Article

# First Order and Second Order Learning Algorithms on the Special Orthogonal Group to Compute the SVD of Data Matrices 

Simone Fiori ${ }^{1, *}$ (D) Lorenzo Del Rossi ${ }^{2}$, Michele Gigli ${ }^{3}$ and Alessio Saccuti ${ }^{4}$<br>1 Department of Information Engineering, Università Politecnica delle Marche, Via Brecce Bianche, 60131 Ancona, Italy<br>2 Graduate School of Computer Engineering, Università di Modena e Reggio Emilia, Via Università 4, 41121 Modena, Italy; S1080698@studenti.univpm.it<br>3 Graduate School of Computer Science and Automation Engineering, Università Politecnica delle Marche, Via Brecce Bianche, 60131 Ancona, Italy; S1076685@studenti.univpm.it<br>4 Graduate School of Computer Science, Università di Parma, Via Università 12, 43121 Parma, Italy; S1077339@studenti.univpm.it<br>* Correspondence: s.fiori@univpm.it

Received: 16 December 2019; Accepted: 11 February 2020; Published: 15 February 2020


#### Abstract

The present paper deals with neural algorithms to learn the singular value decomposition (SVD) of data matrices. The neural algorithms utilized in the present research endeavor were developed by Helmke and Moore (HM) and appear under the form of two continuous-time differential equations over the special orthogonal group of matrices. The purpose of the present paper is to develop and compare different numerical schemes, under the form of two alternating learning rules, to learn the singular value decomposition of large matrices on the basis of the HM learning paradigm. The numerical schemes developed here are both first-order (Euler-like) and second-order (Runge-like). Moreover, a reduced Euler scheme is presented that consists of a single learning rule for one of the factors involved in the SVD. Numerical experiments performed to estimate the optical-flow (which is a component of modern IoT technologies) in real-world video sequences illustrate the features of the novel learning schemes.


Keywords: singular value decomposition; initial value problem; first-order numerical method; second-order numerical method; manifold calculus and Lie groups; learning system; Internet of Things

## 1. Introduction

The computation of the singular value decomposition (SVD) of a non-square matrix [1-3] plays a key role in a number of applications (see, for instance, [4-13]); among them, it is worth citing applications in automatic control [14], digital circuit design [15], time-series prediction [16], and image processing $[17,18]$. Efforts have been devoted to achieving the computation of the SVD of matrices in the artificial neural networks community [19-21].

The neural algorithms utilized in the present research work to learn an SVD of a data matrix were developed by Helmke and Moore (HM) in [1]. An HM learning rule appeared under the form of two continuous-time differential equations over the special orthogonal group of matrices. The purpose of the present research is to develop and compare different numerical schemes, under the form of two alternating learning equations, to learn the SVD of large matrices on the basis of the HM learning paradigm. The numerical schemes developed here are both first-order (Euler-like) and second-order
(Runge-like). Moreover, a reduced Euler scheme will be presented that consists of a single learning equation for one of the factors involved in the SVD.

The developed methods to learn an SVD of a data matrix are applied to optical-flow estimation. Optical flow estimation is a well-known image-processing operation that allows estimating the motion of portions of an image over a video sequence and has found widespread applications (see, for instance, [22-29]). Optical flow is closely related to motion estimation [30]. Optical flow refers to the change of structured light in an image and captures such change through a velocity vector field. Most of the optical-flow estimation algorithms used in video encoding belong to either the class of block matching algorithms (BMAs) or to the class of pixel recursive algorithms (PRAs) [31].

Optical flow estimation algorithms are essential components in a number of complex Internet of Things (IoT) technologies, as testified by several existing applications, such as intelligent fall detection [32,33], mobile robotics [34], intelligent flight monitoring [35], mobile object tracing [36], automated surveillance [37], smart healthcare [38], solar energy forecasting [39], and in IoT-based analytics in urban space, shops, and retail stores to inform policy makers, shop owners, and the general public about how they interact with the physical space [40]. See also the interesting discussion in [41].

The majority of the current optical-flow estimation methods rely on a block-matching algorithm. The BMA methods are based on the concept of template-matching: it is supposed that a single block in a time-frame has moved solidly to another location in the next time-frame, so the image-block is regarded as a template to be looked for in the subsequent frame. The BMA methods try to evaluate the motion of a block by reducing the number of search locations in the search range and/or by reducing the number of computations at each search location. These algorithms are either ad hoc or are based on the assumption that the error increases monotonically from the best-match location. However, typically, the error surface may exhibit local minima, and the majority of the BMAs get trapped in one of the local minima depending on the starting point and on the search direction. Furthermore, the matching algorithms aim at finding the best match with respect to some selected mismatch (error) measure, but the best match may not represent the true motion [42].

Conversely, standard PRAs try to estimate the motion at each pixel. In the method for optical-flow estimation considered here, based on the paper [42], a methodology similar to the PRA is employed, except that the algorithm operates on a pixel-block basis and finds a single motion-vector for each block. In order to make the method robust in a noisy environment, a total least squares (TLS) estimation approach is invoked. Total least squares may be implemented numerically through singular value decomposition (SVD). Other matrix decomposition techniques, such as eigenvalue decomposition, have been used in the scientific literature [43].

Throughout the paper, we use the following notation: symbol $I_{m, p}$ denotes a pseudo-identity matrix of size $m \times p$ and $I_{m}=I_{m, m}$, while symbol $0_{m, p}$ denotes a $m \times p$ all-zero matrix, symbol $X^{\top}$ denotes the transpose of the matrix $X$, and symbol $\operatorname{tr}(X)$ denotes the trace of the square matrix $X$, i.e., the sum of its in-diagonal entries. The special orthogonal group is denoted and defined by $\operatorname{SO}(m):=\left\{X \in \mathbb{R}^{m \times m} \mid X^{\top} X=I_{m}, \operatorname{det}(X)=1\right\}$. For details on this Lie group, see, e.g., [44]. Furthermore, the Frobenius norm of a matrix $X \in \mathbb{R}^{p \times p}$ is denoted and defined by $\|X\|_{\mathrm{F}}:=\sqrt{\operatorname{tr}\left(X^{\top} X\right)}$.

## 2. Helmke-Moore Learning and Advanced Numerical Methods

In the present section, we recall some essential details about the Helmke and Moore (HM) learning algorithm from [1,45]. Moreover, we survey a first-order and two second-order numerical methods to implement a learning paradigm formulated as an initial value problem (IVP, also termed the Cauchy problem) and propose numerical schemes to extend these methods to Riemannian manifolds, of which Lie groups are instances.

There exists a large variety of numerical methods to solve initial value problems. In order to limit the computational burden of the learning rules considered in this paper, we considered a Euler method, a Heun method, and a Runge method (also known as the Runge-Kutta method of the second
order). These methods were developed to solve initial values problems over the Euclidean space $\mathbb{R}^{m}$, although they can be adapted to solve initial value problems over smooth manifolds or Lie groups, as the group $\mathrm{SO}(m)$.

### 2.1. An HM-Type Neural SVD Learning Algorithm

Denoting as $Z \in \mathbb{R}^{m \times p}$ a matrix whose SVD is to be computed and as $r \leq \min \{m, p\}$ the rank of $Z$, its singular value decomposition may be written $Z=U D V^{\top}$, where $U \in \mathbb{R}^{m \times m}$ and $V \in \mathbb{R}^{p \times p}$ are (special) orthogonal matrices and $D$ is a pseudo-diagonal matrix that has all-zero values except for the first $r$ diagonal entries, termed singular values. It is easily checked that the columns of $U$ coincide with the eigenvectors of the product $\mathrm{ZZ}^{\top}$, while $V$ contains the eigenvectors of the self-product $Z^{\top} Z$, with the same eigenvalues.

The Helmke-Moore algorithm is utilized for training, in an unsupervised way, an artificial neural network to learn an SVD of a given rectangular matrix. The HM dynamics arises from the maximization of a specific criterion $\phi_{W}: \mathrm{SO}(m) \times \mathrm{SO}(m) \rightarrow \mathbb{R}$ defined as:

$$
\begin{equation*}
\phi_{W}(A, B):=2 \operatorname{tr}\left(W A^{\top} Z B\right) \tag{1}
\end{equation*}
$$

where $W \in \mathbb{R}^{p \times m}$ is a weighting kernel and we assumed that $m \geqslant p$. The dynamical system, derived as a Riemannian gradient flow on $\mathrm{SO}(m) \times \mathrm{SO}(m)$, reads:

$$
\left\{\begin{array}{l}
H:=A^{\top} Z B  \tag{2}\\
\dot{A}=A\left(W^{\top} H^{\top}-H W\right), A(0)=A_{0} \in \mathrm{SO}(m) \\
\dot{B}=B\left(W H-H^{\top} W^{\top}\right), B(0)=B_{0} \in \mathrm{SO}(m)
\end{array}\right.
$$

where an over-dot denotes derivation with respect to the time parameter. The weighting matrix $W$ has the structure [ $W_{1} 0_{p, m-p}$ ], where $W_{1} \in \mathbb{R}^{p \times p}$ must be diagonal with unequal entries on the main diagonal [45].

Since the matrix $A$ has size $m \times m$, the matrix $Z$ has size $m \times p$, and the matrix $B$ has size $p \times p$, then the product matrix $H$ has size $m \times p$.

Whereas the continuous-time versions of the learning algorithms leave the orthogonal group invariant, this is not true for their discrete-time counterparts, which are obtained by employing a numerical integration scheme, unless a suitable integration method is put into effect. In the present case, we may employ a convenient Lie integration method drawn from a manifold-calculus-based integration theory (see, e.g., the contribution [46] and previous reviews in [47-49]).

### 2.2. Euler Method in $\mathbb{R}^{m}$ and Its Extension to $\mathrm{SO}(m)$

Consider the following initial value problem:

$$
\begin{equation*}
\dot{y}=f(t, y) \tag{3}
\end{equation*}
$$

with $f: \mathbb{R} \times \mathbb{R}^{m} \rightarrow \mathbb{R}^{m}$ being a regular function and with the initial condition $y(0)=y_{0} \in \mathbb{R}^{m}$. Whenever a closed-form solution to the IVP (3) is out of reach, the simplest numerical scheme to approximate numerically its solution is the forward Euler method described by the iterative rule:

$$
\begin{equation*}
y_{n+1}=y_{n}+h f\left(t_{n}, y_{n}\right), n \geqslant 0 \tag{4}
\end{equation*}
$$

with $y_{0}$ known from the initial condition. As a reference for this method and those invoked in the continuation of the paper, readers might refer to [50]. The constant $h>0$ denotes an integration stepsize and represents the time lapse between each time-node $t_{n}=n h$ and the next. The key idea behind the above forward Euler method is to estimate the value $y_{n+1}$ by the slope of the vector field $f(t, y)$ at the present node through a linear interpolation across the time lapse $h$. This Euler method
is asymmetric: it makes use only of the value of the vector field $f$ computed in the leftmost point of the interval $\left[t_{n}, t_{n+1}\right]$. The right-hand side of the expression (4) coincides with the zeroth and the first terms of the Taylor series expansion of the function $f$. The residue owing to truncation (also termed "truncation error") is of type $\mathscr{O}\left(h^{2}\right)$; therefore, this method is of order one.

A learning problem on the special orthogonal group is described by the IVP:

$$
\begin{equation*}
\dot{Y}=f(t, Y), f: \mathbb{R} \times \mathrm{SO}(m) \rightarrow \mathrm{SO}(m) \tag{5}
\end{equation*}
$$

with the initial condition $Y(0)=Y_{0} \in \mathrm{SO}(m)$.
Let us recall that a smooth manifold $\mathbb{M}$ may be endowed with a manifold exponential map and by a parallel transport operator:

- Manifold exponential: The exponential map derived from the geodesic associated with the connection is denoted by $\exp : T \mathbb{M} \rightarrow \mathbb{M}$. Given $x \in \mathbb{M}$ and $v \in T_{x} \mathbb{M}$, the exponential of $v$ at a point $x$ is denoted as $\exp _{x}(v)$. On a Euclidean manifold $\mathbb{M}=\mathbb{R}^{m}, \exp _{x}(v)=x+v$; therefore, an exponential map may be thought of as a generalization of "vector addition" to a curved space.
- Parallel transport along geodesics: The parallel transport map is denoted by $\mathrm{P}: \mathbb{M} \times T \mathbb{M} \rightarrow$ $T \mathbb{M}$. Given two points $x, y \in \mathbb{M}$, parallel transport from $x$ to $y$ is denoted as $\mathrm{P}^{x \rightarrow y}: T_{x} \mathbb{M} \rightarrow$ $T_{y} \mathbb{M}$. Parallel transport moves a tangent vector between two points along their connecting geodesic arc. Given $x, y \in \mathbb{M}$ and $v \in T_{x} \mathbb{M}$, the parallel transport of a vector $v$ from a point $x$ to a point $y$ is denoted by $\mathrm{P}^{x \rightarrow y}(v)$. Parallel transport does not change the length of a transported vector, namely it is an isometry; moreover, parallel transport does not alter the angle between transported vectors, namely, given $x, y \in \mathbb{M}$ and $a, v \in T_{x} \mathbb{M}$, it holds that $\left\langle\mathrm{P}^{x \rightarrow y}(a), \mathrm{P}^{x \rightarrow y}(v)\right\rangle_{y}=$ $\langle a, v\rangle_{x}$, i.e., we say that parallel transport is a conformal map. Formally, on a Euclidean manifold $\mathbb{M}=\mathbb{R}^{m}, \mathrm{P}^{x \rightarrow y}(v)=v$; therefore, a vector transport may be thought of as a generalization of the familiar geometric notion of the "rigid translation" of vectors to a curved space.

In the case of a special orthogonal group endowed with its canonical metric (inherited by the Euclidean metric), these maps read:

$$
\left\{\begin{array}{l}
\exp _{X}(V):=X \operatorname{Exp}\left(X^{\top} V\right),(\text { Exponential map })  \tag{6}\\
\mathrm{P}^{X \rightarrow Y}(V):=X \sqrt{X^{\top} Y} X^{\top} V \sqrt{X^{\top} Y}, \text { (Parallel transport operator) }
\end{array}\right.
$$

where $X, Y \in \mathrm{SO}(m), V \in T_{X} \mathrm{SO}(m)$, "Exp" denotes a matrix exponential (implemented, for example, by the function expm in MATLAB ${ }^{\circledR}$ ), $\sqrt{\cdot}$ denotes a matrix square-root (implemented, for example, by the function sqrtm in MATLAB $\left.{ }^{\circledR}\right)$. Let us recall that a matrix $V \in T_{X} \mathrm{SO}(m)$ may be rewritten as $V=X \Omega$, with $\Omega$ skew-symmetric and, therefore, that the following simplified expression for the exponential map may be used:

$$
\begin{equation*}
\exp _{X}(X \Omega)=X \operatorname{Exp}(\Omega) \tag{7}
\end{equation*}
$$

A rule of thumb to extend the forward Euler method to smooth manifolds is that the sum between a variable representing a point and a vector field at that point needs to be replaced by the exponential map applied to the point and to the vector field at that point. As a result, a Euler method on a manifold $\mathrm{SO}(m)$ may be expressed as:

$$
\begin{equation*}
Y_{n+1}=\exp _{Y_{n}}\left(h f\left(t_{n}, Y_{n}\right)\right), \tag{8}
\end{equation*}
$$

with $Y_{0}$ known from the initial condition. Likewise, the original forward Euler method, the method (8), estimates the value of the solution at the temporal node $t_{n+1}$, namely $Y_{n+1}$, as the value of the solution at the node $t_{n}$ interpolated by the short geodesic arc departing from the current point $Y_{n}$ toward the direction indicated by the vector field $f\left(t_{n}, Y_{n}\right)$.

### 2.3. Heun and Runge Methods in $\mathbb{R}^{m}$ and Their Extension to $\mathrm{SO}(m)$

The following methods are second-order numerical schemes that may be adopted to solve the Cauchy problem (3).

Heun's method reads:

$$
\left\{\begin{array}{l}
k_{1, n}:=f\left(t_{n}, y_{n}\right)  \tag{9}\\
\tilde{y}_{n+1}:=y_{n}+h k_{1, n} \\
k_{2, n}:=f\left(t_{n}+h, \tilde{y}_{n+1}\right) \\
y_{n+1}=y_{n}+\frac{h}{2}\left(k_{1, n}+k_{2, n}\right)
\end{array}\right.
$$

where $h>0$ denotes again a stepsize (or time lapse between two adjacent time nodes). It is worth summarizing an interpretation of the equations that describe the Heun method: The quantity $k_{1}$ represents an estimation of the value of the vector field $f$ at the leftmost point of the time lapse [ $\left.t_{n}, t_{n}+h\right]$, while the quantity $k_{2}$ represents an estimation of the value of the vector field $f$ at the rightmost point of the time lapse $\left[t_{n}, t_{n}+h\right]$. Both $k_{1}$ and $k_{2}$ are estimations, in that the expression for the quantity $k_{1, n}$ uses the value $y_{n}$ that is an estimation of the solution of the Cauchy problem obtained in the previous iteration, while the expression for $k_{2, n}$ utilizes an estimation of $y_{n+1}$, indicated by $\tilde{y}_{n+1}$, based on a linear interpolation from $y_{n}$ in the direction $k_{1, n}$. The actual estimation of the solution at $t_{n}+h$ is obtained as a linear interpolation in a direction computed as the arithmetic average between the directions $k_{1}$ and $k_{2}$.

Runge's method (often denoted as "RK2") is simpler than Heun's, although presenting the same precision order, and is expressed as:

$$
\left\{\begin{array}{l}
k_{1, n}:=f\left(t_{n}, y_{n}\right)  \tag{10}\\
y_{n+1 / 2}:=y_{n}+\frac{h}{2} k_{1, n} \\
k_{2, n}:=f\left(t_{n}+\frac{h}{2}, y_{n+1 / 2}\right), \\
y_{n+1}=y_{n}+h k_{2, n}
\end{array}\right.
$$

The quantity $k_{1}$ represents again an estimation of the value of the vector field $f$ at the leftmost point of the time lapse $\left[t_{n}, t_{n}+h\right]$, while the quantity $k_{2}$ represents an estimate of the value of the vector field $f$ at the midpoint of the time lapse $\left[t_{n}, t_{n}+h\right]$. Both $k_{1}$ and $k_{2}$ are estimations, rather than being exact evaluations. In particular, the expression for $k_{2, n}$ utilizes an estimation of the exact solution to the Cauchy problem, denoted by $\tilde{y}_{n+1 / 2}$, based on a linear interpolation from $y_{n}$ in the direction $k_{1, n}$ extended only to half of a whole step. The actual estimation of the solution at $t_{n}+h$ is obtained as a linear interpolation, from $y_{n}$ in a direction $k_{2}$ extended to a whole step.

The above second-order numerical methods may be extended to a smooth manifold by recalling two simple rules-of-thumb:

- The sum between a variable representing a point on a manifold and a vector field at that point are replaced by the exponential map applied to the point and to the vector field at that point.
- The sum between two tangent vectors belonging to two different tangent spaces may be carried out only upon transporting one of the vectors to the tangent space where the other vector lives by means of parallel transport.

By applying the above rules, it is possible to extend the Heun method and the Runge method from the Euclidean space $\mathbb{R}^{m}$ to the manifold $\mathrm{SO}(m)$.

Heun's method on $\mathrm{SO}(m)$ may be expressed as:

$$
\left\{\begin{array}{l}
K_{1, n}:=f\left(t_{n}, Y_{n}\right) \in T_{Y_{n}} \mathrm{SO}(m),  \tag{11}\\
\tilde{Y}_{n+1}:=\exp _{Y_{n}}\left(h K_{1, n}\right), \\
K_{2, n}:=f\left(t_{n}+h, \tilde{Y}_{n+1}\right) \in T_{\tilde{\Upsilon}_{n+1}} \mathrm{SO}(m), \\
Y_{n+1}=\exp _{Y_{n}}\left(\frac{h}{2}\left(K_{1, n}+\mathrm{P}^{\tilde{Y}_{n+1} \rightarrow Y_{n}}\left(K_{2, n}\right)\right)\right)
\end{array}\right.
$$

The above extension of the original Heun method was derived by following faithfully Heun's concept and by replacing the linear operations with manifold operations. In particular, notice that the mean direction between $K_{1}$ and $K_{2}$ cannot be calculated as $\frac{1}{2}\left(K_{1}+K_{2}\right)$ because $K_{1}$ and $K_{2}$ belong to different tangent spaces.

Runge's method on $\mathrm{SO}(m)$ may be expressed as:

$$
\left\{\begin{array}{l}
K_{1, n}:=f\left(t_{n}, Y_{n}\right) \in T_{Y_{n}} \mathrm{SO}(m)  \tag{12}\\
Y_{n+1 / 2}:=\exp _{Y_{n}}\left(\frac{h}{2} K_{1, n}\right), \\
K_{2, n}:=f\left(t_{n}+\frac{h}{2}, Y_{n+1 / 2}\right) \in T_{Y_{n+1 / 2}} \mathrm{SO}(m) \\
Y_{n+1}=\exp _{Y_{n}}\left(h \mathrm{P}_{n+1 / 2} \rightarrow Y_{n}\left(K_{2, n}\right)\right)
\end{array}\right.
$$

The interpretation of the equations that constitute Runge's method on the manifold $\mathrm{SO}(m)$ is completely faithful to the original method.

In the following subsection, the forward Euler method, the Heun method, and the Runge method are applied to solve the HM learning system of IVPs (2). This application leads to facing at least two challenges: (1) the HM system is made of two differential equations, which entails the application of each numerical method twice; this gives rise to a non-univocality in the extension of the equations, which will then be presented in different versions; (2) the curved space $\mathrm{SO}(m)$ may be treated as a smooth Riemannian manifold and as a Lie group; this gives rise to two different ways to design numerical methods, which will be explored and discussed in the following, even by the help of preliminary numerical tests.

### 2.4. Application of the Euler, Heun, and Runge Method to Solving an HM System

The HM-type learning system to solve is formed by two coupled neural learning equations, namely:

$$
\left\{\begin{array}{l}
\dot{A}=-A \sigma\left(A^{\top} Z B W\right)  \tag{13}\\
\dot{B}=B \sigma\left(W A^{\top} Z B\right)
\end{array}\right.
$$

where the skew-symmetrization operator $\sigma(X):=X^{\top}-X$ has been introduced and where the initial conditions $A(0)=A_{0}$ and $B(0)=B_{0}$ have been fixed.

In the following, we shall outline three different classes of numerical methods to tackle the learning problem (13), namely the (forward) Euler class, the (explicit second-order) Heun method, and the (explicit) Runge method (a second-order instance from the general class of Runge-Kutta methods). For the sake of notation compactness, in the following, we shall make use of the twin vector fields:

$$
\begin{equation*}
f^{A}(A, B):=-A \sigma\left(A^{\top} Z B W\right), f^{B}(A, B):=B \sigma\left(W A^{\top} Z B\right) \tag{14}
\end{equation*}
$$

The numerical schemes adopted in this study are first- and second-order explicit. Implicit methods are overly complex, as they require solving a non-linear problem at every iteration (the computations involved in the implicit methods may be as complex as the problem they were designed to solve). Higher order formulations (such as the fourth-order Runge-Kutta method) are often inappropriate in optimization, because they were designed to provide a highly accurate estimation of the solution to
an initial value problem at every step, while in optimization, such accuracy would be out of the scope as the only step that needs accuracy is the final optimization goal.

### 2.4.1. Forward Euler Method

The Euler method on $\mathrm{SO}(m)$ outlined in Section 2.2, applied to solve an HM learning system, may be expressed as:

$$
\text { Euler: }\left\{\begin{array}{l}
A_{n+1}=\exp _{A_{n}}\left(h^{A} f^{A}\left(A_{n}, B_{n}\right)\right),  \tag{15}\\
B_{n+1}=\exp _{B_{n}}\left(h^{B} f^{B}\left(A_{n}, B_{n}\right)\right),
\end{array}\right.
$$

with $h^{A}>0$ and $h^{B}>0$ being two different learning stepsizes.

### 2.4.2. Explicit Second-Order Heun Method

The Heun method outlined in Section 2.3, applied to an HM learning problem on $\mathrm{SO}(m)$, as a manifold, may be expressed as:

$$
\text { Heun, Version 1: }\left\{\begin{array}{l}
K_{1, n}^{A}:=f^{A}\left(A_{n}, B_{n}\right),  \tag{16}\\
\tilde{A}_{n+1}:=\exp _{A_{n}}\left(h^{A} K_{1, n}^{A}\right), \\
K_{2, n}^{A}:=f^{A}\left(\tilde{A}_{n+1}, B_{n}\right), \\
A_{n+1}=\exp _{A_{n}}\left(\frac{h^{A}}{2}\left(K_{1, n}^{A}+\mathrm{P}^{\tilde{A}_{n+1} \rightarrow A_{n}}\left(K_{2, n}^{A}\right)\right)\right), \\
K_{1, n}^{B}:=f^{B}\left(A_{n}, B_{n}\right), \\
\tilde{B}_{n+1}:=\exp _{B_{n}}\left(h^{B} K_{1, n}^{B}\right), \\
K_{2, n}^{B}:=f^{B}\left(A_{n}, \tilde{B}_{n+1}\right), \\
B_{n+1}=\exp _{B_{n}}\left(\frac{h^{B}}{2}\left(K_{1, n}^{B}+\mathrm{P}^{\tilde{B}_{n+1} \rightarrow B_{n}}\left(K_{2, n}^{B}\right)\right)\right)
\end{array}\right.
$$

It is interesting to observe that the same numerical learning algorithm may be recast in a number of slightly different ways by operating some tiny variations on the equations. An alternative version to the original Heun, Version 1 (16) is:

$$
\text { Heun, Version 2: }\left\{\begin{array}{l}
K_{1, n}^{A}:=f^{A}\left(A_{n}, B_{n}\right),  \tag{17}\\
\tilde{A}_{n+1}:=\exp _{A_{n}}\left(h^{A} K_{1, n}^{A}\right), \\
K_{2, n}^{A}:=f^{A}\left(\tilde{A}_{n+1}, B_{n}\right), \\
A_{n+1}=\exp _{A_{n}}\left(\frac{h^{A}}{2}\left(K_{1, n}^{A}+\mathrm{P}^{\tilde{A}_{n+1} \rightarrow A_{n}}\left(K_{2, n}^{A}\right)\right)\right), \\
K_{1, n}^{B}:=f^{B}\left(\underline{\left.A_{n+1}, B_{n}\right)},\right. \\
\tilde{B}_{n+1}:=\exp _{B_{n}}\left(h^{B} K_{1, n}^{B}\right), \\
K_{2, n}^{B}:=f^{B}\left(\underline{A_{n+1}}, \tilde{B}_{n+1}\right), \\
B_{n+1}=\exp _{B_{n}}\left(\frac{h^{B}}{2}\left(K_{1, n}^{B}+\tilde{P}^{\tilde{B}_{n+1} \rightarrow B_{n}}\left(K_{2, n}^{B}\right)\right)\right),
\end{array}\right.
$$

where the differences have been highlighted. Notice that both versions are explicit numerical schemes.
From a computational burden viewpoint, it is important to underline that the above numerical algorithms require the repeated calculation of parallel transport that contributes non-negligibly to increasing the computational complexity of these methods.

As an alternative, the Heun method may be rewritten without using parallel transport by composing the partial steps in a different way over the space $\mathrm{SO}(m)$, namely:

$$
\text { Heun, Version 3: }\left\{\begin{array}{l}
K_{1, n}^{A}:=f^{A}\left(A_{n}, B_{n}\right),  \tag{18}\\
\tilde{A}_{n+1}:=\exp _{A_{n}}\left(h K_{1, n}^{A}\right), \\
K_{2, n}^{A}:=f^{A}\left(\tilde{A}_{n+1}, B_{n}\right), \\
A_{n+1}=A_{n} \cdot \tilde{A}_{n+1}^{-1} \exp _{\tilde{A}_{n+1}}\left(\frac{h}{2} K_{2, n}^{A}\right) \cdot A_{n}^{-1} \exp _{A_{n}}\left(\frac{h}{2} K_{1, n}^{A}\right), \\
K_{1, n}^{B}:=f^{B}\left(\underline{\left.A_{n+1}, B_{n}\right),}\right. \\
\tilde{B}_{n+1}:=\exp _{B_{n}}\left(h K_{1, n}^{B}\right), \\
K_{2, n}^{B}:=f^{B}\left(\underline{\left.A_{n+1}, \tilde{B}_{n+1}\right),}\right. \\
B_{n+1}=B_{n} \cdot \tilde{B}_{n+1}^{-1} \exp _{\tilde{B}_{n+1}}\left(\frac{h}{2} K_{2, n}^{B}\right) \cdot B_{n}^{-1} \exp _{B_{n}}\left(\frac{h}{2} K_{1, n}^{B}\right) .
\end{array}\right.
$$

The differences with the Heun method, Version 1 (16), have been highlighted. It is interesting to observe that, by swapping the order of the steps in $A_{n+1}$ e $B_{n+1}(1 \leftrightarrow 2)$, we get:

$$
\left\{\begin{array}{l}
A_{n+1}=A_{n}-A_{n}^{-1} \exp _{A_{n}}\left(\frac{h}{2} K_{1, n}^{A}\right) \cdot \tilde{A}_{n+1}^{-1} \cdot \exp _{\tilde{A}_{n+1}}\left(\frac{h}{2} K_{2, n}^{A}\right),  \tag{19}\\
B_{n+1}=B_{n}-B_{n}^{-1} \exp _{B_{n}}\left(\frac{h}{2} K_{1, n}^{B}\right) \cdot \tilde{B}_{n+1}^{-1} \cdot \exp _{\tilde{B}_{n+1}}\left(\frac{h}{2} K_{2, n}^{B}\right),
\end{array}\right.
$$

which is computationally lighter compared to the Heun method, Version 3 (18). It is immediate to verify that Version 3 of the Heun method, further modified by the above equations, will result in being computationally lighter than the versions requiring parallel transport; therefore, the Heun method considered in the following preliminary numerical test will be:

$$
\text { Heun, Version 4: }\left\{\begin{array}{l}
K_{1, n}^{A}:=f^{A}\left(A_{n}, B_{n}\right),  \tag{20}\\
\tilde{A}_{n+1}:=\exp _{A_{n}}\left(h K_{1, n}^{A}\right), \\
K_{2, n}^{A}:=f^{A}\left(\tilde{A}_{n+1}, B_{n}\right), \\
A_{n+1}=\exp _{A_{n}}\left(\frac{h}{2} K_{1, n}^{A}\right) \cdot \tilde{A}_{n+1}^{-1} \cdot \exp _{\tilde{A}_{n+1}}\left(\frac{h}{2} K_{2, n}^{A}\right), \\
K_{1, n}^{B}:=f^{B}\left(A_{n+1}, B_{n}\right), \\
\tilde{B}_{n+1}:=\exp _{B_{n}}\left(h K_{1, n}^{B}\right), \\
K_{2, n}^{B}:=f^{B}\left(A_{n+1}, \tilde{B}_{n+1}\right), \\
B_{n+1}=\exp _{B_{n}}\left(\frac{h}{2} K_{1, n}^{B}\right) \cdot \tilde{B}_{n+1}^{-1} \cdot \exp _{\tilde{B}_{n+1}}\left(\frac{h}{2} K_{2, n}^{B}\right),
\end{array}\right.
$$

### 2.4.3. Explicit Second-Order Runge Method

The Runge method outlined in Section 2.3, applied to an HM learning problem on $\mathrm{SO}(m)$, as a manifold, may be expressed as:

$$
\text { Runge, Version 1: }\left\{\begin{array}{l}
K_{1, n}^{A}:=f^{A}\left(A_{n}, B_{n}\right), \\
A_{n+1 / 2}:=\exp _{A_{n}}\left(\frac{h^{A}}{2} K_{1, n}^{A}\right),  \tag{21}\\
K_{2, n}^{A}:=f^{A}\left(A_{n+1 / 2}, B_{n}\right), \\
A_{n+1}=\exp _{A_{n}}\left(h^{A} \mathrm{P}_{n+1 / 2} \rightarrow A_{n}\left(K_{2, n}^{A}\right)\right), \\
K_{1, n}^{B}:=f^{B}\left(A_{n}, B_{n}\right), \\
B_{n+1 / 2}:=\exp _{B_{n}}\left(\frac{h^{B}}{2} K_{1, n}^{B}\right), \\
K_{2, n}^{B}:=f^{B}\left(A_{n}, B_{n+1 / 2}\right), \\
B_{n+1}=\exp _{B_{n}}\left(h^{B} \mathrm{P}^{B_{n+1 / 2} \rightarrow B_{n}}\left(K_{2, n}^{B}\right)\right) .
\end{array}\right.
$$

The same method may be expressed in a slightly different way, namely by exchanging the order of adaptation of the variables $A$ and $B$ :

$$
\text { Runge, Version 2: }\left\{\begin{array}{l}
K_{1, n}^{B}:=f^{B}\left(A_{n}, B_{n}\right)  \tag{22}\\
B_{n+1 / 2}:=\exp _{B_{n}}\left(\frac{h^{B}}{2} K_{1, n}^{B}\right), \\
K_{2, n}^{B}:=f^{B}\left(A_{n}, B_{n+1 / 2}\right), \\
B_{n+1}=\exp _{B_{n}}\left(h^{B} \mathrm{P}^{B_{n+1 / 2} \rightarrow B_{n}}\left(K_{2, n}^{B}\right)\right), \\
K_{1, n}^{A}:=f^{A}\left(A_{n}, B_{n}\right) \\
A_{n+1 / 2}:=\exp _{A_{n}}\left(\frac{h^{A}}{2} K_{1, n}^{A}\right), \\
K_{2, n}^{A}:=f^{A}\left(A_{n+1 / 2}, B_{n}\right) \\
A_{n+1}=\exp _{A_{n}}\left(h^{A} \mathrm{P}_{n+1 / 2}^{A_{n} \rightarrow A_{n}}\left(K_{2, n}^{A}\right)\right)
\end{array}\right.
$$

The last version of the HM-type learning systems is obtained by regarding the curved space $\mathrm{SO}(m)$ as a Lie group with Lie algebra $\mathfrak{s o}(m)$, which leads to the following numerical method:

$$
\text { Runge, Version 3: }\left\{\begin{array}{l}
T_{1, n}^{A}:=-\sigma\left(A_{n}^{\top} Z B_{n} W\right) \in \mathfrak{s o}(m)  \tag{23}\\
A_{n+1 / 2}:=A_{n} \operatorname{Exp}\left(\frac{h}{2} T_{1, n}^{A}\right), \\
T_{2, n}^{A}:=-\sigma\left(A_{n+1 / 2}^{\top} Z B_{n} W\right) \in \mathfrak{s o}(m), \\
A_{n+1}=A_{n} \operatorname{Exp}\left(\frac{h}{6} T_{1, n}^{A}+\frac{h}{3} T_{2, n}^{A}\right), \\
T_{1, n}^{B}:=\sigma\left(W A_{n}^{\top} Z B_{n}\right) \in \mathfrak{s o}(m), \\
B_{n+1 / 2}:=B_{n} \operatorname{Exp}\left(\frac{h}{2} T_{1, n}^{B}\right), \\
T_{2, n}^{B}:=\sigma\left(W A_{n}^{\top} Z B_{n+1 / 2}\right) \in \mathfrak{s o}(m), \\
B_{n+1}=B_{n} \operatorname{Exp}\left(\frac{h}{6} T_{1, n}^{B}+\frac{h}{3} T_{2, n}^{B}\right)
\end{array}\right.
$$

Even this version of the numerical learning scheme does not require using the computationally intensive parallel transport. The highlighted update equations were drawn from the survey [51].

### 2.5. Preliminary Pyramidal Numerical Tests on the HM Learning Methods

This subsection illustrates the results of a pyramidal comparison of the above learning algorithms in order to find out which algorithm is more convenient from a computational viewpoint, for comparable learning performances. The numerical experiments concerned the computation of the SVD of random matrices $Z$. The numerical experiments discussed in this subsection were performed on an Intel ${ }^{\circledR}$ i5-6400T 4-Core CPU, 2.2 GHz clock, 8 GB RAM machine and were coded in a MATLAB ${ }^{\circledR}$ R2017b environment (we used non-parallel programming in the experiments). In the following experiments, the learning stepsizes $h_{A}$ and $h_{B}$ were set to a common value denoted by $h$. Moreover, we adopted the following criterion to stop the iteration based on the objective function (1):

$$
\begin{equation*}
\phi_{W}\left(A_{n+1}, B_{n+1}\right)-\phi_{W}\left(A_{n}, B_{n}\right) \leqslant \tau \tag{24}
\end{equation*}
$$

where $\tau$ denotes a threshold. The size of the unknown-matrix $B$ was set to $p:=3$ in all experiments, and the sub-matrix $W_{1}$ explained in the Section 2.1 was set to $\operatorname{diag}(3,2,1)$. The size $m$ of the matrix $B$ took values in $\{9,25,225\}$ (these specific values would arise in the computation of the optical-flow of $720 \times 720$ pixels frames, as will be shown in Section 4.2). In all experiments, the initial matrices were chosen as $A_{0}:=I_{m}$ and $B_{0}:=I_{3}$. For comparison purposes, the SVD of a matrix $Z$ was also computed
by MATLAB's SVD engine, whose output triple is denoted by $(U, \cdot, V)$. As objective measures of the quality of the learned SVD factors, we used the following figures of demerit:

$$
\left\{\begin{array}{l}
\omega_{A}\left(A_{n}\right):=\left\|A_{n}^{\top} A_{n}-I_{m}\right\|_{\mathrm{F}}, \text { (Non-orthogonality) }  \tag{25}\\
\omega_{B}\left(B_{n}\right):=\left\|B_{n}^{\top} B_{n}-I_{3}\right\|_{\mathrm{F}}, \\
\varepsilon_{A}\left(A_{n}\right):=\left\|\left|A_{n}^{\top} U\right|-I_{m}\right\|_{\mathrm{F}}, \text { (Discrepancy) } \\
\varepsilon_{B}\left(B_{n}\right):=\left\|\left|B_{n}^{\top} V\right|-I_{3}\right\|_{\mathrm{F}},
\end{array}\right.
$$

where $|\cdot|$ denotes an entry-wise absolute value of a matrix. Both numerical orthogonality and discrepancy between the learned SVD and the MATLAB-computed reference SVD were expected to result in being as close as possible to zero.

The results of the first experiment, which was meant to compare the performances of the Heun method in its Versions 1 and 4, are summarized in Table 1. The random matrix $Z$ was generated entry-wise as a set of random numbers uniformly distributed in the interval [ 0,1 ]. This choice corresponded to a normalized pixel luminance, and the uniform distribution was motivated by the fact that a single pixel location may in principle take any luminance value. In the low-dimensional cases, $h:=0.05$, while in the higher dimensional case, $h:=0.005$. In all cases, $\tau:=10^{-5}$. The acquired runtimes showed that Version 4 was much lighter than Version 1, a phenomenon that became more apparent as the size of the matrix $A$ increased.

Table 1. Results of a comparison between the runtimes (in seconds) of the Heun method in its Versions 1 and 4.

|  | Size of $A$ | $:=\mathbf{9}$ | $:=\mathbf{2 5}$ |
| :---: | :---: | :---: | :---: |
| Method | $\boldsymbol{m}:=\mathbf{2 2 5}$ |  |  |
| Heun, Version 1 | 0.3836 | 0.3468 | 34.5559 |
| Heun, Version 4 | 0.2264 | 0.1148 | 9.1531 |

A graphical comparison of the values taken by the discrepancies defined in (25) and of the learning criterion (1) during learning is displayed in Figure 1. In this case, $m:=9$. The discrepancy and learning curves showed that, as concerns the learning performances, no meaningful differences between the two versions of the HM-type learning algorithms could be appreciated.


Figure 1. Results of a comparison of the values taken by the performance indexes during learning by the Heun method in its Versions 1 and 4.

The results of this experiment entailed that the Heun method, Version 4 was the preferable version in the Heun class.

The results of the second experiment, which was meant to compare the performances of the Runge method in its Versions 1 and 3, are summarized in Table 2. In the low-dimensional cases, $h:=0.05$, while in the higher dimensional case, $h:=0.005$. In all cases, $\tau:=10^{-5}$. The acquired runtimes showed that Version 3 was lighter than Version 1.

Table 2. Results of a comparison between the runtimes (in seconds) of the Runge method in its Versions 1 and 3.

|  | Size of $A$ | $\boldsymbol{m}:=\mathbf{9}$ | $\boldsymbol{m}:=\mathbf{2 5}$ |
| :---: | :---: | :---: | :---: |
| Method | $\boldsymbol{m}:=\mathbf{2 2 5}$ |  |  |
| Runge, Version 1 | 0.4536 | 0.8816 | 65.1238 |
| Runge, Version 3 | 0.3645 | 0.5274 | 21.7729 |

A graphical comparison of the values taken by the discrepancies and of the learning criterion during learning is displayed in the Figure 2. In this case, $m:=9$. The discrepancy and learning curves showed that, as concerns the learning performances, Version 1 converged only slightly more rapidly to the expected solutions.


Figure 2. Results of a comparison of the values taken by the performance indexes during learning by the Runge method in its Versions 1 and 3.

The results of this experiment entailed that the Runge method, Version 3 was the preferable version in the Runge class.

The results of the last experiment of this subsection, which was meant to compare the performances of the best Runge method and of the best Heun method to the performances of the Euler method, are summarized in Table 3. In the low-dimensional cases, $h:=0.05$, while in the higher dimensional case, $h:=0.005$. In all cases, $\tau:=10^{-5}$. The acquired runtimes revealed that the Euler method was the lightest in terms of computational complexity.

A graphical comparison of the values taken by the discrepancies and of the learning criterion during learning is displayed in Figure 3. In this case, $m:=225$. The discrepancy and learning curves showed that, as concerns the learning performances, the Euler method-based HM learning algorithm and the Heun method-based learning algorithm converged more rapidly to the expected solutions.

Table 3. Results of a comparison between the runtimes (in seconds) of the Heun method, Version 4, the Runge method, Version 3, and the Euler method.

|  | Size of $A$ | $\boldsymbol{m}:=\mathbf{9}$ | $\boldsymbol{m}:=\mathbf{2 5}$ |
| :---: | :---: | :---: | :---: |
| $\mathbf{m}:=\mathbf{2 2 5}$ |  |  |  |
| Method |  |  |  |
| Heun, Version 4 | 0.2033 | 0.3219 | 20.0415 |
| Runge, Version 3 | 0.2060 | 0.3601 | 26.3659 |
| Forward Euler | 0.0740 | 0.1096 | 8.5624 |



Figure 3. Results of a comparison of the values taken by the discrepancies and the learning criterion during learning by the Heun method, Version 4, the Runge method, Version 3, and the Euler method.

A graphical comparison of the values taken by the non-orthogonality figures defined in (25) during learning is displayed in Figure 4. In this case, $m:=225$. The orthogonality curves showed that the Euler method-based HM was unable to keep the same numerical precision of the Heun-based and of the Runge-based HM learning algorithms, the former being a first-order method and the latter two second-order methods. However, all these algorithms kept the non-orthogonality figures at very low values and stabilized after learning completion.


Figure 4. Results of a comparison of the values taken by the discrepancies and the learning criterion during learning by the Heun method, Version 4, the Runge method, Version 3, and the Euler method.

The results of these three experiments revealed that the Euler method to implement a Helmke-Moore learning paradigm guaranteed an excellent trade-off between computational complexity and numerical precision.

## 3. Reduced Euler Method

In some applications, such as optical-flow estimation recalled in the next Section 4, only the matrix $B$ is relevant. In view of the optical-flow application discussed in the next section, assume that $B$ is a $3 \times 3$ factor of the singular value decomposition of a data matrix $Z$. Notice that, in general, the matrix $A$ has a much bigger size than the matrix $B$; therefore, it would be certainly convenient to express the matrix $A$ as a function of the matrix $B$ and to express the learning system (2) only in terms of the variable matrix $B$. The following arguments may be extended easily to the general case that $B$ has different sizes.

### 3.1. Derivation of a Reduced Euler Method

The HM-type learning system (2) in the form (13) admits, as stable configurations, all pairs $(A, B) \in \mathrm{SO}(m) \times \mathrm{SO}(3)$ such that:

$$
\left\{\begin{array}{l}
\sigma\left(A^{\top} Z B W\right)=0  \tag{26}\\
\sigma\left(W A^{\top} Z B\right)=0
\end{array}\right.
$$

Therefore, one of these conditions may be used to express the matrix variable $A$ in terms of the matrix variable $B$. In particular, the first condition, in plain form, reads:

$$
\begin{equation*}
W^{\top}\left(A^{\top} Z B\right)^{\top}-A^{\top} Z B W=0 \tag{27}
\end{equation*}
$$

The above equation is equivalent to $\left(W^{\top} B^{\top} Z^{\top}\right) A-A^{\top}(Z B W)=0$. Defining $K:=$ $W^{\top} B^{\top} Z^{\top}$, Equation (27) takes the form:

$$
\begin{equation*}
K A-A^{\top} K^{\top}=0, \tag{28}
\end{equation*}
$$

which is to be solved in $A$. Let us observe that the matrix $K$ has size $9 \times 9$ and that its last six rows are zero, while the matrix $A$ has size $9 \times 9$ and that its last six columns do not influence the result. Let us partition the matrices $K$ and $A$ as follows:

$$
K=\left[\begin{array}{c}
K_{1}  \tag{29}\\
0
\end{array}\right], A=\left[\begin{array}{ll}
A_{1} & A_{2}
\end{array}\right]
$$

with $K_{1}$ is $3 \times 9, A_{1}$ is $9 \times 3$, and $A_{2}$ is $9 \times 6$. The matrix product $K A$ exhibits the structure:

$$
K A=\left[\begin{array}{c}
K_{1}  \tag{30}\\
0
\end{array}\right]\left[\begin{array}{ll}
A_{1} & A_{2}
\end{array}\right]=\left[\begin{array}{cc}
K_{1} A_{1} & K_{1} A_{2} \\
0 & 0
\end{array}\right] .
$$

According to the condition (28), the product $K A$ needs to result in a symmetric matrix; hence, the product $K_{1} A_{1}$ needs to be symmetric, while the product $K_{1} A_{2}$ needs to be zero. Since the matrix $A_{2}$ plays no role in the product $K A$, it is computationally convenient to turn down the orthogonality of the matrix $A$ and to set $A_{2}=0_{m \times n}$, while the columns of the sub-matrix $A_{1}$ still need to be mutually orthogonal and unit-norm. Since $K_{1} A_{1}$ is symmetric, let us set $K_{1} A_{1}=: S$, where $S$ is a $3 \times 3$ symmetric matrix to be determined. To determine the unknown matrix $S$, let us rewrite the last equation by the ansatz $K_{1}^{\top}=A_{1} S$, from which it follows that:

$$
\begin{equation*}
K_{1} K_{1} \top=S A_{1}^{\top} A_{1} S=S^{2} \Rightarrow S=\sqrt{K_{1} K_{1}^{\top}} \tag{31}
\end{equation*}
$$

therefore:

$$
\begin{equation*}
K_{1}^{\top}=A_{1} \sqrt{K_{1} K_{1}^{\top}} \Rightarrow A_{1}=K_{1}^{\top}\left(K_{1} K_{1}^{\top}\right)^{-1 / 2} \tag{32}
\end{equation*}
$$

The matrix $K_{1}$ is defined as a sub-matrix of $K$, which, in turn, is a function of the matrix $B$ (and depends on the weight matrix $W$ and on the data matrix $Z$ ). Hence, the function:

$$
\begin{equation*}
F(B):=\left[K_{1}^{\top}(B)\left(K_{1}(B) K_{1}^{\top}(B)\right)^{-1 / 2} \quad 0\right] \tag{33}
\end{equation*}
$$

may be used to reduce the two update rules in the Euler method (15) to the single update rule:

$$
\left\{\begin{array}{l}
\tilde{f}(B):=B \sigma\left(W F^{\top}(B) Z B W\right)  \tag{34}\\
B_{n+1}=\exp _{B_{n}}\left(h \tilde{f}\left(B_{n}\right)\right),
\end{array}\right.
$$

with $h>0$ and with the only initial condition $B_{0}$. Notice that the inverse matrix square root may be expressed in terms of the matrix exponential Exp and the matrix logarithm Log as $X^{-1 / 2}=$ $\operatorname{Exp}\left(-\frac{1}{2} \log (X)\right)$. The algorithm (34) will be referred to as the reduced Euler method.

### 3.2. Numerical Comparison of the Euler Method and Its Reduced Version

The results of a comparison of the performances of the forward Euler method (15) and of the reduced Euler method (34) to implement an HM learning paradigm are summarized in Table 4. In the low-dimensional cases, $h:=0.05$, while in the higher dimensional case, $h:=0.005$. In all cases, $\tau:=10^{-5}$. The acquired runtimes showed that the reduced Euler method was way lighter than the original Euler method, a phenomenon that became more apparent as the size $m$ of the $m \times 3$ matrix $Z$ increased.

A graphical comparison of the values taken by the figures of demerit defined in (25) and of the learning criterion (1) during learning is displayed in Figure 5. In this case, $m:=9$. The discrepancy and learning curves showed that, as concerns the learning performances, no meaningful differences between the two versions of the HM-type learning algorithms can be appreciated.

The results of this experiment confirmed that the reduced Euler method was both effective and efficient to implement the HM learning paradigm.


Figure 5. Results of a comparison of the figures of demerit and of the learning criterion for the forward Euler method and for the reduced Euler method.

Table 4. Results of a comparison between the runtimes (in seconds) for the forward Euler method and for the reduced Euler method.

| Size $\boldsymbol{m}$ | $\boldsymbol{m}:=\mathbf{9}$ | $\boldsymbol{m}:=\mathbf{2 5}$ | $\boldsymbol{m}:=\mathbf{2 2 5}$ |
| :---: | :---: | :---: | :---: |
| Forward Euler | 0.2003 | 0.1190 | 7.7378 |
| Reduced Euler | 0.0833 | 0.0441 | 0.4350 |

## 4. Experiments on Optical Flow Estimation by a Helmke-Moore Neural Learning Algorithm

In this section, we recall the notion of optical-flow estimation by a total least squares method from [42] and illustrate the behavior of the devised learning algorithm by means of numerical experiments performed on a recorded video sequence.

### 4.1. Optical Flow Estimation by Total Least Squares

Let us consider a gray-scale video sequence $\{\mathscr{I}(x, y, q)\}_{q}$, where $\mathscr{I}$ denotes the scalar image intensity, the pair $(x, y)$ denotes the coordinate pair of any pixel, and $q$ denotes the frame index.

During motion, any pixel moves from frame $q$ and position $(x, y)$ to frame $q+\Delta q$ at position $(x+\Delta x, y+\Delta y)$. The fact that the pixel intensity has moved over the image support may be formally expressed by the optical-flow conservation equation, namely:

$$
\begin{equation*}
\mathscr{I}(x-\Delta x, y-\Delta y, q)=\mathscr{I}(x, y, q+\Delta q) . \tag{35}
\end{equation*}
$$

On the basis of the above conservation law, it is possible to estimate the motion of any pixel within the sequence $\{\mathscr{I}(x, y, q)\}_{q}$. In fact, let us define $\Delta \mathscr{I}(x, y, q):=\mathscr{I}(x, y, q+\Delta q)-\mathscr{I}(x, y, q)$. An application of the Taylor series expansion gives:

$$
\begin{equation*}
\Delta \mathscr{I}(x, y, q)=\mathscr{I}(x-\Delta x, y-\Delta y, q)-\mathscr{I}(x, y, q)=-\mathscr{I}_{x}(x, y, q) \Delta x-\mathscr{I}_{y}(x, y, q) \Delta y+\text { residual }, \tag{36}
\end{equation*}
$$

where $\mathscr{I}_{x}$ and $\mathscr{I}_{y}$ denote the partial derivatives of the image function along the vertical and the horizontal direction, respectively, and the term "residual" denotes the sum of higher order terms in the Taylor expansion of the image function and the sum of random disturbances occurring in the recording of a video sequence, while the vertical and horizontal displacements $(\Delta x, \Delta y)$ were supposed small enough for the above truncated Taylor series to represent accurately the optical-flow change. The latter hypothesis was equivalent to assuming slow motion or, equivalently, sufficiently high-rate image sampling.

As mentioned, we made the solid-block motion assumption, namely the above equation was supposed to hold true, with the same values of displacements, for a set of pixels located within the rectangular patch described by $x \in\left[x_{1}, x_{N_{x}}\right]$ and $y \in\left[y_{1}, y_{N_{y}}\right]$, where integers $N_{x}$ and $N_{y}$ denote the block-size. On the basis of these considerations, it was possible to write the resolving system for any block between frames $q$ and $q+\Delta q$, that is:

$$
\begin{aligned}
\mathscr{I}\left(x_{1}, y_{1}, q+\Delta q\right)-\mathscr{I}\left(x_{1}, y_{1}, q\right) & =-\mathscr{I}_{x}\left(x_{1}, y_{1}, q\right) \Delta x-\mathscr{I}_{y}\left(x_{1}, y_{1}, q\right) \Delta y, \\
\mathscr{I}\left(x_{2}, y_{1}, q+\Delta t\right)-\mathscr{I}\left(x_{2}, y_{1}, q\right) & =-\mathscr{I}_{x}\left(x_{2}, y_{1}, q\right) \Delta x-\mathscr{I}_{y}\left(x_{2}, y_{1}, q\right) \Delta y, \\
\mathscr{I}\left(x_{3}, y_{1}, q+\Delta q\right)-\mathscr{I}\left(x_{3}, y_{1}, q\right) & =-\mathscr{I}_{x}\left(x_{3}, y_{1}, q\right) \Delta x-\mathscr{I}_{y}\left(x_{3}, y_{1}, q\right) \Delta y, \\
\vdots & \\
\mathscr{I}\left(x_{1}, y_{2}, t+\Delta q\right)-\mathscr{I}\left(x_{1}, y_{2}, q\right) & =-\mathscr{I}_{x}\left(x_{1}, y_{2}, q\right) \Delta x-\mathscr{I}_{y}\left(x_{1}, y_{2}, q\right) \Delta y, \\
\vdots & \\
\mathscr{I}\left(x_{N_{x}}, y_{N_{y}}, q+\Delta q\right)-\mathscr{I}\left(x_{N_{x}}, y_{N_{y}}, q\right) & =-\mathscr{I}_{x}\left(x_{N_{x}}, y_{N_{y}}, q\right) \Delta x-\mathscr{I}_{y}\left(x_{N_{x}}, y_{N_{y}}, q\right) \Delta y,
\end{aligned}
$$

where high-order terms were neglected.

By defining a displacement vector $v:=[\Delta x \Delta y]^{\top}$ and upon defining a matrix $L \in \mathbb{R}^{N_{x} N_{y} \times 2}$ and a vector $c \in \mathbb{R}^{N_{x} N_{y}}$, the above system casts into $L v=c$. This is an over-determined linear system of $N_{x} \cdot N_{y}$ equations in two unknowns, which may be solved by the help of a total least squares technique [52].

The TLS approach to solving an over-determined system of linear equations is based on the singular value decomposition (SVD) technique [53,54]. The resulting procedure is stated in Algorithm 1.

```
Algorithm 1 Total least squares based algorithm to estimate a displacement vector.
    Define the \(N_{x} N_{y} \times 3\) matrix \(Z=[L c]\);
    Compute the SVD \((U, D, V)\) of \(Z\), with \(V \in \mathrm{SO}(3)\);
    Define the block-partition \(V=\left[\begin{array}{cc}\cdots & V_{12} \\ \cdots & V_{22}\end{array}\right]\) with \(V_{12} \in \mathbb{R}^{2 \times 1}\);
    Estimate a displacement vector \(v\) as \(-V_{12} V_{22}^{-1}\).
```

It is interesting to underline that, depending on the size of each block, the data matrix $Z$ whose SVD is to be calculated may be large sized. Moreover, for later considerations, it is important to underline that the only factor of an SVD decomposition that plays an effective role in motion estimation is $V$.

### 4.2. Numerical Experiments on Optical-Flow Learning

In order to test the capability of the HM learning paradigm in optical-flow estimation, a video was realized and saved in MP4 format. The pre-processing of the images consisted of applying a filter after the subdivision of the frames in sub-windows: every gradient matrix was examined and then the windows, which presented the null gradient, were excluded from further processing in the successive steps as they corresponded to block of pixels that were not subjected to any movement. In the post-processing, after the velocities along the $x$ - and $y$-axes were calculated by using an HM learning algorithm, their norms were compared to a threshold, and all the windows that presented a velocity norm larger than the threshold were ignored in the optical-flow estimation as they were considered to correspond to noisy pixels. These filters were built to clean the image from noise created by the environment and by the camera, but they also caused a small loss of information, which could be neglected. In the experiments summarized in the present subsection, the following setting was adopted:

- The optical-flow was estimated between Frame 130 and Frame 135.
- The original colored still images drawn from the two considered frames were converted from RGB to gray scale using the luminance formula $0.2989 R+0.5870 G+0.1140 B$ as prescribed by the recommendation ITU-R BT.601-7 [55].
- The threshold to stop iteration was set to $\tau:=0.001$.
- The learning stepsizes were set as $h_{A}=h_{B}=h:=0.0005$.
- Given the frame size of $720 \times 720$, each frame was subdivided into windows of size $3 \times 3,5 \times 5$, or $15 \times 15$ pixels (corresponding to 57,600 matrices of size $9 \times 3$ to compute the SVD, 20,736 matrices of size $25 \times 3$ to compute the SVD, and to 2304 matrices of size $225 \times 3$ to compute the SVD, respectively).
- Computing platform: Intel ${ }^{\circledR}$ i7-8550U 4-Core CPU, 1.8 GHz clock, 16 GB RAM. Coding platform: MATLAB ${ }^{\circledR}$ R2018b.

The computational complexity related to the estimation of the optical-flow between two frames depended on the number of sub-windows and on their size; therefore, it was interesting to test the HM learning algorithms in different instances.

With reference to the HM learning paradigm, in the present application, it holds that $m:=N_{x} \cdot N_{y}$ (which depends on the size of a image's window) while $p:=3$ (fixed). Considering the block decomposition $H=\left[\begin{array}{c}H_{1} \\ \cdots\end{array}\right]$, with $H_{1}$ of size $3 \times 3$, the product $W H$ may be simplified as $W_{1} H_{1}$, while the product $H W$ may be computed as the composite matrix $\left[H W_{1} 0_{N x N y, N x N y-3}\right]$. Table 5 shows the results of a comparison of runtimes taken to estimate the optical-flow between two frames by the forward Euler method-based HM learning paradigm and the reduced Euler method-based algorithm.

Table 5. Comparison of runtimes (in seconds) to compute the optical-flow between two frames by the forward Euler method and the reduced Euler method.

| Window Size | Method | Forward Euler Method | Reduced Euler Method | Time Saving (\%) |
| :---: | :---: | :---: | :---: | :---: |
| $3 \times 3$ | 1833 | 907 | $50.52 \%$ |  |
| $5 \times 5$ | 1177 | 606 | $48.51 \%$ |  |
| $15 \times 15$ | 15,243 | 241 | $98.42 \%$ |  |

It is very interesting to observe how the computational complexity of the forward Euler method increased with the size of the sub-windows, while the complexity of the reduced Euler method showed an opposite trend. The time saving afforded by the reduced Euler method ranged from about $50 \%$ for low-dimensional windows to about $98 \%$ for high-dimensional windows.

Figure 6 illustrates the results of optical-flow estimation when the windows size was set to $3 \times 3$.
Figure 7 illustrates the results of optical-flow estimation when the windows size was set to $5 \times 5$.
Figure 8 illustrates the results of optical-flow estimation when the windows size was taken as $15 \times 15$.

The estimation obtained through the reduced Euler method seemed cleaner and free of artifacts, which may denote a stronger robustness against noise.


Figure 6. Results of optical-flow estimation when the windows size was set to $3 \times 3$.


Figure 7. Results of optical-flow estimation when the windows size was set to $5 \times 5$.


Figure 8. Results of optical-flow estimation when the windows size was set to $15 \times 15$.

## 5. Conclusions

The aim of this paper was to present two first-order (forward Euler and reduced Euler) and two second-order (Heun and Runge) numerical learning methods to learn the SVD of large rectangular matrices.

We recalled the Helmke-Moore learning paradigm to estimate the SVD of a given rectangular matrix, which was expressed by two coupled differential equations on the special orthogonal group. Taking the move from classical numerical calculus methods to solve ordinary differential equations, we suggested several numerical schemes to extend these classical methods to the manifold of special orthogonal matrices.

The resulting methods were tested numerically against each other in a pyramidal comparison in order to determine their merits and demerits in terms of computational complexity and effectiveness in computing the SVD. Observing that, for total least-squares-based optical-flow estimation, only one among the three SVD factors was necessary, a further reduced method was developed and tested against its non-reduced counterpart.

The conclusion of the experimental tests was that the estimation obtained through the reduced Euler method looked clean and free of artifacts, which may suggest robustness against noise and that the time saving of the reduced Euler method with respect to the forward Euler method increased noticeably when the size of the sub-windows increased.

Author Contributions: Conceptualization, S.F.; data curation, M.G.; formal analysis, S.F.; software, S.F., L.D.R., M.G., and A.S.; writing, original draft, L.D.R., M.G., and A.S.; writing, review and editing, S.F. All authors read and agreed to the published version of the manuscript.

Funding: This research received no external funding.
Conflicts of Interest: The authors declare no conflict of interest.

## References

1. Helmke, U.; Moore, J.B. Singular value decomposition via gradient and self-equivalent flows. Linear Algebra Appl. 1992, 169, 223-248. [CrossRef]
2. Hori, G. A general framework for SVD flows and joint SVD flows. In Proceedings of the 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, Vancouver, BC, Canada, 26-31 May 2013; Volume II, pp. 693-696.
3. Smith, S.T. Dynamic system that perform the SVD. Syst. Control Lett. 1991, 15, 319-327. [CrossRef]
4. Alter, O.; Brown, P.O.; Botstein, D. Singular value decomposition for genome-wide expression data processing and modeling. Proc. Natl. Acad. Sci. USA 2000, 97, 10101-10106. [CrossRef] [PubMed]
5. Ciesielski, A.; Forbriger, T. Singular value decomposition of tidal harmonics on a rigid Earth. Geophys. Res. Abstr. 2019, 21, EGU2019-16895.
6. Claudino, D.; Mayhall, N.J. Automatic partition of orbital spaces based on singular value decomposition in the context of embedding theories. J. Chem. Theory Comput. 2019, 15, 1053-1064. [CrossRef]
7. Ernawan, F.; Ariatmanto, D. Image watermarking based on integer wavelet transform-singular value decomposition with variance pixels. Int. J. Electr. Comput. Eng. 2019, 9, 2185-2195. [CrossRef]
8. Garcia-Pena, M.; Arciniegas-Alarcon, S.; Barbin, D. Climate data imputation using the singular value decomposition: An empirical comparison. Revista Brasileira de Meteorologia 2014, 29, 527-536.
9. Hao, Z.; Cuia, Z.; Yue, S.; Wang, H. Amplitude demodulation for electrical capacitance tomography based on singular value decomposition. Rev. Sci. Instrum. 2018, 89, 074705. [CrossRef]
10. Jha, A.K.; Barrett, H.H.; Frey, E.C.; Clarkson, E.; Caucci, L.; Kupinski, M.A. Singular value decomposition for photon-processing nuclear imaging systems and applications for reconstruction and computing null functions. Phys. Med. Biol. 2015, 60, 7359-7385. [CrossRef]
11. Liu, Y.; Sun, Y.; Li, B. A two-stage household electricity demand estimation approach based on edge deep sparse coding. Information 2019, 10, 224. [CrossRef]
12. Montesinos-López, O.A.; Montesinos-Lxoxpez, A.; Crossa, J.; Kismiantini; Ramírez-Alcaraz, J.M.; Singh, R.; Mondal, S.; Juliana, P. A singular value decomposition Bayesian multiple-trait and multiple-environment genomic model. Heredity 2019, 122, 381-401. [CrossRef] [PubMed]
13. Vezyris, C.; Papoutsis-Kiachagias, E.; Giannakoglou, K. On the incremental singular value decomposition method to support unsteady adjoint-Based optimization. Numer. Methods Fluids 2019, 91, 315-331. [CrossRef]
14. Moore, B.C. Principal component analysis in linear systems: Controllability, observability and model reduction. IEEE Trans. Autom. Control 1981, AC-26, 17-31. [CrossRef]
15. Lu, W.-S.; Wang, H.-P.; Antoniou, A. Design of two-dimensional FIR digital filters by using the singular value decomposition. IEEE Trans. Circuits Syst. 1990, CAS-37, 35-46. [CrossRef]
16. Salmeron, M.; Ortega, J.; Puntonet, C.G.; Prieto, A. Improved RAN sequential prediction using orthogonal techniques. Neurocomputing 2001, 41, 153-172. [CrossRef]
17. Costa, S.; Fiori, S. Image compression using principal component neural networks. Image Vision Comput. J. 2001, 19, 649-668. [CrossRef]
18. Nestares, O.; Navarro, R. Probabilistic estimation of optical-flow in multiple band-pass directional channels. Image Vision Comput. J. 2001, 19, 339-351. [CrossRef]
19. Cichocki, A.; Unbehauen, R. Neural networks for computing eigenvalues and eigenvectors. Biol. Cybern. 1992, 68, 155-164. [CrossRef]
20. Sanger, T.D. Two iterative algorithms for computing the singular value decomposition from input/output samples. In Advances in Neural Processing Systems; Cowan, J.D., Tesauro, G., Alspector, J., Eds.; Morgan-Kauffman Publishers: Burlington, MA, USA, 1994; Volume 6, pp. 1441-1451.
21. Weingessel, A. An Analysis of Learning Algorithms in PCA and SVD Neural Networks. Ph.D. Thesis, Technical University of Wien, Wien, Austria, 1999.
22. Afrashteh, N.; Inayat, S.; Mohsenvand, M.; Mohajerani, M.H. Optical-flow analysis toolbox for characterization of spatiotemporal dynamics in mesoscale optical imaging of brain activity. Neuroimage 2017, 153, 58-74., [CrossRef]
23. Ayzel, G.; Heistermann, M.; Winterrath, T. Optical flow models as an open benchmark for radar-based precipitation nowcasting (rainymotion v0.1). Geosci. Model Dev. 2019, 12, 1387-1402. [CrossRef]
24. Caramenti, M.; Lafortuna, C.L.; Mugellini, E.; Khaled, O.A.; Bresciani, J.-P.; Dubois, A. Matching optical-flow to motor speed in virtual reality while running on a treadmill. PLoS ONE 2018, 13, e0195781. [CrossRef] [PubMed]
25. Rahmaniar, W.; Wang, W.-J.; Chen, H.-C. Real-time detection and recognition of multiple moving objects for aerial surveillance. Electronics 2019, 8, 1373. [CrossRef]
26. Rosa, B.; Bordoux, V.; Nageotte, F. Combining differential kinematics and optical-flow for automatic labeling of continuum robots in minimally invasive surgery. Front. Robot. AI 2019, 6, 86. [CrossRef]
27. Ruiz-Vargas, A.; Morris, S.A.; Hartley, R.H.; Arkwright, J.W. Optical flow sensor for continuous invasive measurement of blood flow velocity. J. Biophotonics 2019, 12, 201900139. [CrossRef] [PubMed]
28. Vargas, J.; Quiroga, J.A.; Sorzano, C.O.S.; Estrada, J.C.; Carazo, J.M. Two-step interferometry by a regularized optical-flow algorithm. Opt. Lett. 2011, 36, 3485-3487. [CrossRef]
29. Serres, J.R.; Evans, T.J.; Åkesson, S.; Duriez, O.; Shamoun-Baranes, J.; Ruffier, F.; Hedenström, A. Optic flow cues help explain altitude control over sea in freely flying gulls. J. R. Soc. Interface 2019, 16, 20190486. [CrossRef]
30. Lazcano, V. An empirical study of exhaustive matching for improving motion field estimation. Information 2018, 9, 320. [CrossRef]
31. Liu, L.-K.; Feig, E. A block-based gradient descent search algorithm for block motion estimation in video coding. IEEE Trans. Circuits Syst. Video Technol. 1996, 6, 419-422.
32. Hsieh, Y.; Jeng, Y. Development of home intelligent fall detection IoT system based on feedback optical-flow convolutional neural network. IEEE Access 2018, 6, 6048-6057. [CrossRef]
33. Núñez-Marcos, A.; Azkune, G.; Arganda-Carreras, I. Vision-based fall Detection with convolutional neural networks. Wirel. Commun. Mob. Comput. 2017, 2017, 9474806. [CrossRef]
34. Toda, T.; Masuyama, G.; Umeda, K. Detecting moving objects using optical-flow with a moving stereo camera. In Proceedings of the 13th International Conference on Mobile and Ubiquitous Systems: Computing Networking and Services (MOBIQUITOUS 2016, Hiroshima, Japan, 28 November-1 December 2016; pp. 35-40.
35. Hou, Y.; Wang, J. Research on intelligent flight software robot based on internet of things. Glob. J. Technol. Optim. 2017, 8, 221. [CrossRef]
36. Shi, X.; Wang, M.; Wang, G.; Huang, B.; Cai, H.; Xie, J.; Qian, C. TagAttention: Mobile object tracing without object appearance information by vision-RFID fusion. In Proceedings of the 27th IEEE International Conference on Network Protocols, Chicago, IL, USA, 7-10 October 2019; pp. 1-11.
37. Zulkifley, M.A.; Samanu, N.S.; Zulkepeli, N.A.A.N.; Kadim, Z.; Woon, H.H. Kalman filter-based aggressive behaviour detection for indoor environment. In Lecture Notes in Electrical Engineering; Springer: Berlin/Heidelberg, Germany, 2016; Volume 376, pp. 829-837.
38. Hussain, T.; Muhammad, K.; Khan, S.; Ullah, A.; Lee, M.Y.; Baik, S.W. Intelligent baby behavior monitoring using embedded vision in IoT for smart healthcare centers. J. Artif. Intell. Syst. 2019, 1, 110-124. [CrossRef]
39. IBM Research Editorial Staff. Building Cognitive IoT Solutions Using Data Assimilation. Available online: https:/ /www.ibm.com/blogs/research/2016/12/building-cognitive-iot-solutions-using-data-assimilation/ (accessed on 9 December 2016).
40. Saquib, N. Placelet: Mobile Foot Traffic Analytics System Using Custom Optical Flow. Available online: https://medium.com/mit-media-lab/placelet-mobile-foot-traffic-analytics-system-using-custom-optical-flow-19bbebfc7cf8 (accessed on 3 March 2016).
41. Patrick, M. Why Machine Vision Matters for the Internet of Things. Available online: http:/ /iot-design-zone.com/article/knowhow/2525/why-machine-vision-matters-for-the-internet-of-things (accessed on 12 December 2019).
42. Deshpande, S.G.; Hwang, J.-N. Fast motion estimation based on total least squares for video encoding. Proc. Int. Symp. Circuits Syst. 1998, 4, 114-117.
43. Liu, K.; Yang, H.; Ma, B.; Du, Q. A joint optical-flow and principal component analysis approach for motion detection. In Proceedings of the 2010 IEEE International Conference on Acoustics, Speech and Signal Processing, Dallas, TX, USA, 14-19 March 2010; pp. 1178-1181.
44. Fiori, S. Unsupervised neural learning on Lie groups. Int. J. Neural Syst. 2002, 12, 219-246. [CrossRef]
45. Fiori, S. Singular value decomposition learning on double Stiefel manifold. Int. J. Neural Syst. 2003, 13, 155-170. [CrossRef]
46. Celledoni, E.; Fiori, S. Neural learning by geometric integration of reduced 'rigid-body' equations. J. Comput. Appl. Math. 2004, 172, 247-269. [CrossRef]
47. Fiori, S. A theory for learning by weight flow on Stiefel-Grassman manifold. Neural Comput. 2001, 13, 1625-1647. [CrossRef]
48. Fiori, S. A theory for learning based on rigid bodies dynamics. IEEE Trans. Neural Netw. 2002, 13, 521-531. [CrossRef]
49. Fiori, S. Fixed-point neural independent component analysis algorithms on the orthogonal group. Future Gener. Comput. Syst. 2006, 22, 430-440. [CrossRef]
50. Lambert, J.D. Numerical Methods for Ordinary Differential Systems: The Initial Value Problem, 1st ed.; Wiley: Hoboken, NJ, USA, 1991; ISBN 978-0471929901.
51. Celledoni, E.; Marthinsen, H.; Owren, B. An introduction to Lie group integrators-Basics, new developments and applications. J. Comput. Phys. Part B 2014, 257, 1040-1061. [CrossRef]
52. Golub, G.H.; van Loan, C.F. Matrix Computations, 3rd ed.; The John Hopkins University Press: Baltimore, MD, USA, 1996.
53. Kong, S.; Sun, L.; Han, C.; Guo, J. An image compression scheme in wireless multimedia sensor networks based on NMF. Information 2017, 8, 26. [CrossRef]
54. Wang, D. Adjustable robust singular value decomposition: Design, analysis and application to finance. Data 2017, 2, 29. [CrossRef]
55. RECOMMENDATION ITU-R BT.601-7. Studio Encoding Parameters of Digital Television for Standard 4:3 and Wide-Screen 16:9 Aspect Ratios; Version 2017, Available online: https:/ /www.itu.int/dms_pubrec/itu-r/rec/bt/R-REC-BT.601-7-201103-I!!PDF-E.pdf (accessed on 10 February 2020).
© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).
