Version of Saturday 22nd June, 2019, 10:21

SLIDING INTEGRATION WITH NO PROJECTION

LUCA DIECI

Abstract. In this work we show that numerical integration during sliding motion, for piecewise-smooth systems, can be performed without the need to project the approximate solutions on the constraints' surface. We give a general algorithm, an order of approximation result, and show the effectiveness of the technique on several examples. Comparison between projected and non-projected methods and the standard regularization approach is also given.

1. Introduction: The problem

Piecewise Smooth (PWS) systems, or systems with discontinuous right-hand-side, appear pervasively in applications, and are continuing to receive attention from the mathematical community from several different facets, numerical, theorical, and applications.

The basic problem that motivated our work is the following. We have a differential system where the vector field changes (typically, discontinuously) as the trajectory encounters a discontinuity surface Σ of co-dimension 1, which is defined as 0-set of a $\mathcal{C}^2$-smooth function h. In mathematical terms, we can assume that the surface Σ separates the state space in two regions, where $h > 0$ and where $h < 0$. So, we will consider the following problem:

$$\mathbb{R}^n = R_- \cup \Sigma \cup R_+ \,,$$

where

$$\begin{aligned} \Sigma &= \{x \in \mathbb{R}^n |\ h(x) = 0 \ ,\ \ h : \mathbb{R}^n \to \mathbb{R} \ \} \ , \\ R_- &= \{x \in \mathbb{R}^n |\ h(x) < 0\} \ , \quad R_+ = \{x \in \mathbb{R}^n |\ h(x) > 0\} \ . \end{aligned} \tag{1.1}$$

Further, we have the PWS system

$$x'(t) = f(x(t)) = \begin{cases} f_-(x(t)) \ , & x \in R_- \ , \\ f_+(x(t)) \ , & x \in R_+ \ , \end{cases} \qquad x(0) = x_0 \in R_\pm \ , \tag{1.2}$$

where we will assume that $f_\pm$ are defined in their respective domains $R_\pm$ as well as on Σ and in a neighborhood of it. Finally, we will let ∇h define the gradient (normal) to Σ, and assume that $\nabla h(x) \neq 0$, $\forall x \in \Sigma$ (hence, near it).

1991 *Mathematics Subject Classification.* 65A36, 65P99.

Key words and phrases. Piecewise smooth systems, sliding motion, numerical integration, explicit Runge Kutta methods.

The author is very thankful to Cinzia Elia, for several and fruitful discussions on this topic.

From (1.2), obviosuly $f(x)$ is not defined if $x \in \Sigma$, and an extension of the concept of classical solution is needed. The extension we consider is due to Filippov, [6], who proposed to work with a differential inclusion:

$$x'(t) \in F(x(t)) = \begin{cases} f_-(x(t)) & x \in R_- \\ \{(1-\alpha)f_- + \alpha f_+,\ \alpha \in [0,1]\} & x \in \Sigma \\ f_+(x(t)) & x \in R_+ \end{cases} .$$

With this, he proceeded to define a solution in an interval $[0,\tau)$ (henceforth, a solution in the sense of Filippov) to be *an absolutely continuous function* $x : [0,\tau) \to \mathbb{R}^n$ *such that for almost all* $t \in [0,\tau)$ *it holds that*

$$x'(t) \in F(x(t)) \ .$$

The following fundamental result is also in [6, pp. 107-108].

Theorem 1.1 (Existence and Uniqueness)**.** *Let* $f_\pm$ *be* C^1*, and* h *be* C^2 *(in a neighborhood of* Σ*). If, at any point* $x \in \Sigma$ *we have that at least one of* $\nabla h^T(x) f_-(x) > 0$ *and* $\nabla h^T(x) f_+(x) < 0$ *holds, then there exists a unique Filippov solution from each initial condition.* □

Theorem 1.1 in particular comprises the configurations of *transversal* crossing and *attractive* sliding mode.

Definition 1.2. *Let* $x \in \Sigma$*.*

(i) *In case*

$$[\nabla h^T(x) f_-(x)]\ [\nabla h^T(x) f_+(x)] > 0 \ ,$$

we say that at x *there is a* **transversal crossing***. The trajectory will leave* Σ *to* R_+*, if* $\nabla h^T(x) f_-(x) > 0$*, or* R_-*, if* $\nabla h^T(x) f_+(x) > 0$*.*

(ii) *In case*

$$\nabla h^T(x) f_-(x) > 0 \quad \text{and} \quad \nabla h^T(x) f_+(x) < 0 \ ,$$

we say that at x *there is an* **attracting sliding mode***. In such case, attracting sliding motion takes place on* Σ *with Filippov vector field*

$$\begin{aligned} x' = f_{_F}(x) =& (1-\alpha(x)) f_-(x) + \alpha(x) f_+(x) \\ \alpha(x) :\ \nabla h^T(x) f_{_F}(x) = 0 \to\ \alpha(x) =& \frac{\nabla h^T(x) f_-(x)}{\nabla h^T(x)(f_-(x) - f_+(x))} \ . \end{aligned} \tag{1.3}$$

In what follows, we will make the two following simplifying assumptions (these are generic, for $\mathcal{C}^1$ vector fields $f_\pm$ and $\mathcal{C}^2$ function h).

Assumption 1.3.

(i) **Entries** *on* Σ *are* **transversal***.*

(ii) *In case sliding motion is occurring,* **exits** *from* Σ *are* **tangential***. In particular, a trajectory will exit* Σ *with* f_- *whenever* $\alpha = 0$*, and with* f_+ *when* $\alpha = 1$*.*

1.1. **Numerical methods for PWS systems.** Essentially[1], two numerical approaches have been considered for the integration of a PWS system: to regularize the system (*regularization approach*), or to monitor changes in regime and adjust accordingly (*event driven*). Both methods require integration of a differential system, a task that we henceforth assume is performed with a Runge-Kutta (RK) scheme. As customary, this is represented by the tableau

$$\begin{array}{c|c} c & A \\ \hline & b^T \end{array} \quad , \tag{1.4}$$

where our interest will chiefly be in explicit schemes, so that the coefficient matrix A will be strictly lower triangular.

1.1.1. *Regularization.* The idea here is to replace the PWS vector field (1.2) with a globally smooth one, which reduces to $f_\pm$ away from a small neighborhood of Σ. This basic approach was introduced by Sotomayour and Teixeira in [9], and it has been used many times (by us, and others) also computationally.

For example, one may use the system

$$x' \;=\; (1-g_\epsilon(z))f_-(x) + g_\epsilon(z)f_+(x)\ , \qquad z=h(x)\ , \tag{1.5}$$

where g_ϵ is a C^k transition function, $k\geq 1$, such that

$$g_\epsilon(z) = \begin{cases} 1 & z\geq \epsilon \\ 0 & z\leq -\epsilon \end{cases} \tag{1.6}$$

and $g'(z) > 0$ in $(-\epsilon,\epsilon)$. A possibility (which we will use later in our numerical experiments) is to take

$$g_\epsilon(z) = \begin{cases} 1 & z\geq \epsilon \\ \frac{1}{2}+\frac{z}{4\epsilon}\left(3-(\frac{z}{\epsilon})^2\right) & -\epsilon < z < \epsilon \\ 0 & z\leq -\epsilon \end{cases} \quad \text{and } \ z=h(x)\ .$$

Remarks 1.4.

(i) *When Σ is attractive, it is well understood (e.g., see [9]) that for $x\in\Sigma$, the limit as $\epsilon\to 0$ of (1.5) coincides with the sliding vector field (1.3). A bit emphatically, we can say that, in the limit of $\epsilon\to 0$, the regularization approach recovers the dynamics of (1.2).*

(ii) *Of course, the main computational appeal of this regularization aproach is that the regularized problem can be treated as a standard ODE, and one can use available software for IVPs. However, there are some potential concerns. For one, we need to select small values of ϵ in order to be faithful to the dynamics of (1.2). But, for ϵ small, it is quite possible that the numerical trajectory will fail to step inside the ϵ-neighborhood of Σ, and thus –when Σ is attractive– one may*

[1]Of course, there is always the possibility to integrate the PWS system as if it were a smooth system, but this is not considered in this work

end up using $f_\pm$ causing an undesirable spurious oscillatory behavior around Σ. Also, in a neighborhood of Σ, the vector field (1.5) *has large derivatives, and this is also undesirable if we want to control the local error. Indeed, our results in Section 4 will confirm that the regularization approach is at once advantageous (since it is simple to use) and undesirable (since it is subject to spurious oscillations).*

1.1.2. *Event driven.* Here, one integrates the given PWS system, by **adjusting** the stepsize(s) so to fall "exactly" on Σ, decide whether a transversal crossing or sliding motion takes place, and eventually again adjust the stepsize to leave Σ during sliding motion when $\alpha = 0, 1$. One needs to monitor the different regimes in which the trajectory is: in a region $R_\pm$, or on Σ, and whether there is a transversal crossing or a sliding regime, and appropriate numerical integration schemes have to be adopted during the different phases (see Remark 1.5). Several recent works have been concerned with this approach, and we mention one-sided schemes and time reparametrization (letting $\tau = h(x)$), e.g. see [4, 5], and the public domain code `DISODE45`, see [2]. The obvious advantage of event driven techniques is that they produce approximations which are faithful to the dynamics of the model (1.2).

Remark 1.5. *When integrating* (1.3) *on Σ, we must make sure that the numerical solution remains on it. Using a RK scheme on the differential system* (1.3)*, in general, will not guarantee this fact, and a projection step is thus required (unless we have a special functional form for Σ). Assuming that the basic integrator uses a Runge-Kutta scheme, the projection itself can be done in more than one way, for example only at the end of the step or also for the intermediate stages (which, of course, is an additional expense). In any case, by their very nature, projected schemes require additional work, and thus have an added computational expense.*

In this work, our purpose is twofold.

(i) First, we will show that, while integrating the piecewise smooth (PWS) system during sliding motion regime, we can **avoid projecting** the solution in order to remain on the discontinuity surface. We call the resulting method *Event Method Without Projection*, as opposed to *Event Method With Projection*.

(ii) On a few examples, we will **provide a comparison** between the Event Methods With/Without Projection, and further compare these to the regularization approach. To the best of our knowledge, a comparison of different technique for use in integrating PWS systems is lacking, and ours is a step in that direction.

As it turns out, the method without-projection is considerably less expensive, and much more robust, than the method with-projection. Like projection techniques, it is also reproducing the dynamics of (1.2), unlike solving the regularized problem. Our examples below will validate both of these claims.

The simple idea we will exploit arises from the realization that the surface Σ is typically given by expressing one of the variables, say x_1, in terms of the others. This means that the differential system can be written in partitioned form, singling out the variable x_1, which is however known (in terms of the other variables) on Σ, and therefore on Σ it suffices to find the other components and then recover x_1 from the algebraic relation defining it. One can then generalize this approach, for as long as it is known an index i, $1 \leq i \leq n$, for which the derivatives $h_{x_i} \neq 0$ (that such index must exist is guaranteed by our assumption that $\nabla h(x) \neq 0$ for $x \in \Sigma$). Below we present this idea in somewhat greater generality (not restricting to sliding motion and a co-dimension one discontinuity surface), and show order results and examples, then look at specific instances of sliding motion.

In the context of index-one differential-algebraic equations (DAEs), the technique we present in this work is known as the *state-space* formulation; see [8, pp. 375-376]. As far as we know, the motivation –and use– for sliding motion in the context of an event method is original. We also note that, for DAEs, the state-space method is used to eliminate once and for all the algebraic constraint, while for us the goal is to avoid altogether integration for the variable expressing the surface Σ; in other words, we reduce the size of the differential system we integrate.

1.2. **A model differential system.** Consider the following basic problem. Below, we let $x \in \mathbb{R}^p$ and $y \in \mathbb{R}^s$, where $p, s \geq 1$, and we let $n = p + s$. We have the differential system in $\mathbb{R}^n$:

$$\text{(1.7)} \quad \begin{cases} \dot{x} &= F(x,y) \\ \dot{y} &= G(x,y) \ , \end{cases} \qquad \text{or} \qquad \dot{z} = H(z) \ , \quad z = \begin{pmatrix} x \\ y \end{pmatrix} \ , \ H = \begin{pmatrix} F \\ G \end{pmatrix} \ ,$$

where F and G are as smooth as desired.

Assume we know that the y-part of the solution of (1.7) will need to remain on a surface Σ, where Σ is defined explicitly by the relation $y = k(x)$. In other words, starting with ICs (x_0, y_0) and $y_0 = k(x_0)$, the exact solution of (1.7) will give (x, y) so that $y = k(x)$.

When we discretize (1.7) with an explicit Runge-Kutta scheme, we will obtain a discrete solution (x_j, y_j), which approximates $(x(t_j), y(t_j))$ at times t_j, $j = 0, 1, 2, \ldots$. The basic step with stepsize τ can thus be written as

$$\text{(1.8)} \quad \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + \tau \Phi(\tau, F, G, x_0, y_0) \ , \qquad \text{or} \qquad z_1 = z_0 + \tau \Phi(\tau, H, z_0) \ ,$$

where of course Φ is the discrete solution operator.

Now, it is well understood that, in general[2], the discrete solution will not satisfy the invariance relation, that is $y_j \neq k(x_j)$, $j = 1, 2, \ldots$. This is a standard concern, and one way to ensure having a discrete solution satisfying the constraint is to post-process the answers obtained by the RK scheme, by **projecting** them back on the constraint manifold. One way to do this is

[2]E.g., unless the constraint expressed by $y = k(x)$ is linear

to first take a step as in (1.8), then modify y_1 by projecting it onto the constraint manifold, so that the new y_1 will satisfy $k(x_1) = y_1$. The projection itself can be done with a Lagrange multipliers technique, by minimizing the distance to the constraint manifold. We call the resulting method: *integrate and project*. For later reference, we review a possible implementation of the basic step of this integrate and project approach; this implementation is the same that was used in [3], though we emphasize that at present our model problem (1.7) is not necessarily one of sliding motion integration. Below, we let $h(x, y) \equiv y - k(x)$.

1.2.1. *Algorithm Integrate and Project.*

1. Integrate (1.7) over one step τ, assuming that $y_0 - k(x_0) = 0$, to obtain

$$\widehat{z}_1 = \begin{bmatrix} \widehat{x}_1 \\ \widehat{y}_1 \end{bmatrix} = z_0 + \tau\Phi(\tau, H, z_0) \ . \tag{1.9}$$

2. Let $g(z) = \frac{1}{2}(z - \widehat{z}_1)^T(z - \widehat{z}_1)$. Seek $z = \begin{bmatrix} x \\ y \end{bmatrix}$, such that $h(x, y) = 0$ and $g(z)$ is minimized. Using the Lagrange multipliers technique, this requires solving the nonlinear system

$$\begin{bmatrix} \nabla_z g(z) + \sum_{j=1}^{s} \lambda_j \nabla_z h_j(z) \\ h(z) \end{bmatrix} = 0 \ . \tag{1.10}$$

   For as long as the integration stepsize τ is sufficiently small, there is a unique solution to (1.10), call it z_1, which is the closest value z_1 to $\widehat{z}_1$, satisfying $h(z_1) = 0$. (See Lemma 2.4 below).

3. With z_1, the solution of (1.10), we recover x_1 and y_1 from $z_1 = \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$, and then continue integrating (1.7) starting with the values x_1, y_1. Compactly, we write

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \text{Proj}\left(\begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + \tau\Phi(\tau, F, G, x_0, y_0) \right) \ . \tag{1.11}$$

It is obvious that the technique based on (1.11) results in a numerical scheme that has the same order of the underlying basic RK scheme. At the same time, the need for having a sufficiently small stepsize τ expressed in point 2. above might become a hindrance to obtain an efficient time stepping scheme (see our experiments below). Finally, for completeness, we detail the way in which we solve the system (1.10) in our experiments. We use a simplified Newton's method with frozen Jacobian. More specifically, given the initial guess $z^{(0)} = \widehat{z}_1$ and $\lambda^{(0)} = 0$, we form and factor the Jacobian J at this

initial guess,

$$J = \begin{bmatrix} I & \nabla h_1 & \dots & \nabla h_s \\ (\nabla h_1)^T & 0 & \dots & 0 \\ \vdots & \vdots & \dots & \vdots \\ (\nabla h_s)^T & 0 & \dots & 0 \end{bmatrix} ,$$

and then perform the following simplified Newton's iteration.

(0) Given values of `MaxIts` and TOL.

(1) For $k = 0, 1, \dots,$ MaxIts, or until $\| \begin{bmatrix} z^{(k)} - \widehat{z} + \sum_{j=1}^{s} \lambda_j^{(k)} \nabla h_j(z^{(k)}) \\ h(z^{(k)}) \end{bmatrix} \| <$ `TOL`,

Find the update $\begin{bmatrix} \Delta z \\ \Delta \lambda \end{bmatrix}$ by solving

$$J \begin{bmatrix} \Delta z \\ \Delta \lambda \end{bmatrix} = - \begin{bmatrix} \nabla g(z) + \sum_{j=1}^{s} \lambda_j \nabla_z h_j(z) \\ h(z) \end{bmatrix}$$

and let $z^{(k+1)} = z^{(k)} + \Delta z$, $\lambda^{(k+1)} = \lambda^{(k)} + \Delta \lambda$.

Not reaching the required precision `TOL` or performing more than `MaxIts` iterations will result in a failure.

In this work, we will compare the *Algorithm Integrate and Project* to the following.

1.2.2. *Algorithm Substitute and Integrate.*

1. We substitute a-priori the relation $y = k(x)$ in the differential equation for x. That is, we consider

$$\dot{x} = F(x, k(x)) \ . \tag{1.12}$$

2. We discretize the reduced system (1.12), obtaining approximations x_j at the values t_j, and further approximate the y-values by using the relation $y_j = k(x_j)$.

The first result we give is Theorem 2.2; this is actually a rather obvious result, but we give it for completeness. It states that the method "Substitute and Integrate" will render approximations for both x and y values of the same order as the order of the basic scheme used to integrate (1.12), and thus we will have obtained a method with no need of projection step in order to satisfy the constraint $y = k(x)$, and of order of accuracy the same as that of the scheme used to solve (1.12).

Our second result clarifies the need for a sufficiently small stepsize in the context of a projection scheme.

2. Approximation results

We have (1.7) with exact solution y which can be written as $y = k(x)$. So, let us rewrite the two systems we are considering, (1.7) and (1.12):

$$\begin{aligned} \dot{x} &= F(x,y) \\ \dot{y} &= G(x,y) \\ x(0) = x_0\ ,&\ \ y(0) = y_0 = k(x_0)\ , \end{aligned} \tag{2.1}$$

and

$$\dot{x} = F(x,k(x)), \quad x(0) = x_0\ . \tag{2.2}$$

Lemma 2.1. *If, in problem* (2.1)*, the solution y satisfies the relation $y = k(x)$, then the vector field G has a special form, namely*

$$G(x,y) = Dk(x)F(x,y)\,,$$

where $Dk(x)$ is the Jacobian of k. Thus, as function of x, y satisfies the PDE

$$\begin{bmatrix} y_{x_1} & y_{x_2} & \dots & y_{x_p} \end{bmatrix} \ = \ Dk(x)\ .$$

Proof. If $y = k(x)$, then $\dot{y} = Dk(x)\dot{x} = Dk(x)F(x,y)$ as required. And, clearly, if $y = k(x)$, then $D_x y = Dk$. □

Theorem 2.2. *Let $t \in [0,T]$, and $\tau = T/N$. For $k = 1,2,\dots,N$, let (x_k, y_k) be the numerical solution to* (2.1) *obtained with a method of order p, and let $\widetilde{x}_k$ be the numerical solution of* (2.2) *and further $\widetilde{y}_k = k(\widetilde{x}_k)$. Then, $x_k - \widetilde{x}_k = \mathcal{O}(\tau^p)$, and $y_k - \widetilde{y}_k = \mathcal{O}(\tau^p)$.*

Proof. This is a simple consequence of the following fact, which is a well known order result for numerical schemes. We have

$$\begin{bmatrix} x(t_k) \\ y(t_k) \end{bmatrix} - \begin{bmatrix} x_k \\ y_k \end{bmatrix} \ = \ \mathcal{O}(\tau^p) \qquad \text{and} \qquad x(t_k) - \widetilde{x}_k \ = \ \mathcal{O}(\tau^p)\,.$$

Further,

$$k(x(t_k)) - k(\widetilde{x}_k) \ = \ k_x(\widetilde{x}_k)\,(x(t_k) - \widetilde{x}_k) + \mathcal{O}(\tau^{2p})\,,$$

and thus we immediately obtain $x_k - \widetilde{x}_k \ = \ x(t_k) - \widetilde{x}_k - (x(t_k) - x_k) = \mathcal{O}(\tau^p)$ and $y_k - \widetilde{y}_k \ = \ y(t_k) - \widetilde{y}_k - (y(t_k) - y_k) \ = \ \mathcal{O}(\tau^p)$. □

Remark 2.3. *It should be appreciated that the quantities hidden in the $\mathcal{O}(\tau^p)$ terms in Theorem 2.2 are not the same, since the derivatives appearing in the error term (the $\mathcal{O}(\tau^p)$ term) will be typically evaluated at different points.*

Lemma 2.4. *For τ sufficiently small in* (1.9)*, there is a unique solution to* (1.10)*.*

Proof. Write the system (1.10) as $B(z,\lambda)=0$, where B is the left hand side of (1.10):

$$B(z,\lambda)=\begin{bmatrix}(z-\widehat{z}_1)+\sum_{j=1}^{s}\lambda_j\nabla_z h_j(z)\\ h(z)\end{bmatrix}\,,\quad \widehat{z}_1=z_0+\tau\Phi\,,$$

and note that $\widehat{z}_1$ depends on τ.

Next, let us look at $\tau=0$. Since z_0 satisfies $h(z_0)=0$, obviously we have $B(z_0,0)=0$. Then, evaluate the derivative DB at $(z_0,0)$, to get

$$DB(z_0,0)=[B_z\ ,\ B_\lambda]_{(z_0,0)}=\begin{bmatrix} I & \nabla_z h_1(z_0) & \dots & \nabla_z h_s(z_0)\\ (\nabla_z h_1(z_0))^T & 0 & \dots & 0\\ \vdots & \vdots & \dots & \vdots\\ (\nabla_z h_s(z_0))^T & 0 & \dots & 0\end{bmatrix}\,,$$

or more compactly as $DB(z_0,0)=\begin{bmatrix} I & C\\ C^T & 0\end{bmatrix}$, with $C=[\nabla_z h_1(z_0)\ \dots\ \nabla_z h_s(z_0)]$, that is

$$C=\begin{bmatrix}-\nabla_x k_1 & -\nabla_x k_2 & \dots & -\nabla_x k_s\\ 1 & 0 & \dots & 0\\ 0 & 1 & \dots & 0\\ \vdots & & \ddots & \vdots\\ 0 & 0 & \dots & 1\end{bmatrix}_{z_0}\,.$$

Therefore, $DB(z_0,0)$ factors as $\begin{bmatrix} I & 0\\ C^T & I\end{bmatrix}\begin{bmatrix} I & C\\ 0 & -C^TC\end{bmatrix}$, and thus $DB(z_0,0)$ is invertible if C is full rank. But this last fact is clear from the above form of C. Therefore, $DB(z_0,0)$ is invertible and the Implicit Function Theorem guarantees that in a neighborhood of $\tau=0$ there exists a unique solution $(z(\tau),\lambda(\tau))$, smooth (in τ), solving $B(z,\lambda)=0$ and passing through $(z_0,0)$ at $\tau=0$. □

Remark 2.5. *In a practical situation, it may be necessary to have τ quite small in order to guarantee that there is a unique solution to* (1.10)*, and such small value of τ is not necessarily warranted by purely accuracy requirements. See our Examples in Section 4.*

3. Numerical experiments

The two examples below serve a triple purpose: (i) to confirm that the decrease factors in terms of the stepsize τ we obtain for the method "Substitute and Integrate" are the same we obtain for the method applied to the full system (in other words, that the difference between the solutions decrease according to the expected order); (ii) that the projected integrators and the method "Substitute and Integrate" are of same order of accuracy, with the former method requiring a stepsize restriction; and (iii) that the method

τ	Euler	Midpoint	4-th order
0.1	Fail	1.6107e-02	2.1245e-04
0.01	4.5535e-03	1.0995e-04	1.4311e-08
0.001	4.6957e-04	1.0490e-06	1.3982e-12

TABLE 1. Differences between using Projected integration on (3.1) and (3.2). Time interval is $[0, 10]$, IC $x(0) = 0$, and error is ∞-norm of the differences at all grid points.

"Substitute and Integrate" produces meaningful savings with respect to the method "Integrate and Project."

We implemented three different explicit Runge Kutta schemes, of order 1 (forward Euler), 2 (explicit midpoint rule), and 4 (the 3/8-th rule), with the following tableaux, respectively:

$$\begin{array}{c|c} 0 & 0 \\ \hline & 1 \end{array}, \qquad \begin{array}{c|cc} 0 & 0 & 0 \\ 1/2 & 1/2 & 0 \\ \hline & 0 & 1 \end{array}, \qquad \begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ 1/3 & 1/3 & 0 & 0 & 0 \\ 2/3 & -1/2 & 1 & 0 & 0 \\ 1 & 1 & -1 & 1 & 0 \\ \hline & 1/8 & 3/8 & 3/8 & 1/8 \end{array}$$

Example 3.1. *Here, the systems* (1.7) *is*

$$\frac{d}{dt}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 2y - 3x \\ (2y - 3x)(\cos(x) - x^2) \end{bmatrix} , \tag{3.1}$$

and $y = k(x)$ *is the curve* $y = \sin(x) - x^3/3 + 2$*, so that* (1.12) *is*

$$\frac{dx}{dt} \;=\; 2\sin(x) - 2x^3/3 + 4 - 3x \; . \tag{3.2}$$

We integrate on $[0, T]$*, with stepsize* $\tau = 1/10^k$*,* $k = 1, 2, 3$*, and IC* $x(0) = 0$*. As expected, integrating* (3.1) *and* (3.2) *gave answers fully consistent with the order of the method used, and there is no need to report on these results.*

More interesting are the results in Table 1, where we compute the difference in the answers obtained by projecting the values obtained at each step while integrating (3.1)*, see* (1.11)*, and by integrating* (3.2)*. Initial condition is taken* $x(0) = 0$*, and time interval is* $[0, 10]$*. It should be noted that projected Euler method fails for* $\tau = 0.1$*: what fails is the projection step. Otherwise, the methods behave as one would expect given their order.*

Finally, we measured the computational savings (if any) obtained by integrating (3.2) *with respect to the projected integration of* (3.1) *as in* (1.11)*. We measured the computational cost by the execution time. We chose 100 random initial conditions, and integrated with our three basic schemes: Projected integrators on* (3.1) *versus direct integration of* (3.2)*. For* $\tau = 0.1$*, all* **projected schemes** *(Euler, midpoint, and 4th order RK)* **failed** *some of the time, while for* $\tau = 0.01, 0.001$*. Computational savings are expressed in the table below, as ratios of* `ProjTime/NoProjTime` *(time taken when using*

τ	Savings Euler	Savings Midpoint	Savings 4-th order
0.01	5.4437	2.4053	2.3509
0.001	4.8653	2.2728	2.0697

TABLE 2. Savings when bypassing use of projected integrators. Shown are ratios of times required for projected integration of (3.1) to the times required for direct integration of (3.2). Expenses computed by integrating 100 random ICs over $[0, 10]$.

τ	Euler	Midpoint	4-th order
0.1	2.0582e-02	1.0132e-03	1.2996e-05
0.01	1.4283e-03	8.8429e-06	1.0334e-09
0.001	1.3769e-04	8.6997e-08	1.0869e-13

TABLE 3. Differences between using Projected integration on (3.3) and direct integration of (3.4). Time interval is $[0, 10]$, IC $x_1(0) = x_2(0) = 0$, and error is ∞-norm of the differences at all grid points.

projected schemes, versus time required when bypassing the projection). We summarize in Table 2.

Example 3.2. *This second example is of a system in $\mathbb{R}^4$, where $y = k(x)$ is a two-dimensional surface. The system* (1.7) *is*

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} -2x_1 - x_2 + y_1 - 1 \\ x_1 - 2x_2 + 2 - y_1 y_2 \\ \cos(x_1)\dot{x}_1 - x_2^2\dot{x}_2 \\ -\sin(x_2)\dot{x}_2 + x_1\dot{x}_1 \end{bmatrix}, \tag{3.3}$$

and $\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} k_1(x_1, x_2) \\ k_2(x_1, x_2) \end{bmatrix}$ *is the 2-d surface* $\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} \sin(x_1) - x_2^3/3 + 1 \\ \cos(x_2) + x_1^2/2 - 1 \end{bmatrix}$, *so that* (1.12) *becomes*

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -2x_1 - x_2 + \sin(x_1) - x_2^3/3 \\ x_1 - 2x_2 + 2 - (\sin(x_1) - x_2^3/3 + 1)(\cos(x_2) + x_1^2/2 - 1) \end{bmatrix}. \tag{3.4}$$

We integrate from $t = 0$ to $t = 10$ with stepsize $\tau = 1/10^k$, $k = 1, 2, 3$, and initial condition $x_1(0) = x_2(0) = 0$. Again, as expected, the answers obtained integrating (3.3) *and* (3.4) *are in perfect agreement with the order of the methods.*

Instead, in Table 3, we compute the difference in the answers obtained by projecting the values obtained at each step while integrating (3.3)*, see* (1.11)*, and by integrating* (3.4)*. Initial condition is taken $x_1(0) = x_2(0) = 0$, and time interval is $[0, 10]$. All methods behave as one would expect given their order.*

τ	Savings Euler	Savings Midpoint	Savings 4-th order
0.01	6.6397	2.4582	1.6263
0.001	5.3642	1.9901	1.2822

TABLE 4. Savings when bypassing use of projected integrators. Shown are ratios of times required for projected integration of (3.3) to the times required for direct integration of (3.4). Expenses computed by integrating 100 random ICs over $[0, 10]$.

Finally, in Table 4 we report on the computational savings obtained by integrating (3.4) *with respect to the projected integration of* (3.3) *as in* (1.11)*, over 100 random initial conditions. For* $\tau = 0.1$*,* **all projected RK schemes failed** *some of the time. Computational savings are expressed as ratios of* `ProjTime/NoProjTime`*.*

Summarizing this first set of experiments, we see that use of the Algorithm "Substitute and Integrate" in Section 1.2.2 is clearly as accurate as the Algorithm "Integrate and Project" of Section 1.2.1, but less expensive, and more robust (it never failed). The cost comparison of course depends on several factors, including the problem itself, specifically the value of s (number of constraints) in relation to the value of n (the dimension of the problem), but also the scheme used and the stepsize; these considerations notwithstanding, we observed that on average the cost of the "Substitute and Integrate" algorithm is about 15-20% of the cost of the "Integrate and Project" algorithm when using Euler method, about 40-50% for Heun's method, and in the range 42-78% when using the 4th order RK scheme.

4. Application: sliding motion

Here we show application to the case of sliding motion for a PWS system. Notation is from Section 1. We implemented (and used) three different numerical methods: (a) an event driven approach (see Section 1.1.2), using projected integration during sliding motion regime ("Algorithm Integrate and Project"), with projection step done as described in Section 1.2.1, (b) an event driven approach bypassing the projection phase ("Algorithm Substitute and Integrate" of Section 1.2.2), and (c) the regularization approach (see Section 1.1.1). In all cases, we used the basic 4th order RK scheme of Section 3, in a constant stepsize mode, with the following modifications: (i) for event driven methods, we adjust the stepsize to monitor the events *"exactly"*, that is the points where we arrive on Σ or leave it; (ii) event location is done with a simple bisection approach, declaring convergence when the relevant monitor function (or the interval width) are less than a value `TOL`, which we take to be 10`eps` (recall that `eps` $\approx 2.2 \times 10^{-16}$); (iii) finding the projected values is done with the simplified Newton iteration described in Section 1.2.1 with `MAXITS` $= 10$ and `TOL` $= 10$`eps`; (iv) for the regularization

approach, we integrate the system (1.5) with g_ϵ given in (1.6). Choosing the value of ϵ for the regularization approach is actually quite delicate, since only as $\epsilon \to 0$ one is expected to recover the dynamics of (1.2); see the discussion in the examples below.

Example 4.1. *This is a model of motion of a block of unit mass on a moving belt. When the block gets in contact with the belt, it may* **stick** *to it (and hence it will follow the motion of the belt) until it is pulled back from a restoring spring force,* **slip** *phase, and so forth. Our model is a generalization of one in* [7]*, where the authors considered the case of a horizontal (flat) belt, whereas we allow for imperfections in its shape, reflected in a slightly ondulated form of the belt. This generalization makes the problem much more challenging, since now there is a periodic orbit comprising of several sticking and slipping regimes.*

The differential system is of the form in (1.2)*:*

$$x'(t) = \begin{cases} \begin{bmatrix} x_2 \\ -x_1 + 1/(1.2 - x_2) \end{bmatrix} , & h(x) < 0 , \\ \\ \begin{bmatrix} x_2 \\ -x_1 - 1/(0.8 + x_2) \end{bmatrix} , & h(x) > 0 , \end{cases} , \tag{4.1}$$

where the function $h(x)$ *is*

$$h(x) \;=\; x_2 - 0.2 + \eta \cos(\omega \pi x_1) \;,\;\; \eta,\; \omega > 0 \;, \tag{4.2}$$

and the discontinuity curve is given by $\Sigma := \{x \;:\; h(x) = 0\}$*. For* $\eta = 0$*, we recover the model of* [7]*, and our interest is for small, but nonzero, values of* η*:* $0 < \eta \ll 1$*, and for* $\omega \gg 1$*, so that* Σ *will be a high frequency and small amplitude perturbation of the flat profile.*

Of course, the values of η *and* ω *impact the performance of the basic integrator, as does the value of* ϵ *when using the regularization approach. Nonetheless, some conclusive evidence emerged from our experiments. In particular, in all cases tested, the events-method with no-projection is considerably less expensive than that with projection; see Table 5. Comparisons with the regularization technique depend on the values of* η *and* ω *as well as the regularization parameter* ϵ*; what is interesting is that –since the overhead of events detection is eliminated– the CPU times for the regularized system are insensitive to the values of* η, ω *(at least with our fixed stepsize integrator). As far as the value of* ϵ*, ideally one would like the regularized problem to have the error induced by the regularization of the same size as the integration error obtained with the basic 4th order RK scheme, so it would be natural to select* $\epsilon = \tau^4$*, where* τ *is the stepsize; however, especially in the case of* $\eta = 0$*, this gave unsatisfactory results, with very obvious instabilities in the numerical solution, so we ended up choosing* $\epsilon = 10^{-3}$ *for the results in Table 5. Of course, when choosing* $\epsilon = 10^{-3}$*, the distance of the computed solution from the exact* Σ *(during what would have been a sliding regime) is appreciable. Finally, even for this larger value of* ϵ*, integration of the*

η	ω	CPU Event-No-Project	CPU Event-Project	Regularization
0	–	1	1.6	1.95
0.01	100	1.55	2.85	2
0.01	200	2.25	4.65	1.95

TABLE 5. Example 4.1: CPU times, normalized, for events-method with and without Projection and for regularized system. Time interval is $[0, 15]$, IC $x_1(0) = x_2(0) = -1$, and stepsize $\tau = 10^{-3}$. For the regularization approach, $\epsilon = 10^{-3}$.

regularized system appears to follow a non-horizontal sliding line (and this same behavior was also observed for other values of η and ϵ). To illustrate our last two comments, see Figure 1.

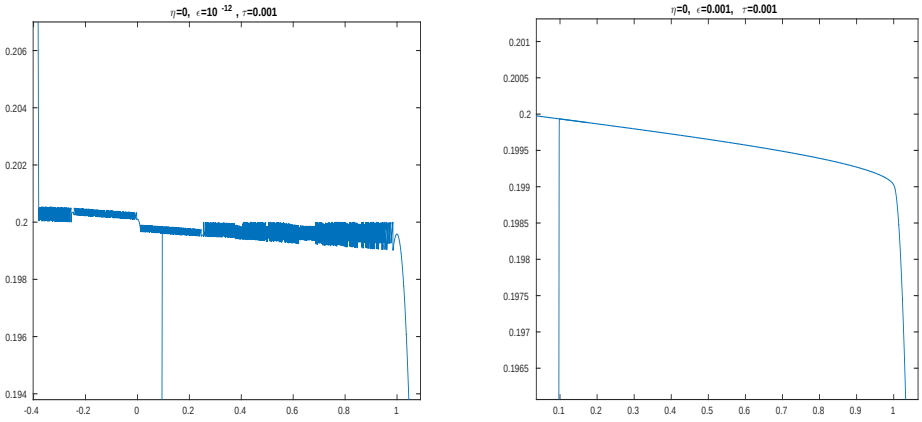

FIGURE 1. Regularized dynamics, $\eta = 0$. Enlargements along "sliding" surface. Left, $\epsilon = 10^{-12}$. Right, $\epsilon = 10^{-3}$.

In summary, at least for the present problem, with our RK integrator, and with constant stepsizes, the events method are more faithfully representing the profile of the original PWS system, whereas integration of the regularized system runs into some instabilities. As far as the events techniques, clearly the one with no-projection outperforms the projected one.

Example 4.2. *This is a modification of a model of an ecosystem from* [1] *(originally from* [10]*), modeling the control of spiders population on a vineyard through human intervention, namely spraying insecticide when the insects population in the vineyard (on which the spiders prey) becomes too large. It is assumed that the vineyard neighbors a wooded area, and that the insects inhabit this area as well. The purpose of spraying is to reduce the number of insects inhabiting the vineyard. In* [1]*, the threshold function triggering spraying is taken constant. Instead, we take the threshold function dependent on the state (namely, on the spiders population), in the form of a*

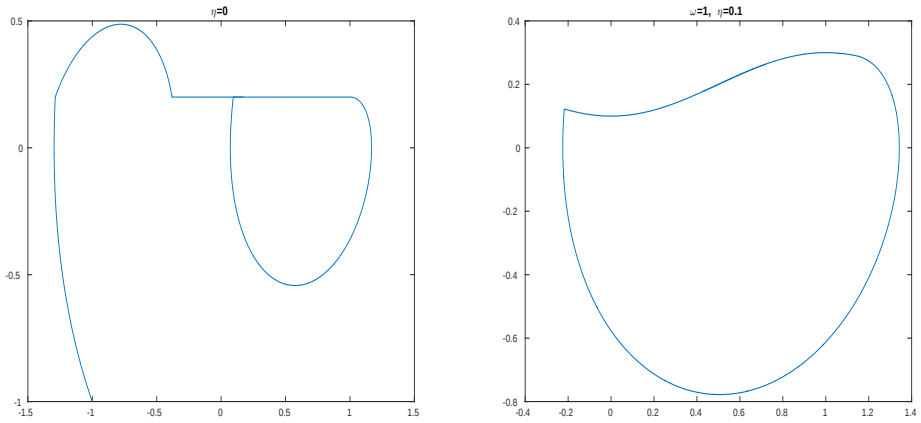

FIGURE 2. Events method. On Left, orbit, and periodic cycle, for $\eta = 0$. On the Right, periodic orbit for $\omega = 1$, $\eta = 0.1$.

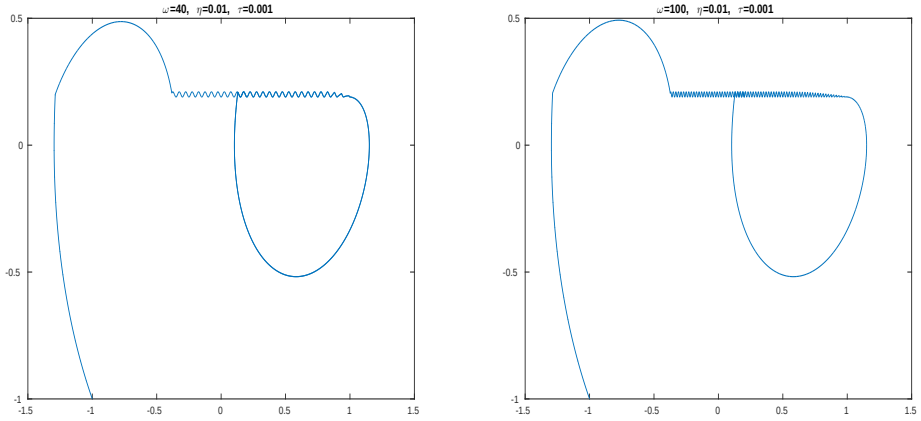

FIGURE 3. Events method. Two different trajectories for $\eta = 0.01$ and $\omega = 40, 100$.

small amplitude periodic function of the spiders' population with respect to a constant threshold.

Below, the variable w indicates the number of insects in the wood, s indicates the number of spiders, and v is the number of insects in the vineyard; also, x indicates the vector $\begin{bmatrix} w \\ s \\ v \end{bmatrix}$.

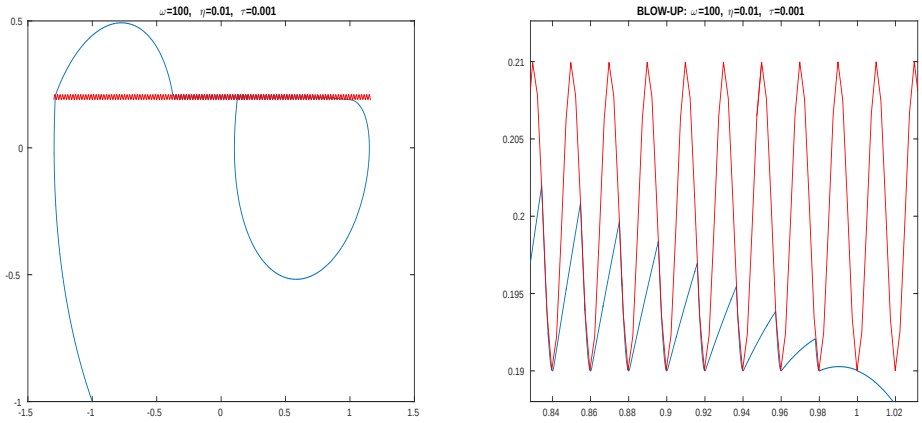

FIGURE 4. Events method. Showing also the surface Σ and a trajectory, when $\eta = 0.01$ and $\omega = 100$. Blow-up on the Right clearly shows repeated exits and re-entries on Σ.

The differential system is of the form in (1.2)*:*

(4.3)
$$\frac{d}{dt}\begin{bmatrix} w \\ s \\ v \end{bmatrix} = \begin{cases} f_- \;=\; \begin{bmatrix} rw\left(1-\frac{w}{W}\right) - csw \\ s\left(-a + \frac{kbv}{H+v} + ckw\right) \\ v\left(g - \frac{bs}{H+v}\right) \end{bmatrix}, & h(x) < 0\ , \\ f_+ \;=\; \begin{bmatrix} rw\left(1-\frac{w}{W}\right) - csw - e(1-q)w \\ s\left(-a + \frac{kbv}{H+v} + ckw\right) - eKqs \\ v\left(g - \frac{bs}{H+v}\right) - eqv \end{bmatrix}, & h(x) > 0\ , \end{cases}$$

where the function $h(x)$ *is*

(4.4)
$$h(x) \;=\; v - \bar{v} + \eta\cos(\omega\pi s)\ ;\ \eta,\ \omega > 0\ ,\quad \bar{v} = 3\ ,$$

and the discontinuity curve is given by $\Sigma := \{x \;:\; h(x) = 0\}$*. We are interested in* η*:* $0 \le \eta \ll 1$*, and* $\omega \gg 1$*.*

In [1]*, the authors do not report on the values of the constants in* (4.3) *that they used, so we used three different sets of values, to highlight different behavior of the solution and different sliding behavior on* Σ*; the third set of values is adapted from* [10]*. In all cases, population units are in kg/ha and time is measured in weeks. The values we used are summarized in Table 6.*

Remark 4.3. *For the set of parameter values # 1, the solution is attracted to* Σ*, and it remains there, exhibiting a decaying oscillatory behavior (seemingly, slowly spiraling toward an equilibrium of the PWS system on* Σ*). For the set of parameters # 2, the solution is attracted to* Σ*, slides on it, then exits on the plane* $x_1 = 0$ *in* R_-*, where it remains, periodically entering and exiting from* Σ*. Finally, for the set of parameters # 3, the solution arrives*

Parameter	Meaning	Set # 1	Set # 2	Set # 3
r	growth rate insects in woods	1	1	1
g	growth rate insects in vineyard	0.1	0.1	0.5
W	carrying capactiy of woods	20	10	1
H	carrying capactiy of vineyard	100	20	7
k	conversion factor insects to spiders	0.5	1	1
a	death rate of spiders without insect	0.2	0.1	0.2
c	birth/death rates due to encounters	1	1	0.24
b	birth/death rates due to encounters	1	1	1.118
q	% insecticide sprayed on vineyard $1-q$ dispersed in woods	0.75	0.75	0.9
e	effectiveness factor	0.8	0.8	0.6
K	effect of spraying on the spiders	0.2	0.2	0.01

TABLE 6. Three sets of parameter values used in our experiments for (4.3).

to Σ and then exits into R_-, periodically returning to Σ. See Figures 5 and 6.

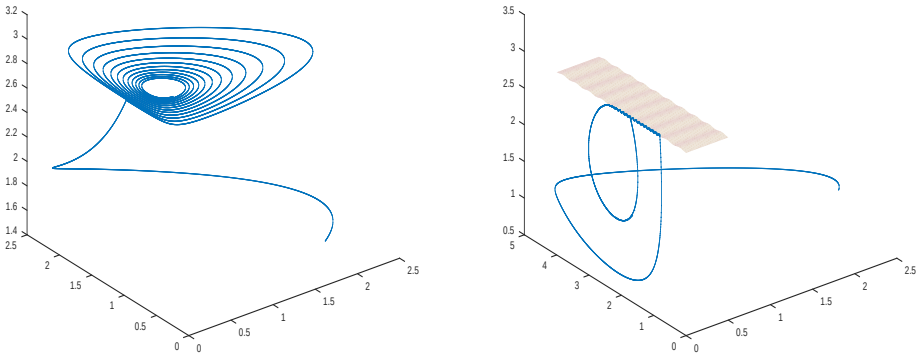

FIGURE 5. Left: Parameter set # 1, and $\eta = 0$. Right: Parameter set # 2, and $\omega = 20$, $\eta = 0.01$. Time interval $[0, 200]$.

For all three sets of parameter values in Table 6, and taking $\eta = 0$ in (4.4), the regularization approach exhibits the same behavior observed in Example 4.1. Namely, it produces spurious oscillatory behavior (artificial chattering) around the sliding surface Σ (here, $v = \bar{v}$ is Σ), all the more visible the smaller is the value of the regularization parameter ϵ. Similar spurious behavior is observed also for $\eta > 0$. At the same time, the regularization approach produces some approximation for all ranges of η and ω we tried,

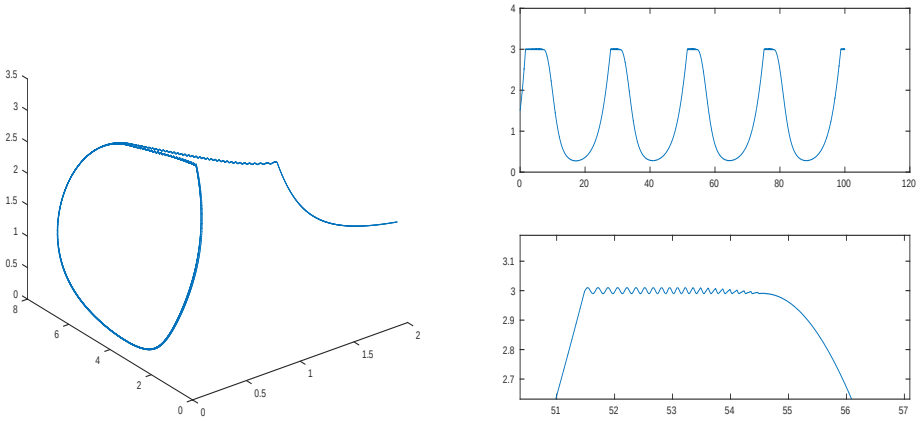

FIGURE 6. Parameter set # 3. On the Right, we show the trajectory for $v(t)$ and a zoom in of the same during sliding. Time interval $[0, 100]$, $\omega = 20$ and $\eta = 0.01$.

Parameter Set	η	ω	CPU Event-No-Project	CPU Event-Project
# 1	0	–	1	1
# 1	0.01	20	1.01	1
# 1	0.01	50	1.05	Fails
# 2	0	–	1.16	1.17
# 2	0.01	20	1.11	1.12
# 2	0.01	50	1.05	Fails
# 2	0.01	100	1	Fails
# 3	0	–	1.04	1.04
# 3	0.01	20	1	1.1
# 3	0.01	50	1.17	1.42

TABLE 7. Example 4.3 for the three different parameters set of Table 6. Time interval is $[0, 200]$ for Sets # 1,2, and $[0, 100]$ for Set # 3. IC $(2, 1/2, 3/2)$, stepsize $\tau = 10^{-3}$.

and its execution time (recall that we are using fixed stepsizes) is effectively independent of the values of η, ω. This robustness is undoubtedly its main merit.

When they both complete execution, the other two approaches (sliding with and without projection) behave closely to one another, although the projection approach fails more frequently than the unprojected one. Some results are summarized in Table 7, where the times are normalized across each set of parameter values.

5. Conclusions

In this work, we proposed and implemented an event based technique which avoids projection when integrating a PWS system, with a co-dimension 1 singularity manifold, during sliding motion. We further compared this approach to one that performs projection, and to one that regularizes the system. Of course, we had to make some algorithmic choices, the most important being what basic integration scheme to use: we opted for explicit RK schemes in fixed stepsize mode. We made also other choices which may have impacted our study; e.g., how to perform the projection (we used a stationary Newton's iteration in the Lagrange multipliers formulation), and how to locate events (we used a simple bisection method), and of course how many iterations we allowed and the tolerance value used to declare convergence. There are implications/limitations of all these choices, most relevant being that failure during projection or events' search leads to a halt, rather than a stepsize reduction. All of these considerations notwithstanding, we believe that some conclusions unambigously emerged. First, the events' method without projection outperformed that with projection, both in terms of execution time and of robustness. Second, the regularization approach showed some instabilities around the discontinuity surface, and it introduces very large gradients in the solution, which makes it an iffy candidate to faithfully recover the dynamics of the PWS system; at the same time, the regularization approach is largely immune by having to make algorithmic choices typical of the events methods (there is no projection or events' location). Thus, given its overall simplicity, the regularization method will probably remain a method of choice for several people.

Future extensions of the present work must include comparison with other choices for integrators and with variable stepsize implementations. Also of interest will be to see what can be said/done when the discontinuity manifold is of higher co-dimension, say 2, at least for as long as sliding motion in this case is well defined.

References

[1] M. Biák, T. Hanus, D. Janovská. Some applications of Filippovs dynamical systems. Journal of Computational and Applied Mathematics, 254 (2013), pp. 132-143.

[2] M. Calvo and J. Montijano and L. Rández. Algorithm 968. Disode45: A Matlab Runge-Kutta solver for piecewise smooth IVPs of Filippov type. ACM Transactions on Mathematical Software, 43-3 (2016), pp. 1-14.

[3] L. Dieci and L. Lopez. Sliding motion in Filippov differential systems: Theoretical results and a computational approach. SIAM Journal of Numerical Analysis. 47-3 (2009), pp 2023–2051.

[4] L. Dieci and L. Lopez. A survey of numerical methods for IVPs of ODEs with discontinuous right-hand side. Journal of Computational and Applied Mathematics, 236 (2012), pp. 3967–3991.

[5] L. Dieci and L. Lopez. One-sided direct event location techniques in the numerical solution of discontinuous differential systems. BIT Numerical Mathematics, 55-4 (2015), pp. 987-1003.

[6] A.F. Filippov. *Differential Equations with Discontinuous Right-Hand Sides*. Mathematics and Its Applications, Kluwer Academic, Dordrecht, 1988.
[7] U. Galvanetto and S.R. Bishop. Dynamics of a simple damped oscillator undergoing stick-slip vibrations. Meccanica v. 34 (1999), pp. 337–347.
[8] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*, 2nd Edition. Springer series in Computational Mathematics. Springer-Verlag, Berlin-Heidelberg, 1996.
[9] J. Sotomayor and M.A. Teixeira. Regularization of discontinuous vector fields, Proceedings of the International Conference on Differential Equations, Lisboa, (1996), pp. 207–223.
[10] E. Venturino, M. Isaia, F. Bona, S. Chatterjee and G. Badino. Biological controls of intensive agroecosystems: wanderer spiders in the langa astigiana. Ecological Complexity v. 5 (2008), pp. 157–164.

School of Mathematics, Georgia Tech, Atlanta, GA 30332 U.S.A.
E-mail address: dieci@math.gatech.edu