

Supplementary Material for “Identifiability and Observability of Nonsmooth Systems via Taylor-Like Approximations”

Peter Stechlinski, Sameh Eisa, and Hesham Abdelfattah

I. INTRODUCTION

This document serves to supplement [1]. In particular, we present the MATLAB code used to produce the results in the aforementioned paper.

II. CALCULATING LD-DERIVATIVES OF NONSMOOTH ELEMENTARY FUNCTIONS

In this part we provide code to calculate LD-derivatives [2], [3] of the abs-value and max function, for an arbitrary directions matrix. Note that implementations of nonsmooth automatic differentiation methods in Julia for calculating LD-derivatives is available at <https://github.com/kamilkhanlab/nonsmooth-forward-ad.git> (see [4]).

A. LD-Derivative of Absolute-Value Function:

Given any directions matrix $\mathbf{M} = [m_1 \ m_2 \ \dots \ m_k] \in \mathbb{R}^{1 \times k}$, the LD-derivative of $\text{abs}(x) = |x|$ at $x_0 \in \mathbb{R}$ in the directions \mathbf{M} is given by

$$\text{abs}'(x_0; \mathbf{M}) = \text{fsign}(x_0, m_1, m_2, \dots, m_k) \mathbf{M} \in \mathbb{R}^{1 \times k},$$

where the first-sign function returns the sign of the first nonzero element, or zero if its input is the zero vector:

$$\text{fsign}(x_1, \dots, x_n) := \begin{cases} 1, & \text{if } x_j > 0, j = \min\{i : x_i \neq 0\}, \\ -1, & \text{if } x_j < 0, j = \min\{i : x_i \neq 0\}, \\ 0, & \text{if } \mathbf{x} = \mathbf{0}_n. \end{cases}$$

The following MATLAB function (see the file `abs_LD.m`) takes the domain point \mathbf{x}_0 and the directions matrix \mathbf{M} as its input. The function returns the LD-derivative $\text{abs}'(x_0; \mathbf{M})$.

```
function abs_LD = abs_LD(x,M)
% % M should be row vector of size 1xk
column_count = size(M,2);
first_non_zero = 0;
if ne(sum(x), 0)
    first_non_zero = sign(x);
else
    for k=1:column_count
        S = sum(M(:,k));
        if ne(S,0)
            first_non_zero = sign(M(:,k));
            break
        end
    end
end
fsign = first_non_zero';
abs_LD = fsign*M;
end
```

B. LD-Derivative of Maximum Function:

Given any directions matrix

$$\mathbf{M} = \begin{bmatrix} \mathbf{M}_1 \\ \mathbf{M}_2 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & \cdots & m_{1k} \\ m_{21} & m_{22} & \cdots & m_{2k} \end{bmatrix} \in \mathbb{R}^{2 \times k},$$

the LD-derivative of $\max(x, y)$ at $(x_0, y_0) \in \mathbb{R}^2$ in the directions \mathbf{M} is given by

$$\begin{aligned} \max'(x_0, y_0; \mathbf{M}) &= \max' \left(\begin{bmatrix} x_0 \\ y_0 \end{bmatrix}; \begin{bmatrix} m_{11} & m_{12} & \cdots & m_{1k} \\ m_{21} & m_{22} & \cdots & m_{2k} \end{bmatrix} \right) \\ &= \mathbf{smax}((x_0, m_{11}, \dots, m_{1k}), (y_0, m_{21}, \dots, m_{2k})) \\ &:= \begin{cases} \mathbf{M}_1, & \text{if } \text{fsign}((x_0, m_{11}, \dots, m_{1k}) - (y_0, m_{21}, \dots, m_{2k})) \geq 0, \\ \mathbf{M}_2, & \text{otherwise,} \end{cases} \end{aligned}$$

where the shifted-lexicographic maximum function $\mathbf{smax} : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}^{n-1}$ returns the lexicographic maximum of the two vectors, left-shifted by one element. That is,

$$\mathbf{smax}(\mathbf{x}, \mathbf{y}) := \text{lshift}(\mathbf{lmax}(\mathbf{x}, \mathbf{y})),$$

where

$$\begin{aligned} \mathbf{lmax}(\mathbf{x}, \mathbf{y}) &:= \begin{cases} \mathbf{x}, & \text{if } \text{fsign}(\mathbf{x} - \mathbf{y}) \geq 0, \\ \mathbf{y}, & \text{if } \text{fsign}(\mathbf{x} - \mathbf{y}) < 0, \end{cases} \\ \text{lshift}(x_1, x_2, \dots, x_n) &:= (x_2, \dots, x_n). \end{aligned}$$

The following MATLAB function (see the file SLmax.m) takes the input $\mathbf{xM1} = [x_0 \ \mathbf{M}_1] \in \mathbb{R}^{1 \times (k+1)}$ and $\mathbf{yM2} = [y_0 \ \mathbf{M}_2] \in \mathbb{R}^{1 \times (k+1)}$, i.e., the domain point (x_0, y_0) and the directions matrix $\mathbf{M} \in \mathbb{R}^{2 \times k}$ stacked together. The function returns the LD-derivative $\max'(x_0, y_0; \mathbf{M})$.

```
function SLmax_output = SLmax(xM1, yM2)
% Shifted Lmax
% Returns the lexicographic maximum of 2 vectors xM1&yM2, left-shifted by
% one element
% Vectors xM1&yM2 must have the same size
SLmax_output = xM1(2:end);
for k=1:size(xM1, 2)
    if xM1(k) > yM2(k)
        SLmax_output = xM1(2:end);
        break
    elseif xM1(k) < yM2(k)
        SLmax_output = yM2(2:end);
        break
    end
end
end
```

III. CODE FOR APPLYING L-SERC ALGORITHM IN EXAMPLE III.10 IN [1]

The following MATLAB code (see the file Example_4.7.m) applies the full L-SERC algorithm (Algorithm 1 in [1]) to Example III.10 in [1]. Namely, we consider the following system:

$$\begin{aligned} \dot{x}(t) &= \max(0, 1 - e^{-(x(t) - \theta_1)}), \quad x(0) = \theta_2, \\ y(t) &= x(t), \end{aligned} \tag{1}$$

where $\boldsymbol{\theta} = (\theta_1, \theta_2) \in \Theta = \mathbb{R}^2$ are the parameters and $\boldsymbol{\theta}^* = (1, 1)$ are the reference parameter values. Note that the ODE system in (1) admits the following closed-form solution:

$$x(t; \boldsymbol{\theta}) = \begin{cases} \ln(e^t + e^{\theta_1 - \theta_2} - e^{t + \theta_1 - \theta_2}) + \theta_2, & \text{if } \theta_1 \leq \theta_2, \\ \theta_2, & \text{if } \theta_1 > \theta_2. \end{cases}$$

Thus, we can see from the closed-form solution that (1) is partially identifiable at $\boldsymbol{\theta}^*$ with $V = \{\boldsymbol{\theta} : \theta_1 \leq \theta_2\}$ and unidentifiable otherwise as θ_1 cannot be determined from the output in the second case.

The nonsmooth sensitivity system for (1) is given by

$$\begin{aligned}\dot{\mathbf{X}}(t) &= \text{slmax} \left(\begin{bmatrix} 0 & \mathbf{0}_{1 \times 3} \end{bmatrix}, [1 - e^{-(x^*(t) - \theta_1^*)} \quad e^{-(x^*(t) - \theta_1^*)} (\mathbf{X}(t) - [d_1 \quad 1 \quad 0])] \right), \\ \mathbf{Y}(t) &= \mathbf{X}(t), \\ \mathbf{X}(0) &= [d_2 \quad 0 \quad 1],\end{aligned}\tag{2}$$

where $x^*(t)$ is the reference solution of (1) when $\theta = \theta^*$. Then (1) and (2) can be numerically solved simultaneously on $[0, t_f]$ to furnish \mathbf{x}^* , \mathbf{y}^* , \mathbf{X}^* , \mathbf{Y}^* , and therefore Υ_d in the L-SERC algorithm by sampling $\mathbf{Y}^*(t_k)$. (For more details, see Example III.10 in [1].)

The MATLAB code is given as follows, which requires initializations for

- $p1 = \theta_1^*$, $p2 = \theta_2^*$ (parameter reference values),
- $\mathbf{P_Matrix_set} = D$ (the set of primary probing directions \mathbf{d}_i in Algorithm 1),
- $\text{epsilon_twin_probing} = \epsilon_{\text{twin}} > 0$, $\text{Matrix_twin_probing_set} = D_{\text{twin}}$ (i.e., the set of twin directions \mathbf{e}_j in Algorithm 1),
- $\text{epsilon_sing_probing} = \epsilon_{\text{twin}} > 0$ and $q = 1$ (the singular probing stage is executed one time).

The function outputs the L-SERC test answer, i.e., ‘Partially identifiable’ or ‘non-identifiable’.

```
clear all
clc
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Initialization %%%%%%%%%%
% Number of parameters (theta) of the system
np = 2;
% Values of reference parameters
Param = zeros(np,1);
Param(1) = 1; %p1
Param(2) = 1; %p2
% Number of states of the system
n = 1;
% Initial conditions
x0 = Param(2);
% Primary probing directions within the directions matrix P
P_Matrix_set = {[[1 0]', eye(2)], [[-1 0]', eye(2)]...
                , [[0 1]', eye(2)], [[0 -1]', eye(2)]];
% Value of epsilon for twin probing (epsilon_twin)
epsilon_twin_probing = 0.01;
% Directions of perturbations of epsilon_twin
Matrix_twin_probing_set = {[[1 0]', zeros(2)], [[0 1]', zeros(2)]};
% Number of iterations for the singularity probing stage
q = 1;
% Value of epsilon for singular probing(epsilon_sing)
epsilon_sing_probing = 0.01;
D_sing=[];
V_sing=[];

T_f=[];
T_P_Matrix_f=[];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Primary & twin Probing stages
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for k = 1:length(P_Matrix_set)
    for k_twin_prob = 1:length(Matrix_twin_probing_set)
        P_Matrix = cell2mat(P_Matrix_set(k)); %Get current directions matrix
        P_Matrix_twin_probing = cell2mat(Matrix_twin_probing_set(k_twin_prob)); %Get
            current epsilon_twin
        if abs(P_Matrix(:,1)) == P_Matrix_twin_probing(:,1)
            continue
        end
        P_Matrix_twin_probing(:,1) = P_Matrix_twin_probing(:,1).*epsilon_twin_probing;
        if sum(sign(P_Matrix(:,1))) == 1 %check the sign of the sum of elements of the
            first column of P
```

```

    P_Matrix_twin_probing = P_Matrix_twin_probing+P_Matrix;
elseif sum(sign(P_Matrix(:,1))) == -1
    P_Matrix_twin_probing = -P_Matrix_twin_probing+P_Matrix;
end
%start solving the forward sensitivity system for every direction in
%primary and twin probing directions matrices
xp0 = [P_Matrix(2,1) P_Matrix(2,2) P_Matrix(2,3)]; %dx/dp initial conditions
xp0_twin_probing = [P_Matrix_twin_probing(2,1) P_Matrix_twin_probing(2,2)
    P_Matrix_twin_probing(2,3)]; %dx/dp initial conditions
X0 =[x0;xp0(:)]; %vector of states & state sensitivities to be used in ODE solver
X0_twin_probing =[x0;xp0_twin_probing(:)]; %vector of states & state sensitivities
    to be used in ODE solver
% Time interval/steps
N = 2*np;
tspan = 0:1/(N):1;
[Time,X] = ode45(@(t,X) Example_4_7_ODE_solve(t,X,Param,P_Matrix),tspan,X0(:));
[T_twin_probing,X_twin_probing] = ode45(@(t_twin_probing,X_twin_probing)
    Example_4_7_ODE_solve(t_twin_probing,X_twin_probing,Param,
    P_Matrix_twin_probing),tspan,X0_twin_probing(:));
StepCount = length(Time);

% Calculate the the SVD for the LSERC matrix to check identifiability
h = X(:,1);
[m,dummy1]=size(h);
dhdx = 1;
Yp = zeros(StepCount,np);
LD_Y_theta = zeros(StepCount,np+1);
LD_Y_theta_twin_probing = zeros(StepCount,np+1);
t=tspan(StepCount);
step = 1;
p1 = Param(1);
p2 = Param(2);
for ti=1:StepCount
    xp = [X(ti,2) X(ti,3) X(ti,4)];
    xp_twin_probing = [X_twin_probing(ti,2) X_twin_probing(ti,3) X_twin_probing(
        ti,4)];
    dhdp = xp;
    dhdp_twin_probing = xp_twin_probing;
    LD_Y_theta(step,:) = dhdx * xp ;
    LD_Y_theta_twin_probing(step,:) = dhdx * xp_twin_probing ;
    step = step+1;
end
solution = X(:,1);
[U_LD,S_LD,V_LD] = svd(LD_Y_theta);
[U_LD_twin_probing,S_LD_twin_probing,V_LD_twin_probing] = svd(
    LD_Y_theta_twin_probing);

L_Y_theta = LD_Y_theta/P_Matrix;
L_Y_theta_twin_probing = LD_Y_theta_twin_probing/P_Matrix_twin_probing;
[U_L,S_L,V_L] = svd(L_Y_theta);
[U_L_twin_probing,S_L_twin_probing,V_L_twin_probing] = svd(L_Y_theta_twin_probing
);
if q>0
    %Save a record of the directions that resulted in a rank deficient LSERC
    %matrix
    if rank(S_L)<np
        zero_vector_index = find(all(S_L==0));
        d_sing = P_Matrix(:,1);

```

```

        v_sing = V_L(:,zero_vector_index);
        D_sing = [D_sing d_sing];
        V_sing = [V_sing v_sing];
    end
end
%Collect results in tables for easier access
T = array2table(LD_Y_theta);
T_P_Matrix = array2table(P_Matrix);
T.LD_Y_theta_twin_probing = LD_Y_theta_twin_probing;
T.L_Y_theta = L_Y_theta;
T.L_Y_theta_twin_probing = L_Y_theta_twin_probing;
T.S_LD = S_LD;
T.S_LD_twin_probing = S_LD_twin_probing;
T.S_L = S_L;
T.S_L_twin_probing = S_L_twin_probing;
T.solution = solution;
T_f=[T_f;T];
T_P_Matrix_f = [T_P_Matrix_f;T_P_Matrix];
end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Singularity probing stage
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
while q>0
    check_empty=isempty(V_sing);
    if ne(check_empty,1)
        epsilon_sing_probing = 0.01;
        Param_sing = [Param+epsilon_sing_probing*V_sing(:,1)];
        D_sing = unique(D_sing.', 'rows').';
        Matrix_sing_probing_set = {[D_sing(:,1),eye(2)]}; %construct the singularity
        probing directions matrix from directions saved in the previous stages
        for i=2:length(D_sing)
            Matrix_sing_probing_set{end+1} = [D_sing(:,i),eye(2)];
            Param_sing = [Param_sing Param+epsilon_sing_probing*V_sing(:,i) Param-
                epsilon_sing_probing*V_sing(:,i)];
        end
        Param_sing = unique(Param_sing.', 'rows').';

        T_f_sing=[];
        for k = 1:length(Matrix_sing_probing_set)
            for i=1:length(Param_sing)
                P_Matrix_sing = cell2mat(Matrix_sing_probing_set(k));
                %start solving the forward sensitivity system for every direction in
                %the singularity probing directions matrix
                Param = Param_sing(:,i);
                xp0_sing = [P_Matrix_sing(2,1) P_Matrix_sing(2,2) P_Matrix_sing(2,3)]; %
                dx/dp initial conditions
                X0_sing=[x0;xp0_sing(:)]; %vector of states & state sensitivities to be
                used in ODE solver
                % Time interval/steps
                N = 2*np;
                tspan = 0:1/(N):1;
                [T_sing,X_sing] = ode45(@(t_sing,X_sing) Example_4_7_ODE_solve(t_sing,
                    X_sing,Param,P_Matrix_sing),tspan,X0_sing(:));
                StepCount = length(T_sing);
                % Calculate the the SVD for the LSERC matrix to check identifiability
                h = X(:,1);
                [m,dummy1]=size(h);
                dhdx = 1;
            end
        end
    end
    q=q-1;
end

```

```

        Yp = zeros(StepCount,np);
        LD_Y_theta_sing = zeros(StepCount,np+1);
        t=tspan(StepCount);
        step = 1;
        p1 = Param(1);
        p2 = Param(2);
        for ti=1:StepCount
            xp_sing = [X_sing(ti,2) X_sing(ti,3) X_sing(ti,4)];
            dhdp = xp_sing;
            LD_Y_theta_sing(step,:) = dhdx * xp_sing ;
            step = step+1;
        end
        solution_sing = X_sing(:,1);
        [U_LD_sing,S_LD_sing,V_LD_sing] = svd(LD_Y_theta_sing);
        L_Y_theta_sing = LD_Y_theta_sing/P_Matrix_sing;
        [U_L_sing,S_L_sing,V_L_sing] = svd(L_Y_theta_sing);
        %Collect results in tables for easier access
        T = array2table(LD_Y_theta_sing);
        T.L_Y_theta_sing = L_Y_theta_sing;
        T.S_LD_sing = S_LD_sing;
        T.S_L_sing = S_L_sing;
        T.solution_sing = solution_sing;
        T_f_sing=[T_f_sing;T];
    end
end
end
q = q-1;
end
if (rank(S_L)==size(S_L,2)) || (rank(S_L_sing)==size(S_L_sing,2))
    disp('Partially identifiable');
else
    disp('Non-identifiable');
end
function dXdt = Example_4_7_ODE_solve(t,X,Param,P_Matrix)
syms x p1 p2 P11 P12 P13 P21 P22 P23 Sx1 Sx2

% ODE system1
f = 0;
g = 1-exp(-1*(x-p1));

Jxf = jacobian(f,[p1,p2]);
Jyf = jacobian(f,x);
Jxg = jacobian(g,[p1,p2]);
Jyg = jacobian(g,x);
P = [P11 P12 P13;P21 P22 P23];

% x value
x = X(1);
% Parameters values
p1 = Param(1);
p2 = Param(2);

% Substitute x, p1 and p2 values into the functions
f_val = subs(f);
g_val = subs(g);

dxdt = max(f_val,g_val);
% Substitute x, p1 and p2 values into x_dot

```

```

dxdt_val = double(subs(dxdt));
% Substitute x, p1 and p2 values into the jacobians
Jxf_val = subs(Jxf);
Jyf_val = subs(Jyf);
Jxg_val = subs(Jxg);
Jyg_val = subs(Jyg);

X_1 = X(2);
X_2 = X(3);
X_3 = X(4);
N = [X_1 X_2 X_3];

Slmax_input_sym = [f_val Jxf_val*P_Matrix+Jyf_val*N;g_val Jxg_val*P_Matrix+Jyg_val*N
    ];

Slmax_input_val = subs(Slmax_input_sym);
Slmax_input = double(Slmax_input_val);

Slmax_output = SLmax(Slmax_input(1,:),Slmax_input(2,:));

% ODE system2
dxpdt1 = Slmax_output(1);
dxpdt2 = Slmax_output(2);
dxpdt3 = Slmax_output(3);

dxpdt = [dxpdt1;dxpdt2;dxpdt3];
dXdt = [dxdt_val;dxpdt];
end

```

IV. CODE FOR APPLYING L-SERC ALGORITHM TO STOMMEL-BOX CLIMATE MODEL IN [1]

The following MATLAB code (see the file `Stommel_Box_identifiability_smooth_heatmap.m`) applies the L-SERC algorithm (Algorithm 1 in [1]) to the nonsmooth Stommel-box climate model in [1]. The nonsmooth Stommel-Box Climate model is given as follows:

$$\dot{T} = \theta_1 + u_1 - T - T|T - V|, \quad T(0) = T_0, \quad (3a)$$

$$\dot{V} = \theta_2 + u_2 - V\theta_3 - V|T - V|, \quad V(0) = V_0, \quad (3b)$$

$$y = \max(T, T_{\min}), \quad (3c)$$

where T and V represent the dimensionless temperature and salinity, t is the dimensionless time, the parameters θ_1 , θ_2 and θ_3 represent the strength of the thermal forcing, the strength of the freshwater forcing, and the ratio of the thermal and freshwater surface restoring time scales, respectively. We assume the output is a temperature measurement, and that T_{\min} is the lowest measurable temperature due to sensory, equipment, and/or computational limitations. The inputs u_1 and u_2 represent variations in parameters from their baseline values θ_1 and θ_2 ; following [5], let $u_1(t) = B \sin(\Omega t)$, $u_2(t) = \hat{B} \sin(\Omega t)$, where B , \hat{B} , Ω are constants.

We consider initial conditions $T_0 = 1, V_0 = 2$, reference control inputs $\mathbf{u}^*(t) = (u_1^*(t), u_2^*(t)) = (2 \sin(20t), \sin(20t))$, reference parameters $\boldsymbol{\theta}^* = (\theta_1^*, \theta_2^*, \theta_3^*) = (3, 1.1, 0.3)$, and $T_{\min} = 1.5$. The nonsmooth sensitivity system for the Stommel-box model in (3) is given by

$$\begin{aligned}
 \dot{\mathbf{X}}_1 &= [d_1 \quad 1 \quad 0 \quad 0] - (1 + |T^* - V^*|)\mathbf{X}_1 - T^* \text{fsign}(T^* - V^*, \mathbf{X}_1 - \mathbf{X}_2) (\mathbf{X}_1 - \mathbf{X}_2), \\
 \dot{\mathbf{X}}_2 &= [d_2 - V^* d_3 \quad 0 \quad 1 \quad -1] - (\theta_3^* + |T^* - V^*|)\mathbf{X}_2 - V^* \text{fsign}(T^* - V^*, \mathbf{X}_1 - \mathbf{X}_2) (\mathbf{X}_1 - \mathbf{X}_2), \\
 \mathbf{Y} &= \text{slmax}([T^* \quad \mathbf{X}_1], [T_{\min} \quad \mathbf{0}_{1 \times 4}]) \\
 \mathbf{X}(0) &= \mathbf{0}_{2 \times 4},
 \end{aligned}$$

where $\mathbf{X}_i(t) := \text{row}_i(\mathbf{X}(t)) \in \mathbb{R}^{1 \times 4}$. (For more details, see Section V in [1].)

The MATLAB code is given as follows, which requires initializations for

- $p1 = \theta_1^*$, $p2 = \theta_2^*$ and $p3 = \theta_3^*$ (parameter reference values),
- B, \hat{B}, A (input reference values),

- T_0, V_0 (initial values of states),
- $\text{P_Matrix_set} = D$ (the set of primary probing directions \mathbf{d}_i in Algorithm 1).

Note that $\text{epsilon_twin_probing} = \epsilon_{\text{twin}} = 0$ (twin probing stage is turned off) and $\text{epsilon_sing_probing} = \epsilon_{\text{sing}} = 0$ and $q = 0$ (singular probing stage is turned off). The function produces figures of: (i) state and output numerical solutions (ii) output L-sensitivity functions (i.e., \mathbf{S}_y^L) (iii) heatmap comparing smoothing SERC approach with nonsmooth L-SERC approach.

```
clear
clc
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Initialization section
% Number of parameters (theta) of the system
ThetaCount = 3;
% Values of reference parameters
Param = zeros(ThetaCount,1);
Param(1) = 3; %theta1
Param(2) = 1.1; %theta2
Param(3) = 0.3; %theta3
% Number of states of the system
n = 2;
% Initial conditions
T_0=1;
V_0=2;
x0 = [T_0;V_0];
% Reference input values
B = 2;
B_hat = 1;
A = B - B_hat;
omega = A/0.05;
u1_ref = B*sin(omega*t0);
u2_ref = B_hat*sin(omega*t0);
%Primary directions matrix
M_Matrix = [[1 0 0]',eye(3)]; %M=[e_1 I_3]
% Value of epsilon for twin probing (epsilon_twin), turned off in this case
epsilon_twin_probing = 0;
% Number of iterations for the singularity probing stage
q = 0;
% Value of epsilon for singular probing(epsilon_sing)
epsilon_sing_probing = 0;
X0 = zeros(2,4);

%vector of states & state sensitivities to be used in ODE solver
X0_All =[x0;X0(:)];

% Time interval/steps
N = 2*ThetaCount;
multiplier = 1000;
N = N*multiplier;
tspan = 0:1/(N):1;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% %
% Nonsmooth section
%Solving LD-sensistivity matrix
[Time,X] = ode45(@(t,X) Nonsmooth_Stommel_Box_model_ODE(t,X,Param,M_Matrix),tspan,
    X0_All(:));

% Ytheta identifiability check matrix
%h=max
m=1;
```



```

[Time_smooth,X_smooth] = ode45(@(t,X) Stommel_Box_model_ODE_smoothing(t,X,Param,
    M_Matrix,alpha1),tspan,X0_All(:));

% Ytheta identifiability check matrix
%h=max
m=1;
StepCount_smooth = length(Time_smooth);
for jj = 1:total_samples_smooth
    number_of_samples_smooth = jj;
    time_sample_increment_smooth = (length(Time_smooth)-1)/total_samples_smooth;
    t_samples_smooth = [];
    for s=1:number_of_samples_smooth
        t_samples_smooth = [t_samples_smooth 1+s*time_sample_increment_smooth];
    end

    t_samples_smooth=sort(t_samples_smooth);
    t_samples_smooth = ceil(t_samples_smooth);
    Y_theta_case_smooth = zeros(StepCount_smooth*m,ThetaCount+1);
    Y_theta_samples_case_smooth = [];
    t_smooth=tspan(StepCount_smooth);
    step_smooth = 1;
    step_sample_smooth = 1;
    Time_subset_plot_smooth=[];

%This loop to solve for y in each time step
for ti=1:StepCount_smooth
    T_min = 1.5;
    if alpha2 == 0
        h_case_smooth = max(X_smooth(ti,1),T_min);
        dh_case_smooth = SLmax([X_smooth(ti,1) X_smooth(ti,3) X_smooth(ti,5) X_smooth
            (ti,7) X_smooth(ti,9)],[T_min 0 0 0 0]);
    else
        % max(a,b) = 1/2*(a + b + |a-b|); |a-b| \approx sqrt((a-b)^2+alpha^2)
        h_case_smooth = 1/2*(X_smooth(ti,1)+T_min+sqrt((X_smooth(ti,1)-T_min)^2+
            alpha2^2));
        dh_case_smooth = [1/2*X_smooth(ti,3)*(1+0+1/2*1/sqrt((X_smooth(ti,1)-T_min)
            ^2+alpha2^2))*(2*(X_smooth(ti,1)-T_min))...
            1/2*X_smooth(ti,5)*(1+0+1/2*1/sqrt((X_smooth(ti,1)-T_min)^2+alpha2
            ^2))*(2*(X_smooth(ti,1)-T_min))...
            1/2*X_smooth(ti,7)*(1+0+1/2*1/sqrt((X_smooth(ti,1)-T_min)^2+alpha2
            ^2))*(2*(X_smooth(ti,1)-T_min))...
            1/2*X_smooth(ti,9)*(1+0+1/2*1/sqrt((X_smooth(ti,1)-T_min)^2+alpha2
            ^2))*(2*(X_smooth(ti,1)-T_min))];
    end

    y_output_case2(step_smooth:step_smooth+m-1) = h_case_smooth;
    Y_theta_case_smooth(step_smooth:step_smooth+m-1,1:end) = dh_case_smooth;

    if ismember(ti, t_samples_smooth)
        Y_theta_samples_case_smooth = [Y_theta_samples_case_smooth;dh_case_smooth];
        Time_subset_plot_smooth = [Time_subset_plot_smooth Time_smooth(ti)];
        step_sample_smooth = step_sample_smooth+m;
    end
    step_smooth = step_smooth+m;
end

%Get SY and SVD case2

```



```

 $\mathrm{col}}\}_{_3}(S^L_y)\$'...$ 
        , '$\{\mathrm{col}}\}_{_1}(S_y)\$', '$\{\mathrm{col}}\}_{_2}(S_y)\$', '$\{\mathrm{col}}\}_{_3}(S_y)\$'...
        , 'Location', 'northwest', 'Interpreter', 'latex', 'FontSize', 20);
ylim([-0.5 1]);
xline(0.75, 'k-', 'LineWidth', 2, 'HandleVisibility', 'off');
xline(0.4, 'k-', 'LineWidth', 2, 'HandleVisibility', 'off');
xticks(0:0.2:1)
% 3- Heatmap
heatmap_subplot=subplot(2,2,4, 'Parent', figure_combined);
h = heatmap(Rank_table_logical);
map = [1 0 0;
       0.9290 0.6940 0.1250];
h.Colormap = map;
h.MissingDataColor = [0.4660 0.6740 0.1880];
h.CellLabelColor = 'none';
h.XLabel = 'Time';
h.YLabel = '||\alpha||';
h.FontSize = 24;
h.ColorbarVisible = 'off'; h.GridVisible='off';
alpha_vec_norm = vecnorm(alpha_vec);
alpha_vec_norm_cell = num2cell(alpha_vec_norm');
alpha_vec_norm_cell(2) = {'1e-01'}; alpha_vec_norm_cell(3) = {'1e-02'};
    alpha_vec_norm_cell(4) = {'1e-03'}; alpha_vec_norm_cell(5) = {'1e-04'};
h.XDisplayLabels(total_samples_smooth/5:total_samples_smooth/5:end) = num2cell(
    str2double(h.XDisplayLabels(total_samples_smooth/5:total_samples_smooth/5:end))./
    total_samples_smooth);
for ii = 1:length(h.XDisplayLabels)
    if str2double(h.XDisplayLabels(ii))>1
        h.XDisplayLabels(ii) = {''};
    end
end
h.XDisplayLabels(1) = {'0'};
h.YDisplayLabels = alpha_vec_norm_cell;
h.YDisplayLabels(end)={'L-SERC'};
s = struct(h);
s.XAxis.TickLabelRotation = 0;

function dXdt = Nonsmooth_Stommel_Box_model_ODE(t,X,Param,M_Matrix)
%states
T = X(1);
V = X(2);
% V = abs(T-V);
% Parameters values
theta1 = Param(1);
theta2 = Param(2);
theta3 = Param(3);
%Control inputs
%A/omega = 0.05 figure 1 dynamic tipping, values of B and B_hat, figure 6
B = 2;
B_hat = 1;
A = B - B_hat;
omega = A/0.05;
u1 = B*sin(omega*t);
u2 = B_hat*sin(omega*t);
% ODE system1 (f-function)
dTdt = theta1 + u1 - T - T*abs(T-V); %f1
dVdt = theta2 + u2 - V*theta3 - V*abs(T-V); %f2

```

```

%S_x equations
X11 = X(3);
X21 = X(4);
X12 = X(5);
X22 = X(6);
X13 = X(7);
X23 = X(8);
X14 = X(9);
X24 = X(10);

%Directional derivative with respect to thetas
df1 = [1 0 0]*M_Matrix;
df2 = [0 1 0]*M_Matrix-[0 0 V]*M_Matrix;

%absolute function derivative
u = T-V;
U = [X11-X21 X12-X22 X13-X23 X14-X24];
fsign_u = fsign(u,U);

% ODE system2
X11_dt = df1(1) - X11 - abs(T-V)*X11 - T*fsign_u*(X11-X21);
X12_dt = df1(2) - X12 - abs(T-V)*X12 - T*fsign_u*(X12-X22);
X13_dt = df1(3) - X13 - abs(T-V)*X13 - T*fsign_u*(X13-X23);
X14_dt = df1(4) - X14 - abs(T-V)*X14 - T*fsign_u*(X14-X24);
X21_dt = df2(1) - theta3*X21 - abs(T-V)*X21 - V*fsign_u*(X11-X21);
X22_dt = df2(2) - theta3*X22 - abs(T-V)*X22 - V*fsign_u*(X12-X22);
X23_dt = df2(3) - theta3*X23 - abs(T-V)*X23 - V*fsign_u*(X13-X23);
X24_dt = df2(4) - theta3*X24 - abs(T-V)*X24 - V*fsign_u*(X14-X24);
% dXdT = [dTdt;dVdt;dT_dtheta1_dt;dT_dtheta2_dt;dT_dtheta3_dt;dV_dtheta1_dt;
dV_dtheta2_dt;dV_dtheta3_dt];
dXdT = [dTdt;dVdt;X11_dt;X21_dt;X12_dt;X22_dt;X13_dt;X23_dt;X14_dt;X24_dt];
end
function dXdT = Stommel_Box_model_ODE_smoothing(t,X,Param,M_Matrix,alpha)
%states
T = X(1);
V = X(2);
% V = abs(T-V);
% Parameters values
theta1 = Param(1);
theta2 = Param(2);
theta3 = Param(3);
%Control inputs
%A/omega = 0.05 figure 1 dynamic tipping, values of B and B_hat, figure 6
B = 2;
B_hat = 1;
A = B - B_hat;
omega = A/0.05;
u1 = B*sin(omega*t);
u2 = B_hat*sin(omega*t);

% Smoothing abs
smooth_abs = sqrt((T-V)^2+alpha^2);

% ODE system1 (f-function)
dTdt = theta1 + u1 - T - T*smooth_abs; %f1
dVdt = theta2 + u2 - V*theta3 - V*smooth_abs; %f2

```

```

%S_x equations
X11 = X(3);
X21 = X(4);
X12 = X(5);
X22 = X(6);
X13 = X(7);
X23 = X(8);
X14 = X(9);
X24 = X(10);

%Directional derivative with respect to thetas
df1 = [1 0 0]*M_Matrix;
df2 = [0 1 0]*M_Matrix-[0 0 V]*M_Matrix;

%smooth abs function derivative
grad_smooth_abs = (T-V)*((T-V)^2+alpha^2)^(-1/2);

% ODE system2
X11_dt = df1(1) - X11 - smooth_abs*X11 - T*grad_smooth_abs*(X11-X21);
X12_dt = df1(2) - X12 - smooth_abs*X12 - T*grad_smooth_abs*(X12-X22);
X13_dt = df1(3) - X13 - smooth_abs*X13 - T*grad_smooth_abs*(X13-X23);
X14_dt = df1(4) - X14 - smooth_abs*X14 - T*grad_smooth_abs*(X14-X24);
X21_dt = df2(1) - theta3*X21 - smooth_abs*X21 - V*grad_smooth_abs*(X11-X21);
X22_dt = df2(2) - theta3*X22 - smooth_abs*X22 - V*grad_smooth_abs*(X12-X22);
X23_dt = df2(3) - theta3*X23 - smooth_abs*X23 - V*grad_smooth_abs*(X13-X23);
X24_dt = df2(4) - theta3*X24 - smooth_abs*X24 - V*grad_smooth_abs*(X14-X24);
% dXdT = [dTdt;dVdt;dT_dtheta1_dt;dT_dtheta2_dt;dT_dtheta3_dt;dV_dtheta1_dt;
           dV_dtheta2_dt;dV_dtheta3_dt];
dXdT = [dTdt;dVdt;X11_dt;X21_dt;X12_dt;X22_dt;X13_dt;X23_dt;X14_dt;X24_dt];
end

```

REFERENCES

- [1] P. Stechlinski, S. Eisa, and H. Abdelfattah, “Identifiability and observability of nonsmooth systems via taylor-like approximations,” *arXiv preprint arXiv:2403.12930*, 2024.
- [2] K. A. Khan and P. I. Barton, “A vector forward mode of automatic differentiation for generalized derivative evaluation,” *Optimization Methods and Software*, vol. 30, no. 6, pp. 1185–1212, 2015.
- [3] P. I. Barton, K. A. Khan, P. Stechlinski, and H. A. J. Watson, “Computationally relevant generalized derivatives: theory, evaluation and applications,” *Optimization Methods and Software*, vol. 33, pp. 1030–1072, 2018.
- [4] K. Khan, “Nonsmooth-forward-ad.” <https://github.com/kamilkhanlab/nonsmooth-forward-ad.git>, 2022.
- [5] C. Budd, C. Griffith, and R. Kuske, “Dynamic tipping in the non-smooth Stommel-box model, with fast oscillatory forcing,” *Physica D: Nonlinear Phenomena*, vol. 432, p. 132948, 2022.