1 = max(0, k1);
  dx(6) = x(3) * x(4) * sqrt(k1)
  ...
end
```

- Introduced non-differentiabilities require proper treatment ⟹ End of rapid prototyping ☹
- **Goal:** Remove mathematical workload during model development

Switched ODEs. By Example.

Canonical Example

- 2-dimensional system with state-dependent switching:

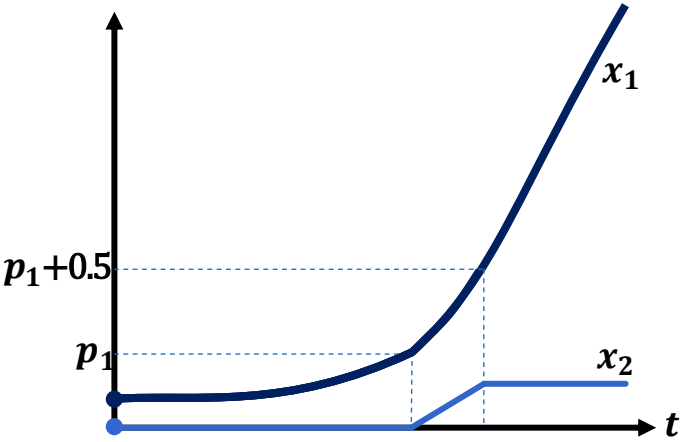
$$\dot{x}(t) = f(t, x, p)$$

$$\dot{x}_1(t) = 0.01 \cdot t^2 + x_2(t)^3$$

$$\dot{x}_2(t) = \begin{cases} 0 & if \quad x_1(t) < p_1 \\ 5 & if \; p_1 \le x_1(t) < p_1 + 0.5 \\ 0 & if \quad x_1(t) \ge p_1 + 0.5 \end{cases}$$

$$x(0) = (1,0)^T \qquad p_1 = 5.437$$

- Qualitative sketch of solution:



Qualitative solution construction

0. System is monotonically increasing
1. Start at $(x_1, x_2) = (1,0)$.
   $\dot{x}_2 = 0$, since $x_1 < p_1$ for a while
   → 2nd component $x_2 \equiv 0$ for a while
   cubic part $x_2(t)^3$ in 1st component is 0
   → 1st component $x_1(t)$ is a cubic polynomial in t
2. After a while, $x_1 \ge p_1$
   → 2nd component rises linearly with slope 5
   → 1st component rises even quicker now due to $x_2(t)^3$
3. After a while, $x_1 \ge p_1 + 0.5$
   → 2nd component stays constant (zero slope)
   → 1st component keeps rising

Let's try in Matlab!

How switched ODEs are frequently “solved”…

Just ignore the switches and integrate!
The error control will do fine!

No. It won’t.

Malicious Example: Unnoticed Disaster

- 2-dimensional system with state-dependent switching:

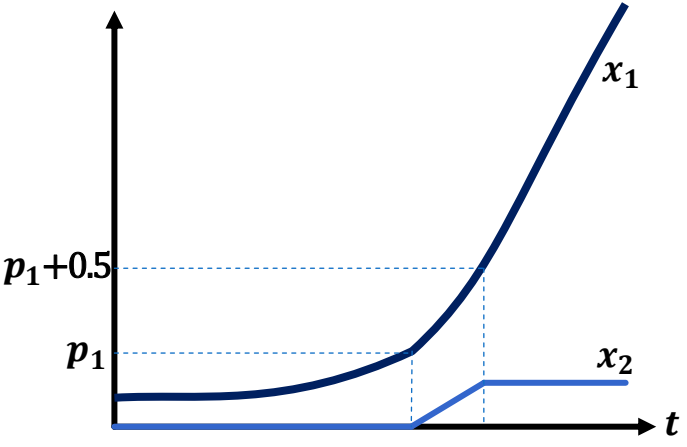
$$\dot{x}(t) = f(t,x,p)$$

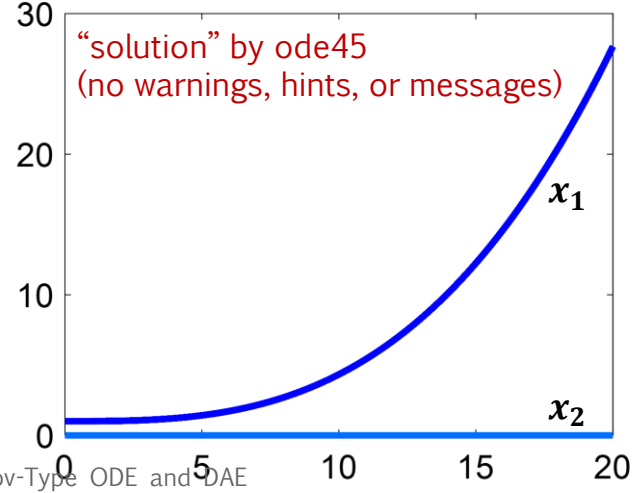
$$\dot{x}_1(t) = 0.01 \cdot t^2 + x_2(t)^3$$

$$\dot{x}_2(t) = \begin{cases} 0 & if \quad x_1(t) < p_1 \\ 5 & if \; p_1 \le x_1(t) < p_1 + 0.5 \\ 0 & if \quad x_1(t) \ge p_1 + 0.5 \end{cases}$$

$$x(0) = (1,0)^T \qquad p_1 = 5.437$$

- qualitative sketch of solution:

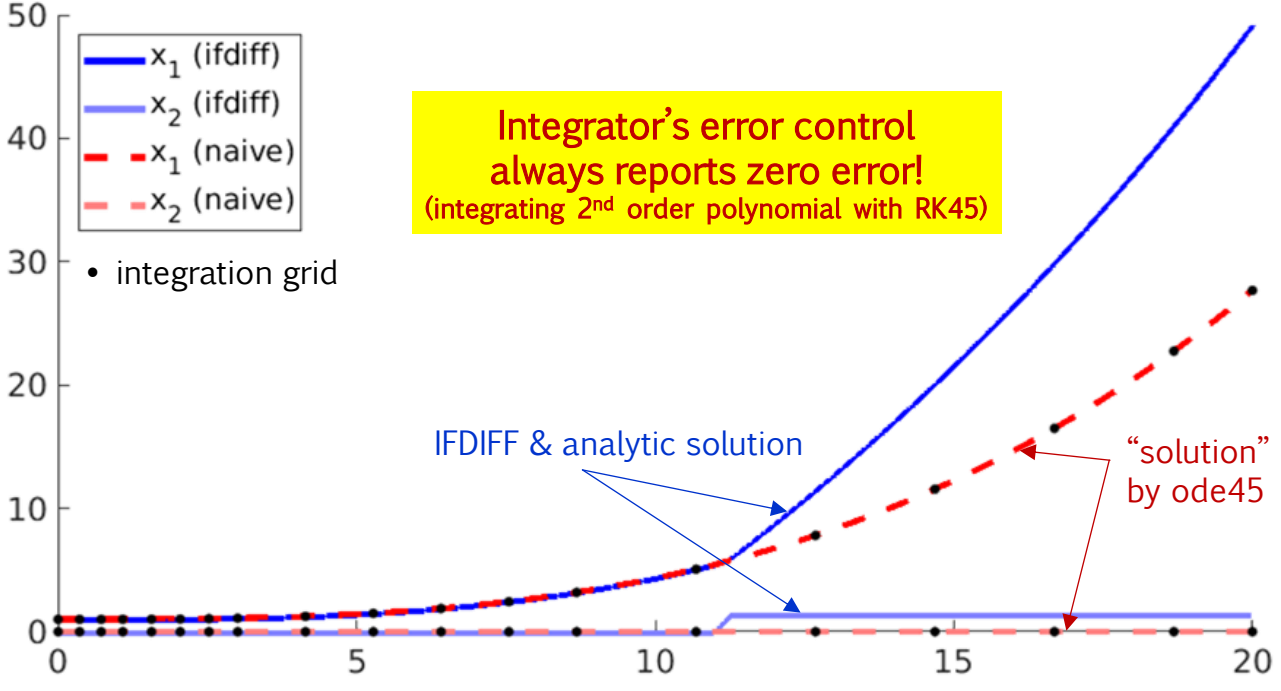





```
function dx = f(t,x,p)

  dx = zeros(2,1);
  dx(1) = 0.01 * t.^2  +  x(2).^3;

  if x(1) < p(1)
     dx(2) = 0;
  else
     if x(1) < p(1) + 0.5
        dx(2) = 5;
     else
        dx(2) = 0;
     end
  end

end

% problem setup
tspan = [0 20];
x0    = [1 0];
p     = 5.437;

% integration
sol = ode45(@(t,x)f(t,x,p),tspan,x0);
```

Malicious Example: Unnoticed Disaster

- naive integration ("rely on error control") vs. IFDIFF
- program flow "jumps" from **dx(2)=0** to **dx(2)=0**



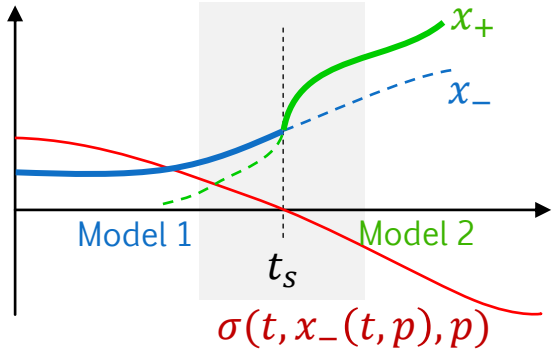
Matlab's ode45 (RK), independent of tolerances, no warnings/errors/hints

```
function dx = f(t,x,p)

  dx = zeros(2,1);
  dx(1) = 0.01 * t.^2  +  x(2).^3;

  if x(1) < p(1)
     dx(2) = 0;
  else
     if x(1) < p(1) + 0.5
        dx(2) = 5;
     else
        dx(2) = 0;
     end
  end

end

% problem setup
tspan = [0 20];
x0    = [1 0];
p     = 5.437;

% integration
sol = ode45(@(t,x)f(t,x,p),tspan,x0);
```

How switched ODEs should be integrated

Switched ODEs: A Glimpse onto Solution Theory

- Switched IVP: $\dot{x}(t) = f\big(t, x(t), p, \operatorname{sign} \sigma(t, x(t), p)\big) \quad x(t_0) = x_0(p)$ (switching function)
- Switching events t_s characterized by zero crossings: $\sigma(t_s, x, p) = 0$
- Technical assumptions:
  - Individual "models" can be evaluated in an environment of each switch
  - Isolated switching points



- **Strategy**: Monitor σ during integration. Stop and restart at zero crossings.
- **Caveats**: - requires **extensive re-formulation** even for small model changes ⟹ bye bye rapid prototyping ☹
  - n conditionals ⟹ up to 2^n possible models (**impracticable** for $n > 20$)
- Canonical Example:

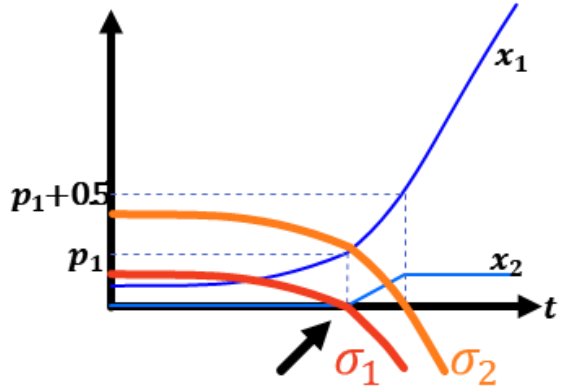
$$\dot{x}_1(t) = 0.01 \cdot t^2 + x_2(t)^3$$

$$\dot{x}_2(t) = \begin{cases} 0 & if \quad x_1(t) < p_1 \\ 5 & if \; p_1 \le x_1(t) < p_1 + 0.5 \\ 0 & if \quad x_1(t) \ge p_1 + 0.5 \end{cases}$$

$$x(0) = (1, 0)^T \qquad p_1 = 5.437$$

Suitable switching functions

$\sigma_1(t, x, p) = p_1 - x_1(t)$
$\sigma_2(t, x, p) = p_1 + 0.5 - x_1(t)$



Switched ODEs: 3 Types

- Switched IVP: $\dot{x}(t) = f\big(t, x(t), p, \operatorname{sign} \sigma(t, x(t), p)\big) \qquad x(t_0) = x_0(p)$

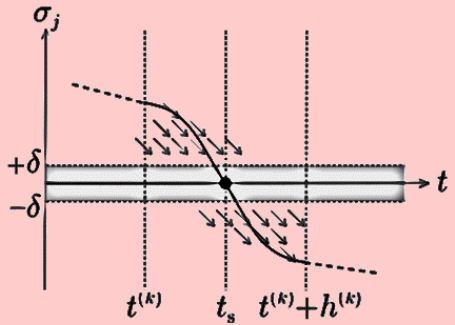
switching function

- **3 types of switches** (the good, the bad, and the ugly):

Consistent switch

leaving $\sigma = 0$ in well-defined direction

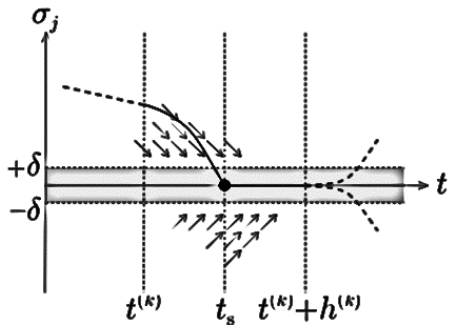
$\frac{d\sigma^-}{dt_s}(t_s) \cdot \frac{d\sigma^+}{dt_s}(t_s) > 0$



Inconsistent switch

staying at $\sigma = 0$

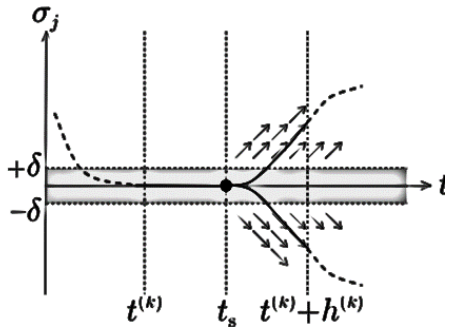
$\frac{d\sigma^-}{dt_s}(t_s) < 0$ and $\frac{d\sigma^+}{dt_s}(t_s) > 0$



Bifurcation

leaving $\sigma = 0$ in both directions

$\frac{d\sigma^-}{dt_s}(t_s) > 0$ and $\frac{d\sigma^+}{dt_s}(t_s) < 0$



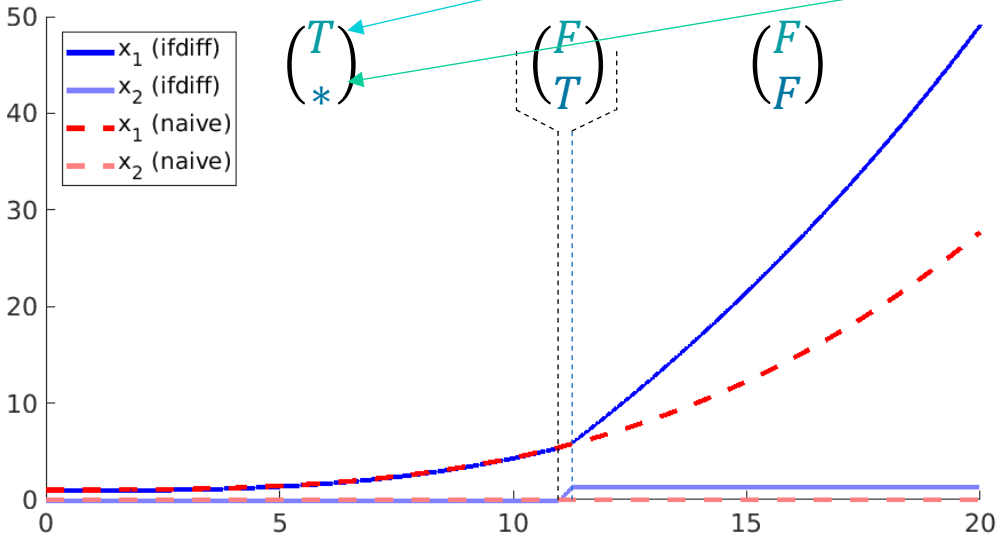
Illustrations: C. Kirches, A Numerical Method for Nonlinear Robust Optimal Control with Implicit Discontinuities and an Application to Powertrain Oscillations, 2006

Where are Switching Functions in Computer Code?

Surveilling Switches using Signatures and Augmented IFs

- **Switching functions $\sigma(\cdot)$ implicitly given** in the code:
  - e.g. as the condition in **`if`**-expressions
  - usually **depend on intermediate values, sub-functions, etc.**
- IFDIFF surveils the switching behavior
  - Store the **branching pattern (signature)** of individual non-differentiable operators and **monitor changes** in that signature

```
function dx = f(t,x,p)

  dx = zeros(2,1);
  dx(1) = 0.01 * t.^2  +  x(2).^3;

  if x(1) < p(1)
     dx(2) = 0;
  else
     if x(1) < p(1) + 0.5
        dx(2) = 5;
     else
        dx(2) = 0;
     end
  end

end
```



Automatic processing in IFDIFF:

- augment **`if`**-expressions via source code trans-form* with monitoring and forced-branching capabilities

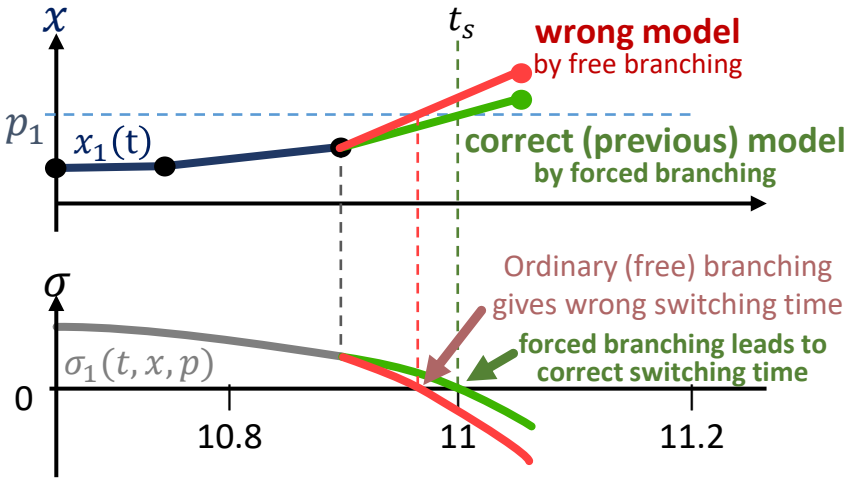
  * based on Matlab's internal abstract syntax tree `mtree`
- IFDIFF considers intermediate variables, nested functions, recursion, ...

How to Compute the Switching Times?

Surveilling Switches using Signatures and Augmented IFs

- **Determine root of switching function** $\sigma(t,x,p)$ using continuous solution representation $x(t)$ from integrator
  - ⇒ **free branching** leads to **wrong (mixed) model evaluation**
  - ⇒ **must not switch** for consistency **!!**

- **IFDIFF:** **Monitor** the `if`-condition, detect switch, but **force branching** into previously active branch



- previous integrator steps
- step with free branching (wrong!)
- step with forced branching

```
function dx = f(t,x,p)

  dx = zeros(2,1);
  dx(1) = 0.01 * t.^2  +  x(2).^3;

  if x(1) < p(1)
     dx(2) = 0;
  else
     if x(1) < p(1) + 0.5
        dx(2) = 5;
     else
        dx(2) = 0;
     end
  end

end
```

IFDIFF:

- Generates **only required** switching functions σ during integration ⇒ **breaks the curse of dimension**
- Determines the **switching time up to machine precision**

Surveilling Switches using Signatures and Augmented IFs

- IFDIFF provides **automatic preprocessing** of RHS code files:

```
function dx = f(t,x,p)

  dx = zeros(2,1);
  dx(1) = 0.01 * t.^2  +  x(2).^3;

  if x(1) < p(1)
     dx(2) = 0;
  else
     if x(1) < p(1) + 0.5
        dx(2) = 5;
     else
        dx(2) = 0;
     end
  end
end
```

```
function dx = rhs_preprocessed_canonicalExampleRHS(datahandle,t,x,p)
    dx = zeros( 2, 1 );
    dx( 1 ) = 0.01 * t .^ 2 + x( 2 ) .^ 3;
    switchEval_if_1 = x( 1 ) - (p( 1 ));
    condValue = ctrlif( switchEval_if_1, false, true, 1, 0, datahandle );
    if condValue
        dx( 2 ) = 0;
    else
        switchEval_if_2 = x( 1 ) - (p( 1 ) + 0.5);
        condValue = ctrlif( switchEval_if_2, false, true, 2, 0, datahandle );
        if condValue
            dx( 2 ) = 5;
        else
            dx( 2 ) = 0;
        end
    end
end
```

```
function svalue = sw_rhs_preprocessed_f_10(dhandle,t,x,p)
    switchEval_if_1 = x( 1 ) - (p( 1 ));
    switching_value = switchEval_if_1;
end
```

switching function $\sigma_1(t,x,p)$

- IFDIFF uses the best available code processor: **Matlab's own parser.**
  Internally, a Matlab program is represented by an "mtree" object, an (undocumented) abstract syntax tree.

Automatic Generation of Switching Functions

- **All switching functions must be monitored** during integration to detect model changes and state jumps

  ⇒ $\boldsymbol{n}$ **conditionals** may lead up to $\mathbf{2^n}$ **different program flows**, i.e. 2^n different models!

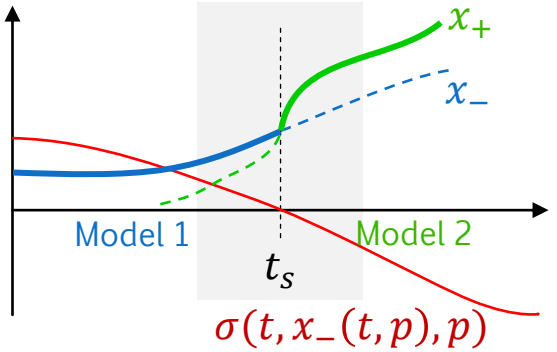
  ⇒ Even for moderate numbers of $n > 10$, that is **not computationally feasible**

- In most cases, not all possible switches actually switch

  Example from pharmacology: ≈700 possible switches, but only 3 of them switching!

- **IFDIFF generates only actually required switching functions!**

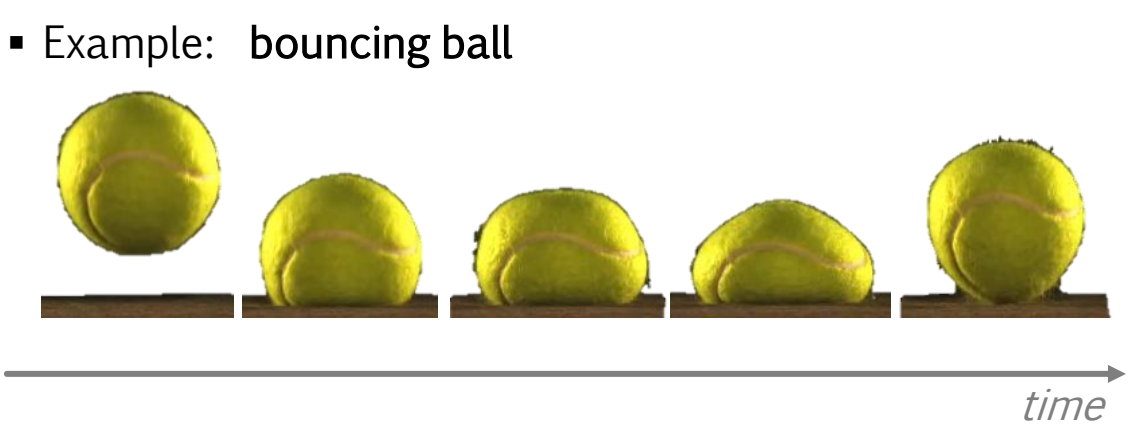
  ⇒ no excessive precomputations needed

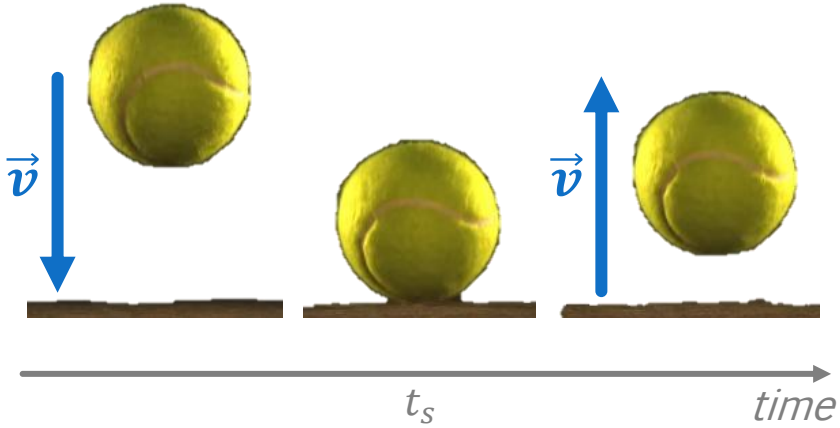


Example: State Jumps

State Jumps

- Example: **bouncing ball**





- **Simplification**: Swap sign in the velocity vector upon ground contact
  state jump

Switching event/model change when the ball hits the ground

- Dynamics: $\begin{pmatrix}\dot{h}\\ \dot{v}\end{pmatrix} = \begin{pmatrix} v \\ -9.81\end{pmatrix}$, Initial value $\begin{pmatrix}h(t_0)\\ v(t_0)\end{pmatrix} = \begin{pmatrix}h_0\\ v_0\end{pmatrix}$

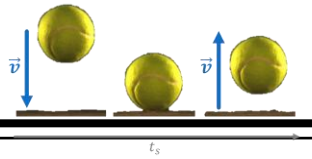
- On Impact: h (height) unchanged
  v (velocity) sign changed and damping by factor p_1

- Update: $x^+(t_s) = \Delta(t_s, x_-(t_s), p) = \begin{pmatrix} 0 \\ (-1+p_1)\cdot v_-(t_s)\end{pmatrix}$

jump function Δ

$$\Delta : I \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_p} \to \mathbb{R}^n$$
$$(t, x_-(t), p) \mapsto x_+(t)$$

State Jumps: Bouncing Ball Example



- IFDIFF: Dynamics and jump function Δ coded in rhs:

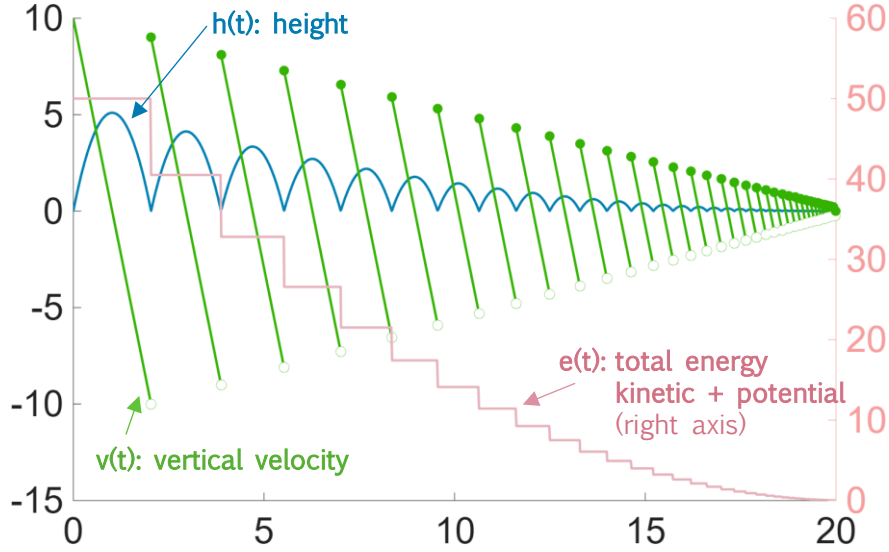
Dynamics: $\begin{pmatrix}\dot{h}\\ \dot{v}\end{pmatrix} = \begin{pmatrix} v \\ -9.81\end{pmatrix}$

Jump Function: $\Delta(t_s, x_-(t_s), p) = \begin{pmatrix} 0 \\ (-1+p_1)\cdot v_-(t_s)\end{pmatrix}$

- Right-hand-side function:

```
function dx = bounceballRHS(~, x, p)
   dx = [x(2); -9.81];
   if ifdiff_jumpif(x(1), -1)
      deltaH = 0;
      deltaV = (-1+p(1))*x(2);
      ifdiff_update([deltaH; deltaV]);
   end
end
```

possibility to specify direction of zero crossing
0: both directions
+1: towards positivity
−1: towards negativity



Filippov Sliding Mode Detection and Handling

Switched ODEs: 3 Types

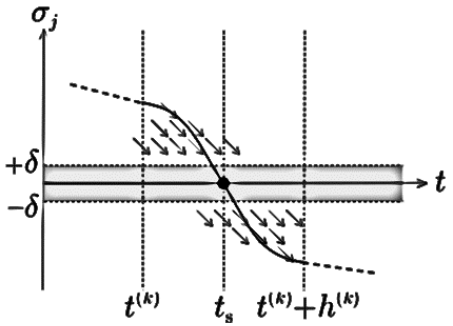
- Switched IVP: $\dot{x}(t) = f\big(t, x(t), p, \operatorname{sign} \sigma(t, x(t), p)\big) \qquad x(t_0) = x_0(p)$

  switching function

- **3 types of switches** (the good, the bad, and the ugly):

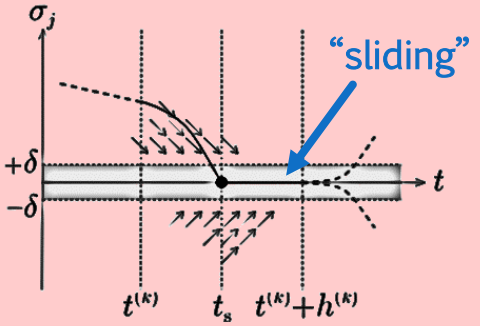
Consistent switch
leaving $\sigma = 0$ in well-defined direction

$\frac{d\sigma^-}{dt_s}(t_s) \cdot \frac{d\sigma^+}{dt_s}(t_s) > 0$



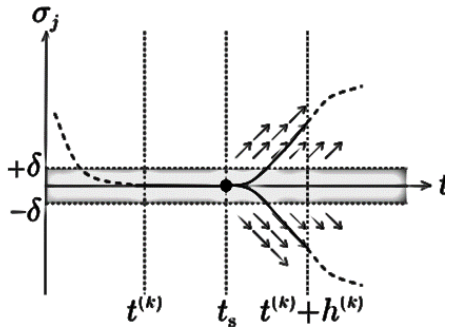
Inconsistent switch
staying at $\sigma = 0$

$\frac{d\sigma^-}{dt_s}(t_s) < 0$ and $\frac{d\sigma^+}{dt_s}(t_s) > 0$



Bifurcation
leaving $\sigma = 0$ in both directions

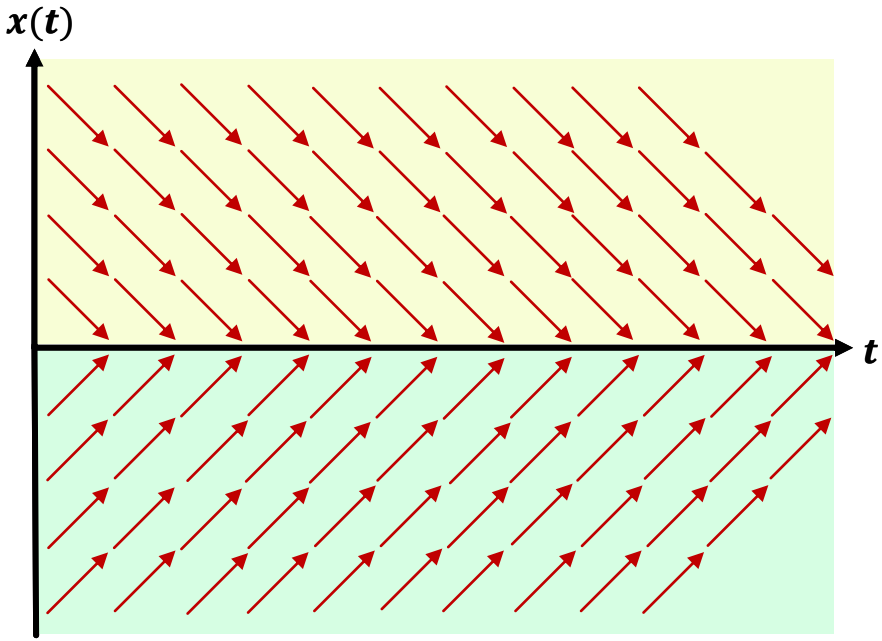
$\frac{d\sigma^-}{dt_s}(t_s) > 0$ and $\frac{d\sigma^+}{dt_s}(t_s) < 0$

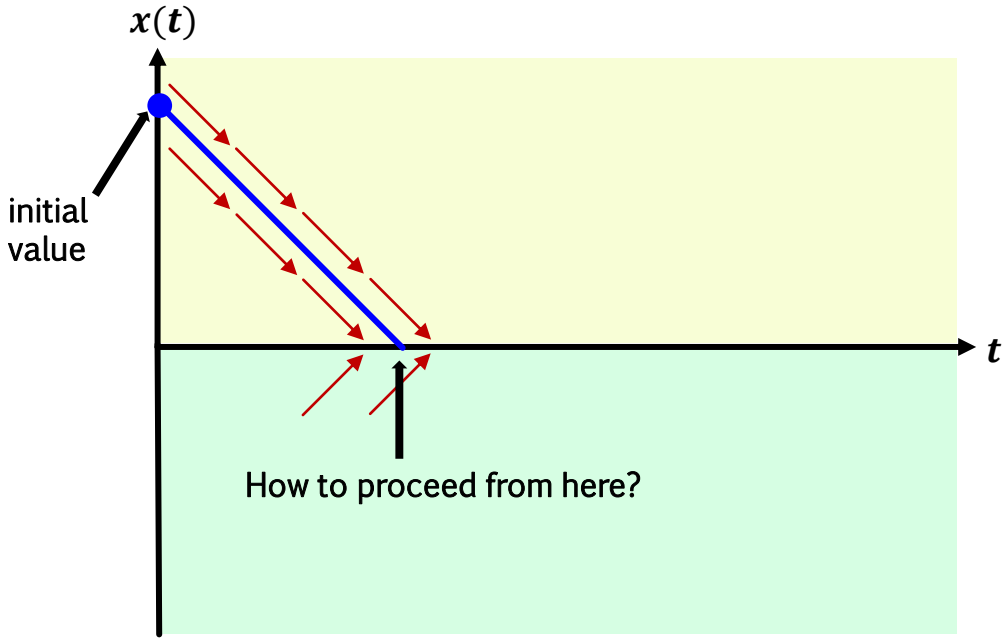


Illustrations: C. Kirches, A Numerical Method for Nonlinear Robust Optimal Control with Implicit Discontinuities and an Application to Powertrain Oscillations, 2006

State-dependent switching in a differential equation: A simple(?) example

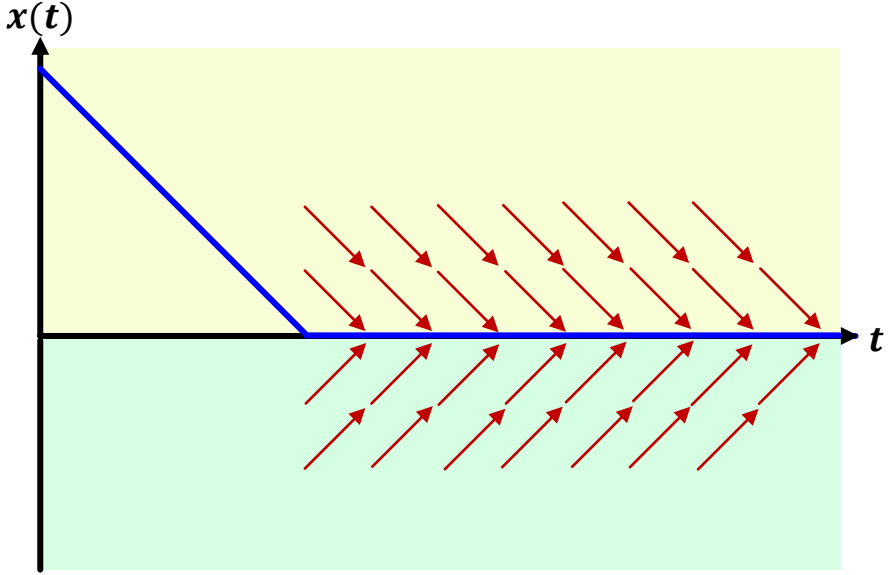
Consider FILIPPOV-Prototype: $\frac{dx}{dt} = \begin{cases} -1 & if \ \ x \geq 0 \\ +1 & if \ \ x < 0 \end{cases}$





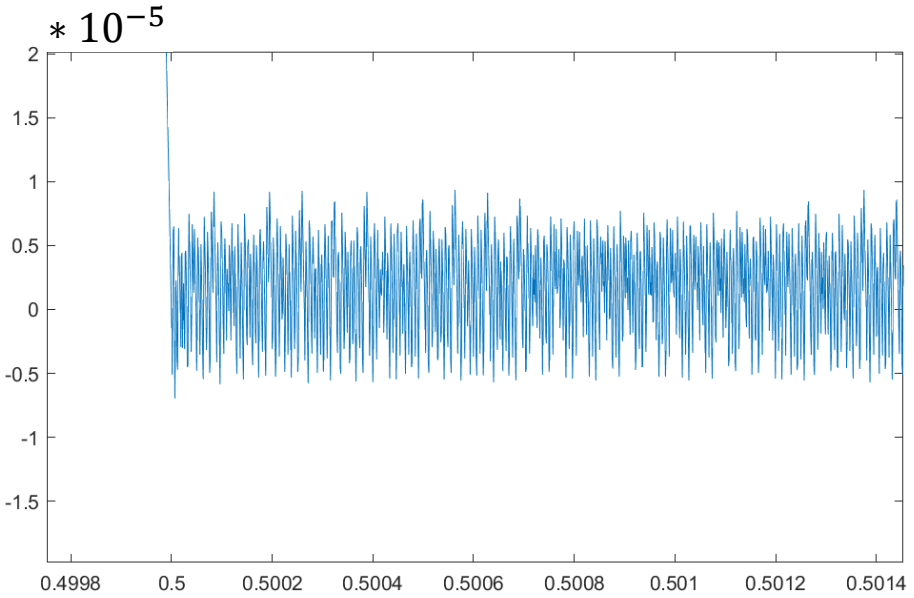
- Two models: „descend“ (−1) and „ascend“ (+1)
- ↗ : vector field (continuation of solution)

- FILIPPOV'S ansatz:
  Combine the two models at the intersection manifold



- Convex combination of two models
  $$f(t,x) = \alpha \cdot f_1(t,x) + (1-\alpha) \cdot f_2(t,x)$$
- Note: $\alpha = \alpha(t,x)$

- Challenge: algorithmic solution
  → ignoring switches leads to chattering
  → classical solvers **fail**

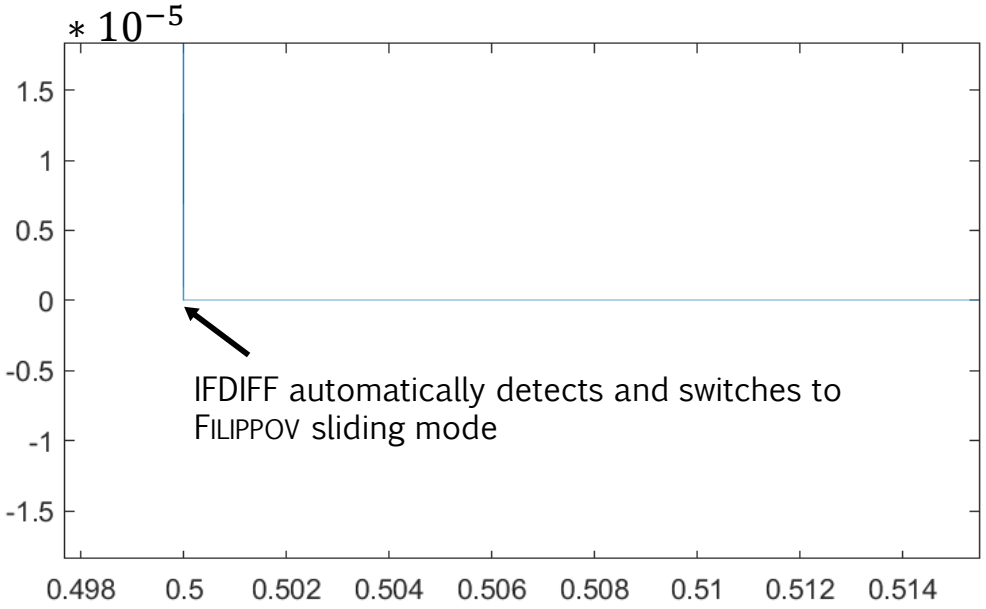


„Solution" with inadequate method (**without** IFDIFF)

Runge-Kutta 4(5) by Dormand/Prince, Matlab's ode45

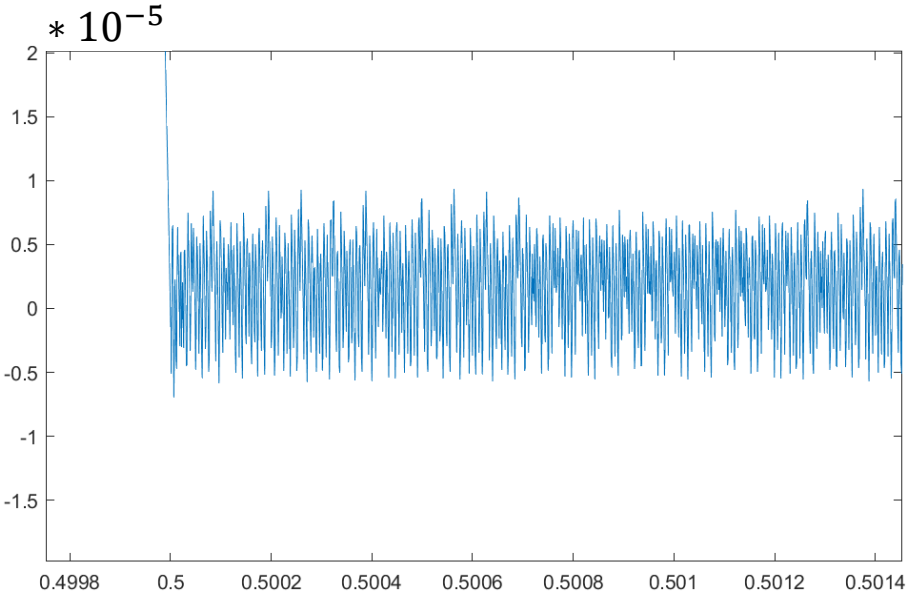
Solution generated by IFDIFF

- **Solution generated by IFDIFF**
  - → automatic detection of FILIPPOV sliding mode
  - → automatic generation of model blending



Correct solution with IFDIFF

- Challenge: algorithmic solution
  - → ignoring switches leads to chattering
  - → classical solvers **fail**

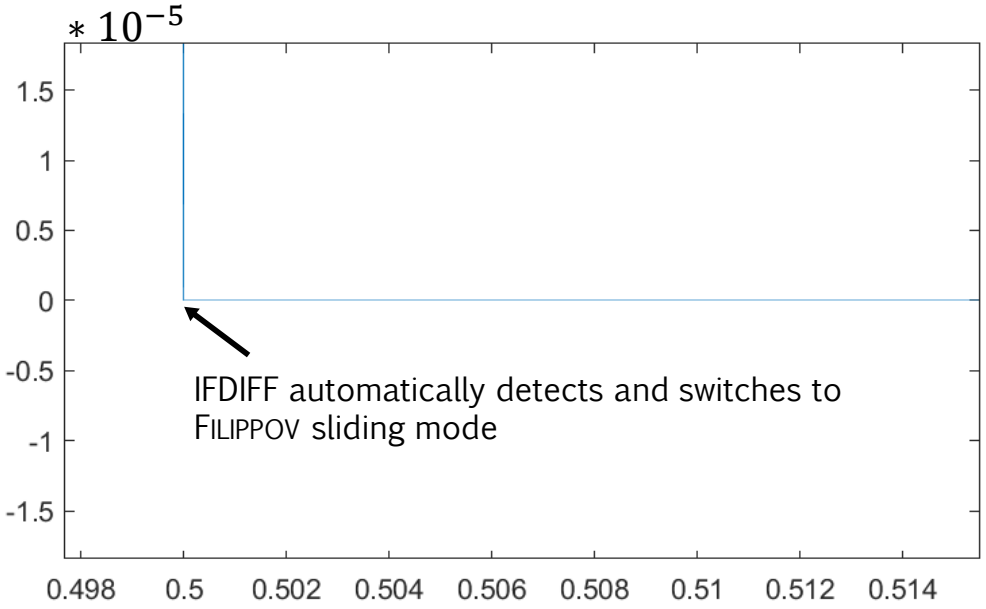


„Solution“ with inadequate method (**without** IFDIFF)

Runge-Kutta 4(5) by Dormand/Prince, Matlab‘s ode45

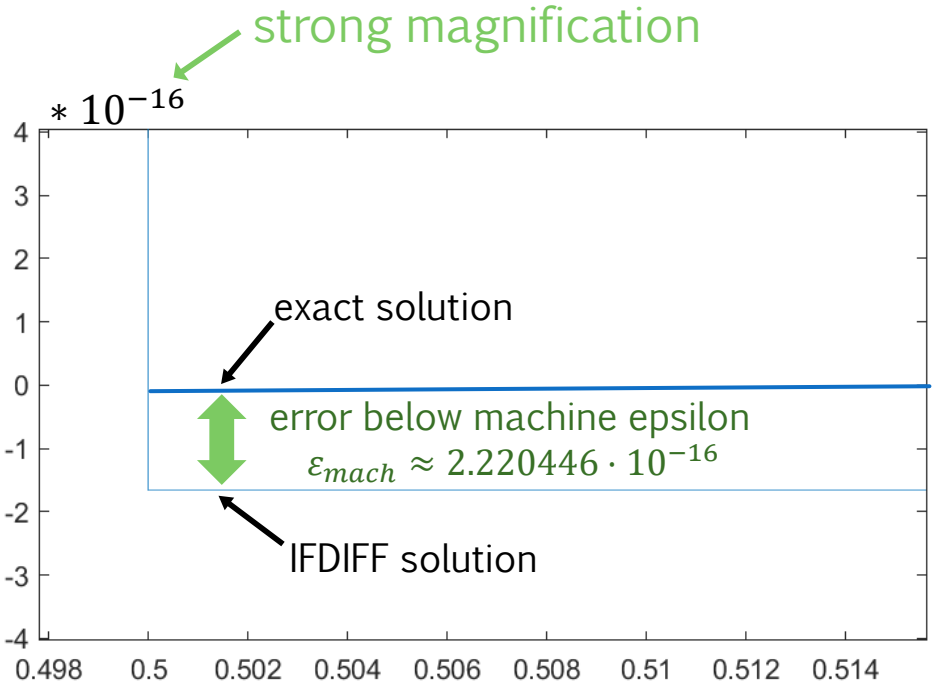
Comparison to exact solution

- Solution generated by IFDIFF



Correct solution with IFDIFF

- Exact solution (analytical)



Comparison: exact solution and IFDIFF solution

Filippov: Computing the Convex-Combination

Consider the single-switched IVP:

$$\frac{dx}{dt}(t) = f\big(t, x(t), p, \operatorname{sgn}\big(\sigma(x)\big)\big), \qquad t \in T$$

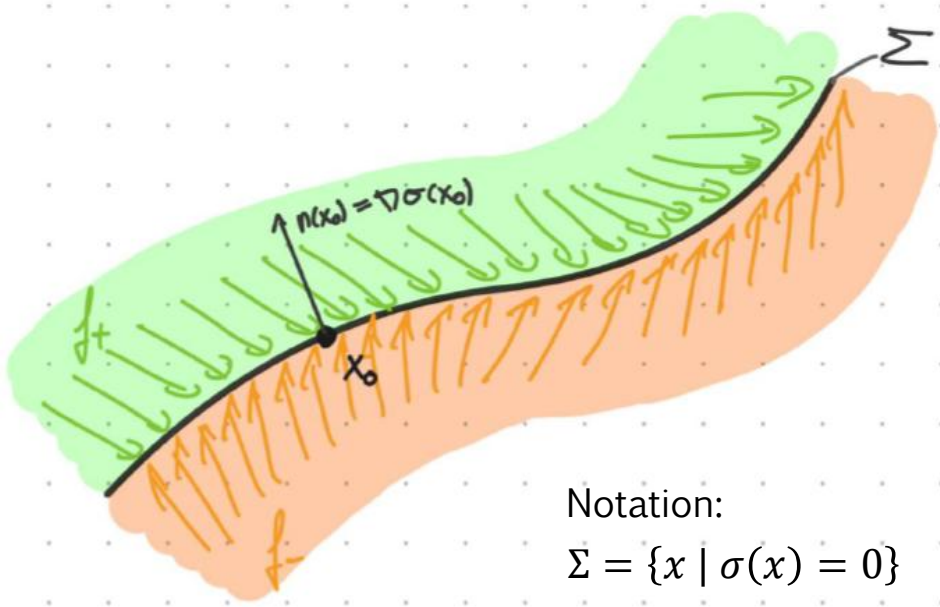
$$x(t_0) = x_0$$

where we assume $\sigma(x_0) = 0$ with inconsistent switching at x_0:

$$n^T(x) f_-(x) > 0$$

and $$n^T(x) f_+(x) < 0$$

with normal vector $n(x) = \nabla\sigma(x)$



Notation:

$\Sigma = \{x \mid \sigma(x) = 0\}$

How to choose the **convex combination parameter** $\alpha(t, x)$ in

$f_F(t, x) = \alpha(t, x) f_-(t, x) + (1 - \alpha(t, x)) f_+(t, x)$?

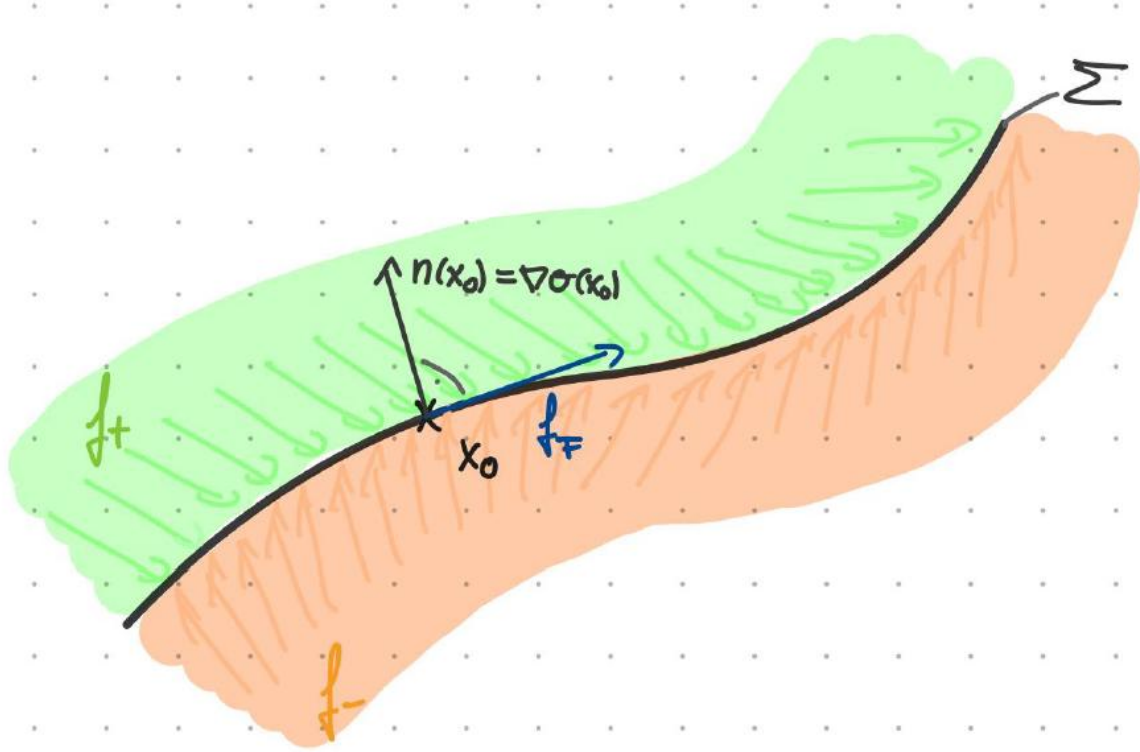
Filippov: Computing the Convex-Combination

To continue the solution on the zero-manifold $\Sigma = \{x \mid \sigma(x) = 0\}$,

$f_F(t,x)$ must be orthogonal to the normal $n\big(x(t)\big) = \nabla(\sigma(x))$

Compute $\alpha(t,x)$ from $n^T(x) f_F(t,x) = 0$:

$$\alpha(t,x) := \frac{n^T(x) f_+(t,x)}{n^T(x)\big(f_+(t,x) - f_-(t,x)\big)}$$



Example: Filippov-Type 1 Predator – 2 Prey Model

Example: Filippov-type 1 Predator – 2 Prey Model

- 3D Predator-Prey Model [1], [2]

$$\dot{\mathbf{x}} = \begin{cases} \begin{pmatrix} (r_1 - \beta_1 P) p_1 \\ r_2 p_2 \\ (e\, q_1 \beta_1 p_1 - m) P \end{pmatrix} & \text{if} \quad H(p_1, p_2, P) > 0, \\ \begin{pmatrix} r_1 p_1 \\ (r_2 - \beta_2 P) p_2 \\ (e\, q_2 \beta_2 p_2 - m) P \end{pmatrix} & \text{if} \quad H(p_1, p_2, P) < 0, \end{cases}$$

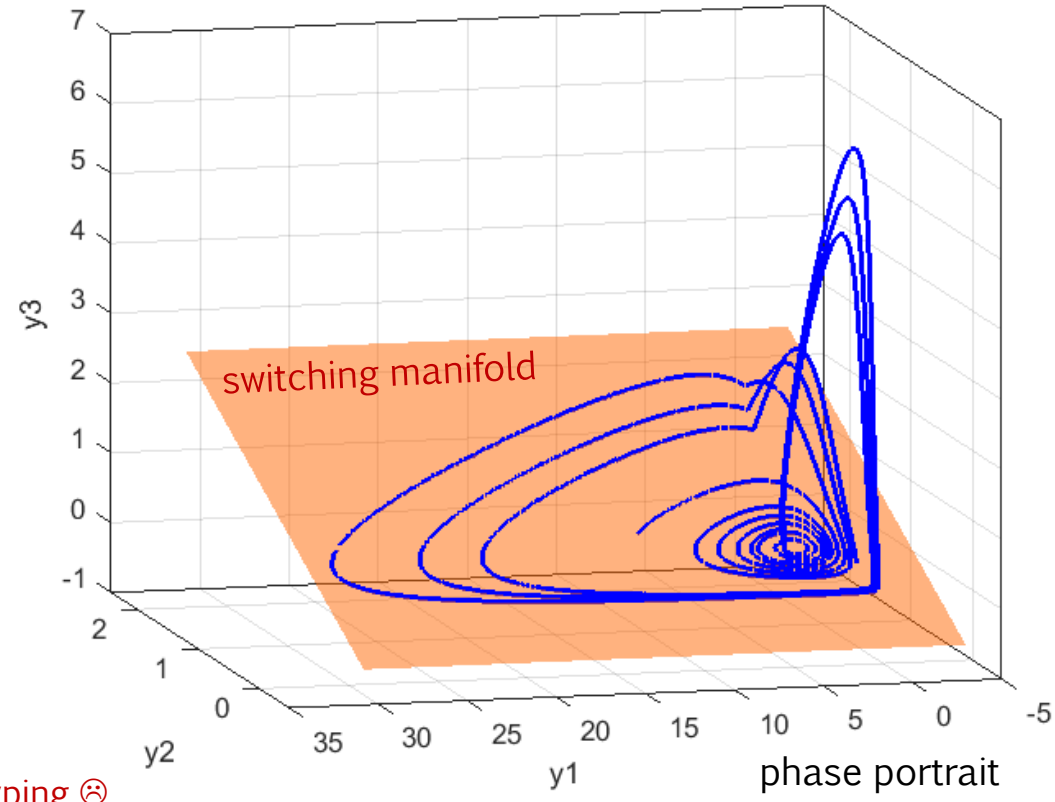
where $\dot{\mathbf{x}} = \left(\dot{p}_1, \dot{p}_2, \dot{P}\right)^T$, $(p_1, p_2, P) \in \mathbb{R}^3_{\geq 0}$ and

$$H(p_1, p_2, P) = \beta_1 p_1 - a_q \beta_2 p_2.$$

Does this system enter a sliding mode?

Depends on parameter values!
⇒ a priori unknown, requires analysis

⟹ bye bye rapid prototyping ☹

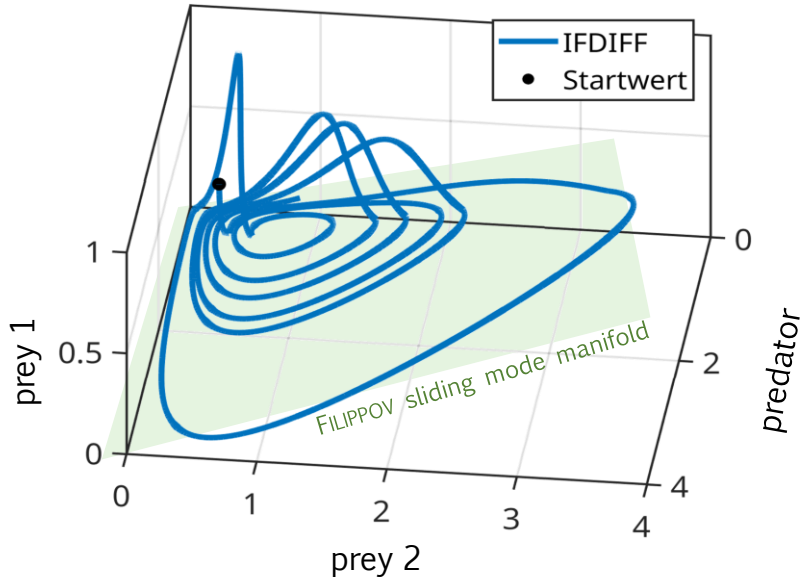


[1] Piltz, S.H., Porter, M.A., Maini, P.K.: Prey switching with a linear preference trade-off. SIAM J. Appl. Dyn. Syst. 13(2), 658–682 (2014)
[2] Carvalho, T., Duarte Novaes, D. & Gonçalves, L.F.: Sliding Shilnikov connection in Filippov-type predator–prey model. Nonlin Dyn 100, 2973–2987 (2020)

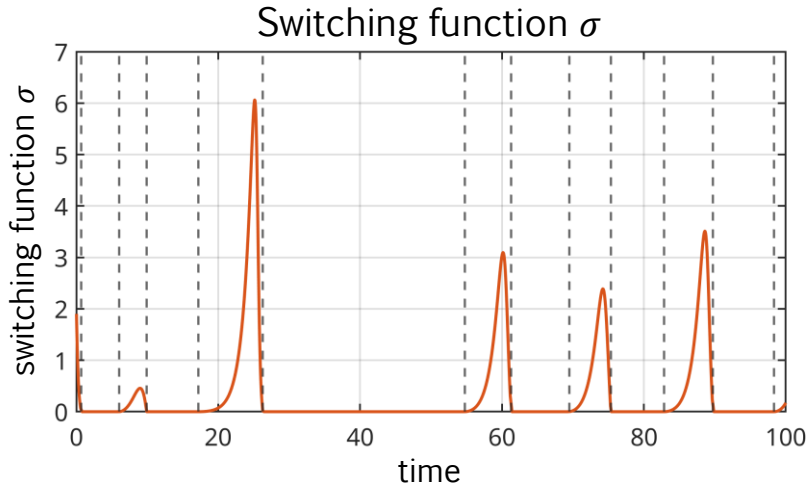
Example: Filippov-type 1 Predator – 2 Prey Model

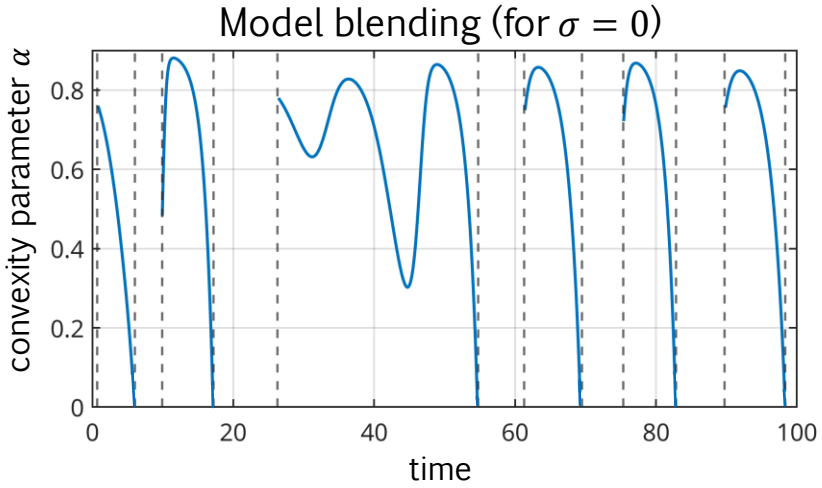
- Solution enters and exists the FILIPPOV sliding manifold several times
- Model blending for $\sigma = 0$:

$$f(t,x) = \alpha(t,x) \cdot f_1(t,x) + \big(1 - \alpha(t,x)\big) \cdot f_2(t,x)$$



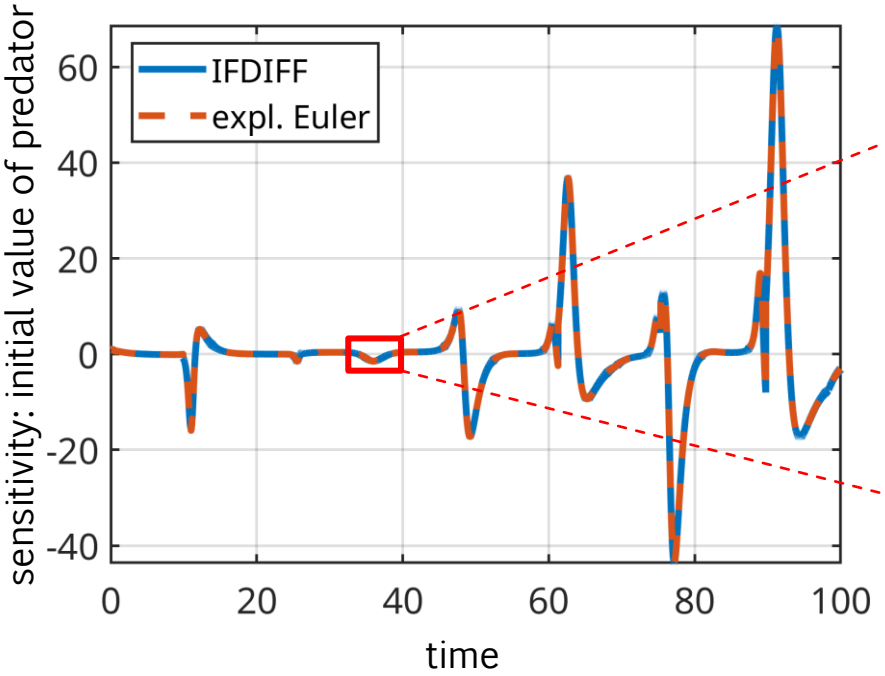
- Note: Leaving manifold when $\alpha = 0$ or $\alpha = 1$



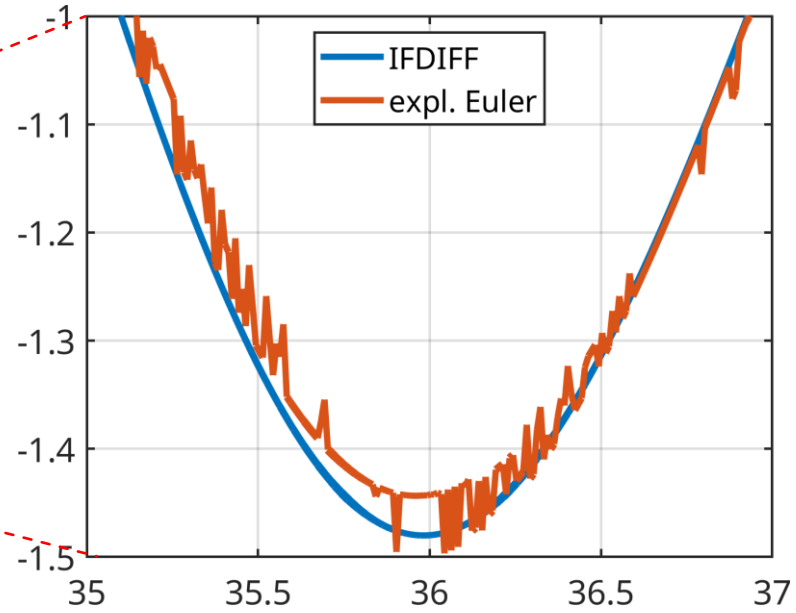


Filippov 2-Prey-1-Predator System: Sensitivities

- Comparison to extremely accurate sensitivities generated by a tailored explicit Euler integrator [step size 10^{-7}] (expl. Euler is ***extremely*** computationally costly!)
- **IFDIFF** delivers **fast and accurate solution and sensitivities**



Correct Sensitivities with IFDIFF

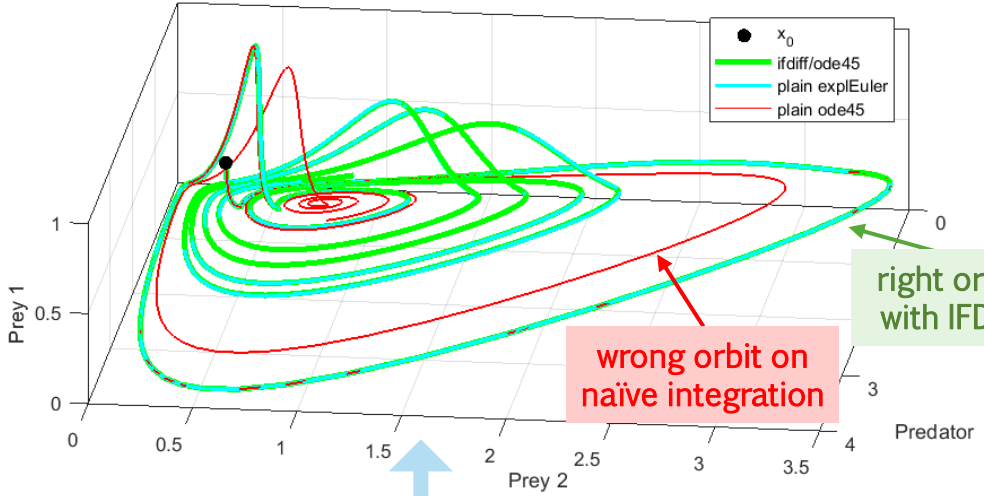


Severe inaccuracies in expl. Euler
Smooth (correct) sensitivity with IFDIFF

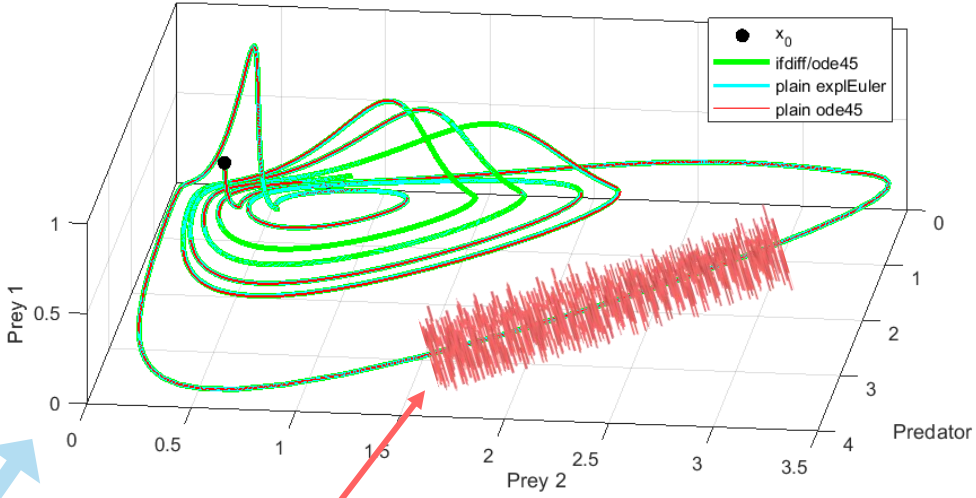
(1) E. Reinhold Mannshardt, 1978, https://doi.org/10.1007/BF01397472

Some Results: Filippov-type 1 Predator – 2 Prey Model

Phase portrait with $AbsTol = Reltol = \mathbf{10^{-5}}$



Phase portrait with $AbsTol = Reltol = \mathbf{10^{-7}}$



RelTol|AbsTol	10^{-5}		10^{-6}		10^{-7}		$10^{-8}\|10^{-12}$	
	time	acc*	time	acc*	time	acc*	time	acc*
plain ode45	0.8s	✘	6.5s	✘	71.3s	O	>10min	?
IFDIFF/ode45	2.6s	✓	3.0s	✓	3.4s	✓	5.8s	✓

* compared to expl. Euler with step size 10^{-7}

? ode45 abort after ≈10min on 8GB memory limit due to extreme oscillations on switching manifold

⇒ Sensitivities inaccessible and no solution for high accuracy

⇒ IFDIFF delivers fast and accurate solutions and sensitivities

Example: Stick-Slip (Dry Friction)

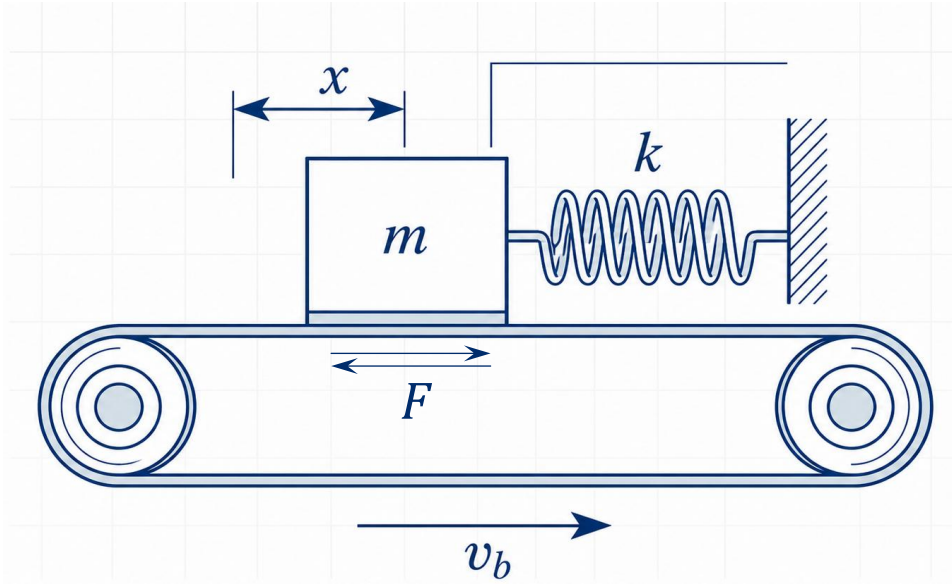
Dry-Friction Model („stick-slip“)

- Stick-Slip Model

$$v_{rel}(t) \coloneqq x_2(t) - v_b$$

$$\dot{x}(t) = \begin{bmatrix} x_2(t) \\ -\frac{k}{m} x_1(t) + \frac{F\left(v_{rel}(t)\right)}{m} \end{bmatrix}$$

$$F\left(v_{\text{rel}}\right) \coloneqq \begin{cases} \dfrac{-F_s}{1+\delta v_{\text{rel}}} & \text{if } v_{\text{rel}} > 0 \quad \text{slip forwards} \\ \dfrac{F_s}{1-\delta v_{\text{rel}}} & \text{if } v_{\text{rel}} < 0 \quad \text{slip backwards} \end{cases}$$



Dry-Friction Model („stick-slip“)

- Stick-Slip Model

$$v_{rel}(t) := x_2(t) - v_b$$

$$\dot{x}(t) = \begin{bmatrix} x_2(t) \\ -\frac{k}{m} x_1(t) + \frac{F(v_{rel}(t))}{m} \end{bmatrix}$$

$$F(v_{\text{rel}}) := \begin{cases} \dfrac{-F_s}{1+\delta v_{\text{rel}}} & \text{if } v_{\text{rel}} > 0 \quad \text{slip forwards} \\ \dfrac{F_s}{1-\delta v_{\text{rel}}} & \text{if } v_{\text{rel}} < 0 \quad \text{slip backwards} \end{cases}$$

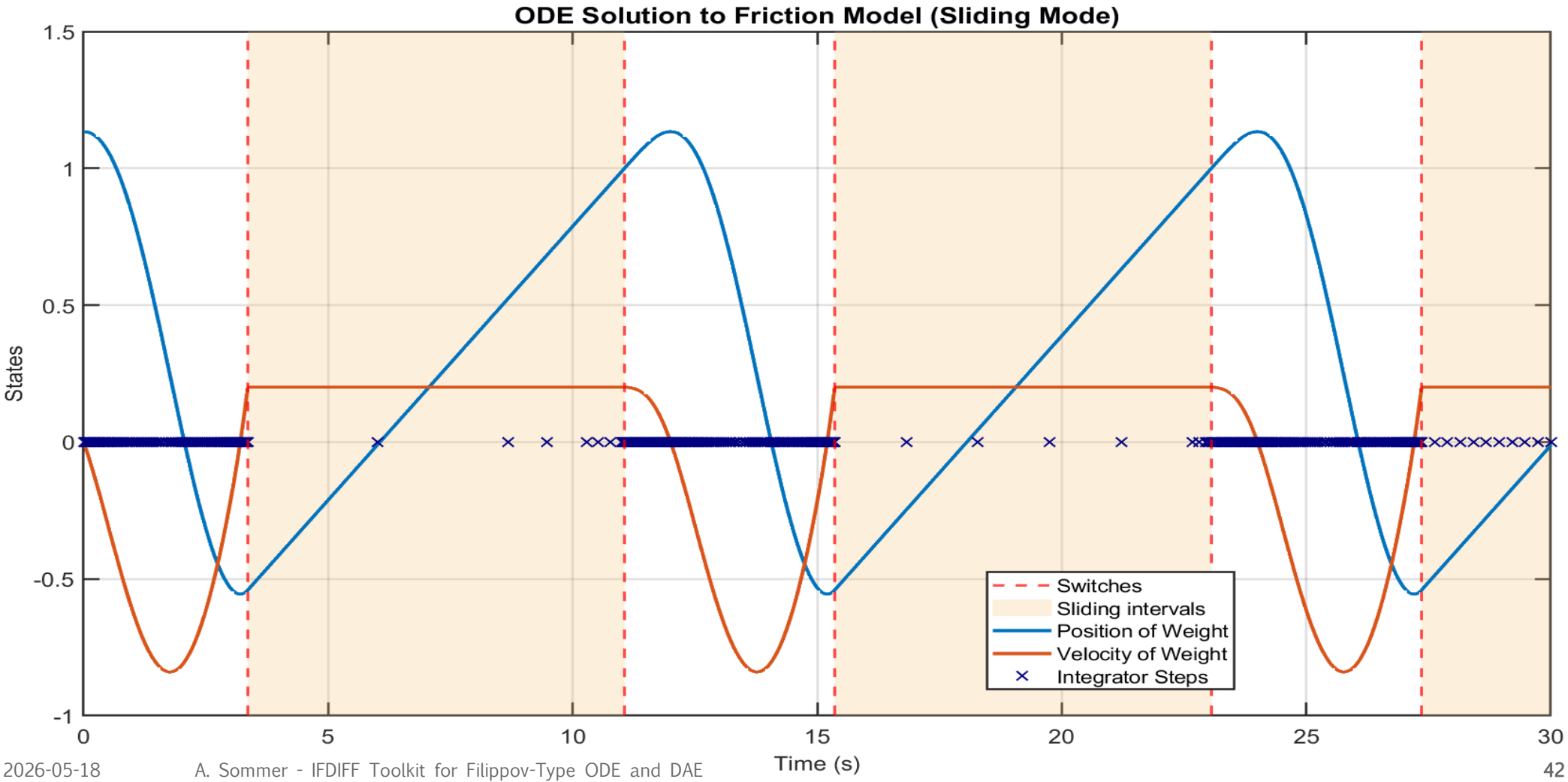
```
function dx = stickslip_rhs(t,x,p)
k  = 1.0;       % N/m  spring constant
m  = 1.0;       % kg   mass
Fs = 0.35;      % N    maximum static friction force
vb = 1;         % m/s  relative belt velocity
vrel = x(2) - vb; % relative speed
dx = zeros(2,1);
dx(1) = x(2);
dx(2) = -k/m * x(1) + stickslip_F(vrel, Fs, k) / m;
end
```

File: stickslip_rhs.m

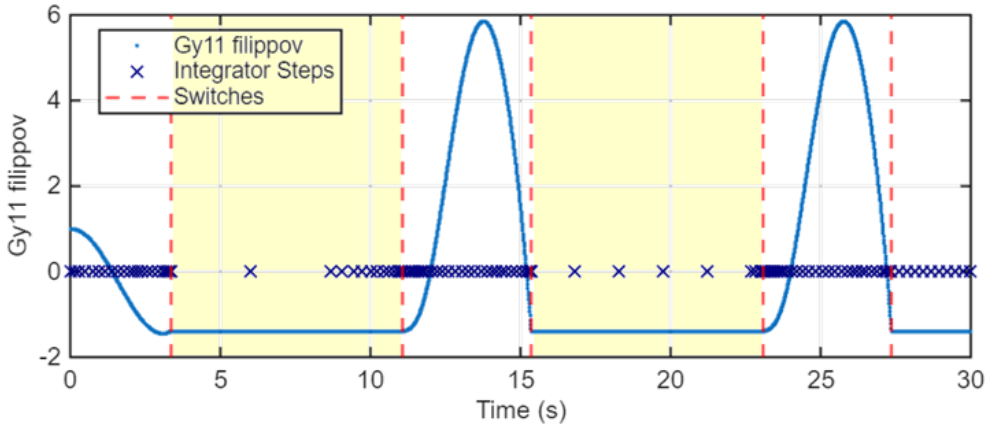
```
function F = stickslip_F(vrel, Fs, k, delta)
  if mu_rel >= 0
     res = -F_s / (1 + delta * mu_rel);
  else
     res =  F_s / (1 - delta * mu_rel);
  end
end
```

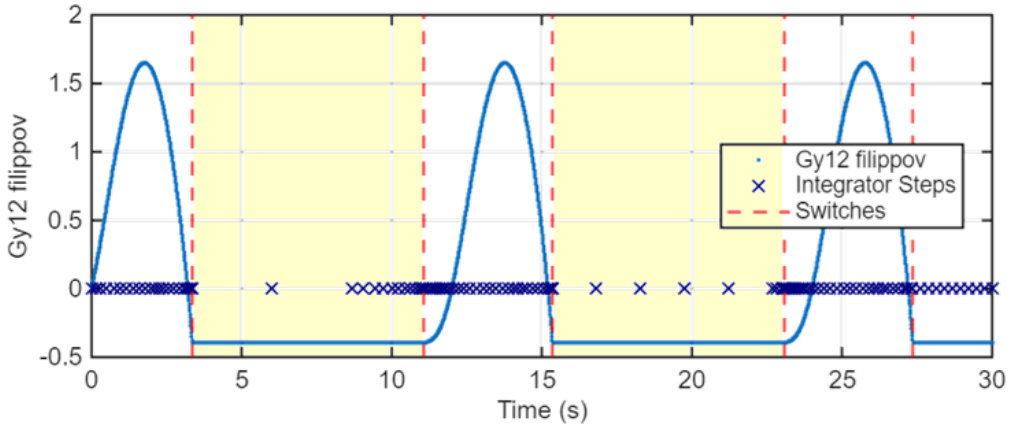
File: stickslip_F.m

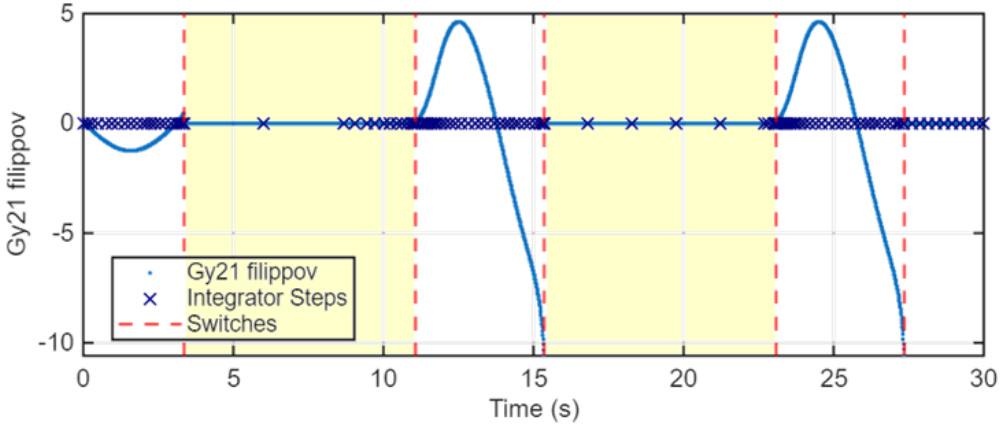
Dry-Friction Model („stick-slip“)

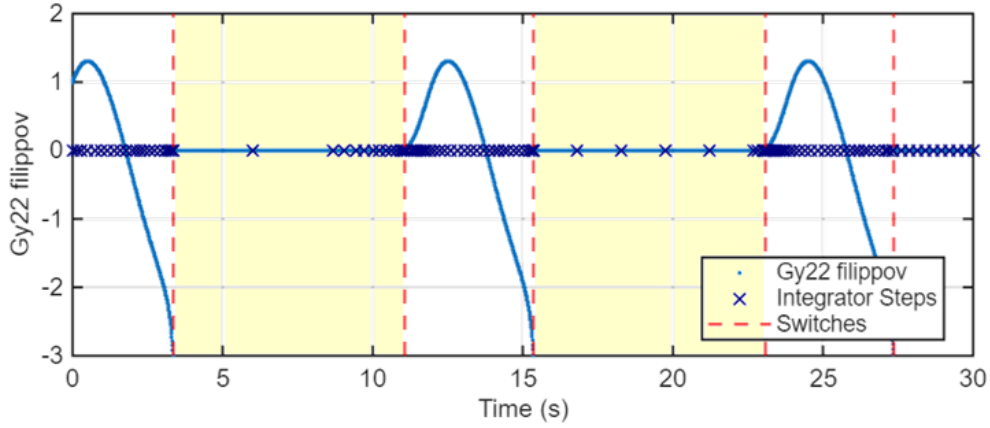


Dry-Friction Model („stick-slip“) - Sensitivities







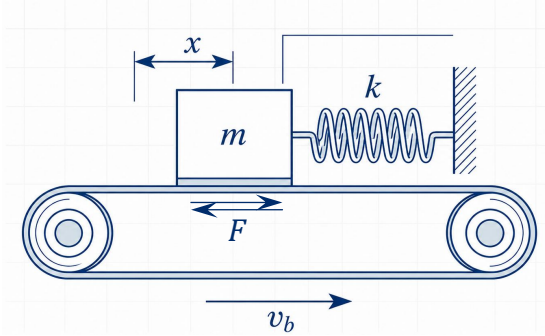


Dry-Friction Model („stick-slip“)

- Stick-Slip Model (Filippov-type)

$$v_{rel}(t) \coloneqq x_2(t) - v_b$$

$$\dot{x}(t) = \begin{bmatrix} x_2(t) \\ -\frac{k}{m} x_1(t) + \frac{F(v_{rel}(t))}{m} \end{bmatrix}$$



$$F(v_{\text{rel}}) \coloneqq \begin{cases} \dfrac{-F_s}{1+\delta v_{\text{rel}}} & \text{if } v_{\text{rel}} > 0 \\ \dfrac{F_s}{1-\delta v_{\text{rel}}} & \text{if } v_{\text{rel}} < 0 \end{cases}$$

- Alternative Non-Filippov formulation

$$\boldsymbol{v}_{\text{rel}}(t) = \boldsymbol{x}_2(t) - v_b$$

$$\dot{\boldsymbol{x}}(t) = \begin{bmatrix} \boldsymbol{x}_2(t) \\ -\frac{k}{m}\boldsymbol{x}_1(t) + \frac{\boldsymbol{F}(\boldsymbol{x}_1(t), \boldsymbol{v}_{\text{rel}}(t))}{m} \end{bmatrix}$$

$$\boldsymbol{F}(x, v_{\text{rel}}) = \begin{cases} \min(|kx|, F_s)\,\operatorname{sgn}(kx), & v_{\text{rel}} = 0 \\ -F_s\, v_{\text{rel}}, & v_{\text{rel}} \neq 0 \end{cases}$$

- 5 conditionals
  → up to $2^5 = 32$ switching functions
- Preformulation possible but tedious!

How to use IFDIFF

- Write code of rhs
- Setup time horizon, initial values, integrator options

```
tspan = [0 20];
x0    = [1 0];
p     = 5.437;
opts  = odeset('reltol', 1e-6);
```

- Single preparation call

```
dhandle = prepareDatahandleForIntegration( ...
              'rhs_f', 'solver', 'ode45', 'options', opts);
```

- Call integrator

```
sol = ode45(@(t,x) rhs_f(t,x,p), tspan, x0, opts);
sol = solveODE(dhandle, tspan, x0, p);
```

- Evaluate and use, plot, etc..

```
T = 0:0.1:10;
X = deval(sol, T);
plot(T,X);
```

```
function dx = rhs_f(t,x,p)
  dx = zeros(2,1);
  dx(1) = 0.01 * t.^2  +  x(2).^3;
  if x(1) < p(1)
     dx(2) = 0;
  else
     if x(1) < p(1) + 0.5
        dx(2) = 5;
     else
        dx(2) = 0;
     end
  end
end
```

File: rhs_f.m

- IFDIFF:
  - Unmodified rhs code
  - Single preparation call (only once for rhs file)
  - Transparent usage of Matlab's sol structure (no further code modifications necessary)

Summary

- **Ignoring switches** leads to **poor integration** and **useless sensitivities – Unnoticed disasters ahead!**

- **IFDIFF** is an **automated Matlab toolkit** for **multi-dimensional ODE and DAE with state-dependent switches** providing …
- … **accurate switching point detection**
- … **accurate integration**
- … **accurate forward sensitivities** for states and parameters
- … automatic **Filippov sliding mode** detection and handling
- … and **integrates seamlessly** in Matlab's ode solver suite (`ode23`, `ode45`, `ode15s`, …) delivering a `sol` structure compatible to Matlab's `deval`

View Project on GitHub

`https://ifdiff.github.io`

THANKS TO FUNDERS …

VECTOR STIFTUNG

SPONSORED BY THE Federal Ministry of Education and Research

05M2022 - MORFAE

… AND COLLABORATORS

Kilian Folger
Lev Gromov
Marvin Hertweck
Pilar Keller
Tobias Reith
Ilona Slutsker
Michael Strik
Valentin von Trotha
Leonard Wirsching

Hans Georg Bock
Ekaterina Kostina

VECTOR STIFTUNG

IFDIFF is financially supported by Vector Stiftung (Project: PErFDiff)

HEIDELBERG Chapter of siam

HGS MathComp

IWR Interdisciplinary Center for Scientific Computing

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

Filippov-ODEs and DAEs with State-dependent Switches

Theory and Hands-On Practical with IFDIFF

Thursday, July 09, 2026 • 9-16h

Mathematikon • CIP-Pool • 3rd Floor and Online

Register here!

https://t1p.de/ifdiff2026

State and Parameter Sensitivities (without Switches)

Forward Sensitivities of ODEs (without switches)

- Parametrized initial value problem: $\dot{x}(t) = f(t, x(t), p), \quad x(t_0) = x_0$ Solution: $x(t; t_0, x_0, p)$
- State and parameter sensitivities: $G_{x_0}(t) := \frac{dx}{dx_0}(t; t_0, x_0, p)$ and $G_p(t) := \frac{dx}{dp}(t; t_0, x_0, p)$
- Variational Differential Equations (VDE): $\dot{G}_{x_0}(t) = \frac{df}{dx}(t, x(t), p) \cdot G_{x_0}(t) \qquad G_{x_0}(t_0) = I$ (unit matrix)

- How **NOT** to compute:
  - FD / External Numerical Differentiation (END): $\frac{dx}{dx_{0,i}}(t) \approx \frac{x(t; t_0, x_0, p) - x(t; t_0, x_0 + he_i, p)}{h}$ — different functions due to integrator adaptivity (e.g. step sizes); e_i: i-th unit vector
    very problematic, due to adaptive components
  - AD tools on whole integration scheme: problematic also due to adaptive components in integrator

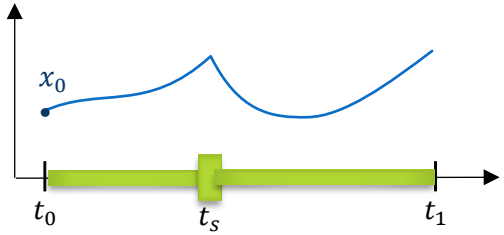
- How to compute
  - Internal Numerical Differentiation (IND): freezing adaptive components (requires adjustment to integrator)
  - Solving VDE with numerical integrator: error controlled (requires continuous representation of solution y)

Sensitivity Propagation across Switches

Propagation of Sensitivities across Switches

- Parametrized initial value problem: $\dot{x}(t) = f(t, x(t), p), \quad x(t_0) = x_0(p)$
- State sensitivities: $G_{x_0}(t) := \frac{dx}{dx_0}(t; t_0, x_0, p)$
- Assume a single switch: $t_s \in (t_0, t_1)$

Solution: $x(t; t_0, x_0, p)$



- Integrator generates solution:

$$x_n = \left\{\begin{matrix}\text{integration}\\ \text{from } t_s \text{ to } t_1\end{matrix}\right\} \circ \left\{\begin{matrix}\text{switch handling at } t_s\\ x_-(t_s) \mapsto x_+(t_s)\end{matrix}\right\} \circ \left\{\begin{matrix}\text{integration}\\ \text{from } t_0 \text{ to } t_s\end{matrix}\right\} (x_0)$$

- Chain rule yields for the sensitivities:

$$G_{x_0}(t_1; t_0, x_0, p) = G_{x_0}(t_1; t_s, x_+(t_s), p) \cdot U \cdot G_{x_0}(t_s; t_0, x_0, p)$$

Note: t_s depends on x_0 and p
$\Longrightarrow$ implicit function theorem on $\sigma(t_s, x, p) = 0$

- Sensitivity update: $U = I + \frac{\partial \Delta}{\partial x_-} + \left(f_+ - f_- - \frac{\partial \Delta}{\partial t_s} - \frac{\partial \Delta}{\partial x_-} f_-\right) \cdot \frac{\frac{\partial \sigma_i}{\partial x_-}}{\frac{d\sigma_i}{dt_s}}$

$\Longrightarrow$ correct sensitivities **require** switching function derivatives

Full 1st Order Sensitivity Updates

- State sensitivities G_{x_0} and parameter sensitivities G_p for switches in RHS and states:

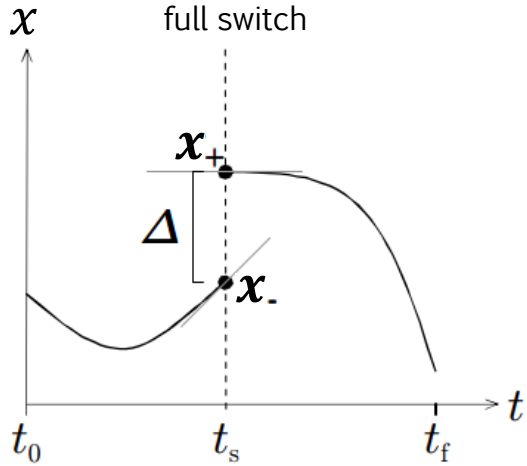
$$G_{x_0}(t_1; t_0, x_0, p) = G_{x_0}(t_1; t_s, x_+(t_s), p) \cdot U_{x_0} \cdot G_{x_0}(t_s; t_0, x_0, p)$$
$$G_{x_0}(t_0; t_0, x_0, p) = I$$

$$G_p(t_1; t_0, x_0, p) = G_{x_0}(t_1; t_s, x_+(t_s), p) \cdot \left(U_{x_0} G_p(t_s; t_0, x_0, p) + U_p\right) \cdot G_{x_0}(t_s; t_0, x_0, p) + G_p(t_1; t_s, x_+(t_s), p)$$
$$G_p(t_0; t_0, x_0, p) = \frac{\partial}{\partial p} x_0(p)$$

- Sensitivity updates:

$$U_{x_0} = I + \frac{\partial \Delta_i}{\partial x_-} + \left(f_+ - f_- - \frac{\partial \Delta_i}{\partial t_s} - \frac{\partial \Delta_i}{\partial x_-} f_-\right) \cdot \frac{\frac{\partial \sigma_i}{\partial x_-}}{\frac{d\sigma_i}{dt_s}}$$

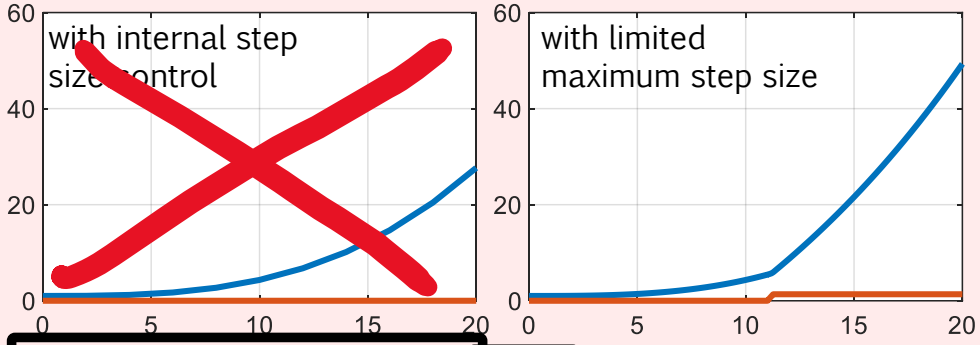
$$U_p = \frac{\partial \Delta_i}{\partial p} + \left(f_+ - f_- - \frac{\partial \Delta_i}{\partial t_s} - \frac{\partial \Delta_i}{\partial p} f_-\right) \cdot \frac{\frac{\partial \sigma_i}{\partial p}}{\frac{d\sigma_i}{dt_s}}$$

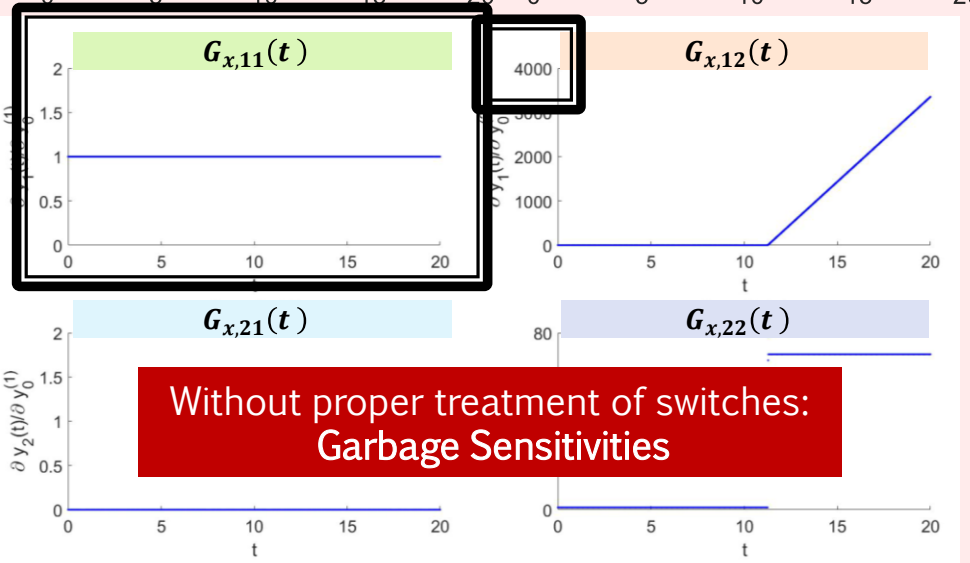


$$x_- = \lim_{\substack{\varepsilon \to 0 \\ \varepsilon < 0}} x(t_s + \varepsilon; t_0, x_0, p)$$
$$x_+ = \lim_{\substack{\varepsilon \to 0 \\ \varepsilon > 0}} x(t_s + \varepsilon; x_-, \Delta, p)$$
$$f_- = \lim_{\substack{\varepsilon \to 0 \\ \varepsilon < 0}} f(t_s + \varepsilon, x_-, p)$$
$$f_+ = \lim_{\substack{\varepsilon \to 0 \\ \varepsilon > 0}} f(t_s + \varepsilon, x_+, p)$$
$$\Delta\colon [t_0, t_1] \times \mathbb{R}^{n_x} \times \mathcal{P} \to \mathbb{R}^{n_x}$$
$$(t, x_-(t), p) \mapsto x_+(t)$$

Canonical Example: Sensitivity Accuracy

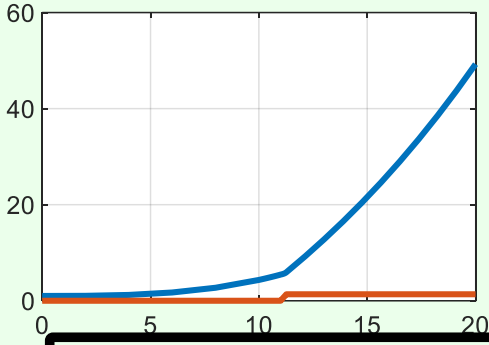
Naïve approach with ode45 fails



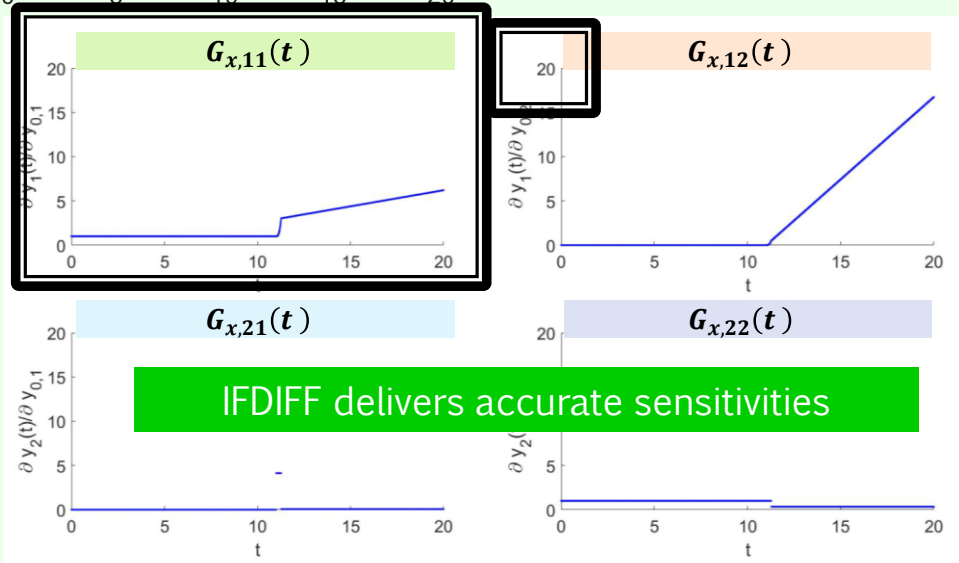


Without proper treatment of switches:
Garbage Sensitivities

IFDIFF with ode45 succeeds



Comparison to analytical sol'n
IFDIFF maximum error in …
… trajectory (absolute): $< 10^{-8}$
… sensitivity (relative): $< 10^{-5}$
… switch time #1 (abs.): $< 10^{-14}$
… switch time #2 (abs.): $< 10^{-11}$



IFDIFF delivers accurate sensitivities