Simplified State Update Calculation for Fast and Accurate Digital Emulation of CNN Dynamics

F. Pozas-Flores, R. Carmona-Galán, Á. Rodríguez-Vázquez
Institute of Microelectronics of Seville (IMSE-CNM-CSIC)
Consejo Superior de Investigaciones Científicas-Universidad de Sevilla
Email: pozas@imse-cnm.csic.es

***Abstract*—Compared to other one-step integration methods, the 4th-order Runge-Kutta is much more accurate while still consisting in a rather reduced algorithmic structure. However, in terms of the computing power, it is more expensive than others. While the Forward Euler's method updates the state variable with a single evaluation of the derivative, 4th-order Runge-Kutta's method requires four. This is the reason why, when simulation speed is a central matter, e. g. in the digital emulation of CNN dynamics, the speed-accuracy trade-off is resolved in favour of the simpler, though less accurate, methods. A workaround for the computationally intensive calculation of the state variable update can be found for certain CNN models. If a FSR CNN model is employed, where the state variable is not allowed to go beyond the limits of the linear region of the cell output characteristic, the output can be identified with the state. In these conditions, and having linear templates, the update of the state variable can be computed, for a 4th-order Runge-Kutta's method, with a single function evaluation. It means that a digital emulation of the CNN dynamics following this method is as light-weighted as a Forward Euler's integrator, but much more accurate.**

I. Introduction

After the invention of the CNN, an agile but reliable integration algorithm for the simulation of CNN dynamics was needed. Several numeric integration methods, commonly employed in engineering, were surveyed and compared [1]. From these studies, the major conclusion was that simulation speed and the accuracy of the results had to be traded off according to the application needs. In this regard, CNN simulators developed along the years [2–7] have been providing various different integration cores. This, for the user to select the most appropriate integrator for his/her speed/accuracy requirements. In all the cases, the 4th-order Runge-Kutta (RK4) method [8] represents a good compromise between computational effort and accuracy. However, when very demanding timing requirements are to be met, what is the case for real-time emulation of CNN dynamics, the use of RK4 is impractical. Either the available computing power is not enough to meet the requirements, or the amount of hardware to be incorporated to each processing element results in an unusefully small network. A much simpler Forward Euler's method (FE), instead of RK4, is usually employed either if a dedicated architecture is devised [9–11], or a general purpose architecture with parallel data processing capabilities is programmed to implement a CNN operating at high speed [12, 13]. The reason behind this widespread use of FE method can be easily seen from the inspection computational requirements. The objective of an integration method is the dicrete-time emulation of this equation:

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x}, t) \tag{1}$$

where the time-derivative of vector $\mathbf{x}$, which represents the state of the system, is given by a function of the state itself, and either explicitly or implicitly, of time. Numerical integration of such equation consists in the approximation of the integral:

$$\mathbf{x}(t + \Delta t) = \mathbf{x}(t) + \int_t^{t+\Delta t} \mathbf{f}(\mathbf{x}, \zeta)\, d\zeta \tag{2}$$

Following the FE algorithm, it is approximated by:

$$\int_t^{t+\Delta t} \mathbf{f}(\mathbf{x}, \zeta)\, d\zeta \approx \mathbf{f}(\mathbf{x}, t)\Delta t \tag{3}$$

what means that function $\mathbf{f}(\mathbf{x}, t)$ is evaluated only once, precisely at the beginning of the interval. On the other side, RK4 estimates the integral in Eq. (2) by:

$$\int_t^{t+\Delta t} \mathbf{f}(\mathbf{x}, \zeta)\, d\zeta \approx \frac{\Delta t}{6}\left(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4\right) \tag{4}$$

where:

$$\begin{aligned} \mathbf{k}_1 &= \mathbf{f}(\mathbf{x}, t) \\ \mathbf{k}_2 &= \mathbf{f}(\mathbf{x} + \mathbf{k}_1\Delta t/2, t + \Delta t/2) \\ \mathbf{k}_3 &= \mathbf{f}(\mathbf{x} + \mathbf{k}_2\Delta t/2, t + \Delta t/2) \\ \mathbf{k}_4 &= \mathbf{f}(\mathbf{x} + \mathbf{k}_3\Delta t, t + \Delta t) \end{aligned} \tag{5}$$

From these equations, it results obvious that, in the more general case, updating the state variable in RK4 requires four evaluations of $\mathbf{f}(\mathbf{x}, t)$. However, there are special cases in which these function evaluations can be reduced to one single calculation, like in FE. This is the case for a CNN that follows the full signal range (FSR) model [14].

II. FSR CNN Model

The evolution law of a standard CNN [15] with linear templates can be expressed in matrix form as:

$$\tau\frac{d\mathbf{x}}{dt} = -\mathbf{x} + \mathbf{A}\mathbf{y} + \mathbf{B}\mathbf{u} + \mathbf{z} \tag{6}$$

where $\mathbf{u}$, $\mathbf{x}$ and $\mathbf{y}$ are the input, state and output vectors, respectively, whose length, for a $M \times N$ grid, is just MN. Also in Eq. (6), matrices $\mathbf{A}$ and $\mathbf{B}$ have been obtained

978-1-4244-6678-8/10/$26.00 ©2010 IEEE

from the feedback and control templates, respectively, following the procedure established in [16], and have dimensions $MN \times MN$. Besides, the bias term $\mathbf{z}$ is also a MN-component long vector. The output vector is computed through the cell's activation function, that is usually a sigmoid-type nonlinear function:

$$\mathbf{y} = \frac{1}{2}\left(|\mathbf{x}+\mathbf{1}| - |\mathbf{x}-\mathbf{1}|\right) \tag{7}$$

where $\mathbf{1}$ is a $MN \times 1$ vector with all elements equal to unity.

Besides, the FSR model introduces some modifications to this standard CNN . Everything starts with the restriction of the possible values of the state variable to a limited interval by introducing a nonlinear implementation of the losses term:

$$\tau \frac{d\mathbf{x}}{dt} = -\mathbf{g}(\mathbf{x}) + \mathbf{A}\mathbf{y} + \mathbf{B}\mathbf{u} + \mathbf{z} \tag{8}$$

where the losses can be written in this compact form:

$$\mathbf{g}(\mathbf{x}) = \mathbf{x} + (m-1)\left(\mathbf{x} + \frac{1}{2}|\mathbf{x}-\mathbf{1}| - \frac{1}{2}|\mathbf{x}+\mathbf{1}|\right) \tag{9}$$

and ideally $m \to \infty$. If Eq. (9) is substituted in (8), it is easy to see that as soon as any component of $\mathbf{x}$ try to cross the boundaries of the linear region, i. e. the interval $[-1, 1]$, it is pushed back against the interval limit. In these conditions, the output and state variables can be identified, and the evolution law of the CNN reduces to:

$$\tau \frac{d\mathbf{x}}{dt} = (\mathbf{A}-\mathbf{I})\,\mathbf{x} + \mathbf{B}\mathbf{u} + \mathbf{z} \tag{10}$$

as long as each component of the state vector is forced to belong to $[-1, 1]$. Notice that $\mathbf{I}$ is here a $MN \times MN$ elements identity matrix.

Although discrepancies between the steady-states of networks following the standard and the FSR models have been found in some cases [17], having a bound in the state variable range represents a useful property when devising a real circuit. Therefore, the FSR model has been widely adopted in VLSI implementation of CNNs, both in analog and mixed-signal circuits [18] and in digital emulation [19].

III. 4th-order Runge-Kutta simplification

Let us now integrate Eq. (10) by using the RK4 method described by Eqs. (4) and (5). We start by computing the first term:

$$\mathbf{k}_1 = \mathbf{f}(\mathbf{x}, t) = \frac{1}{\tau}\left[(\mathbf{A}-\mathbf{I})\,\mathbf{x} + \mathbf{B}\mathbf{u} + \mathbf{z}\right] \tag{11}$$

Notice that the FE method will stop here, after the first evaluation of $\mathbf{f}(\mathbf{x})$, the time derivative of the state, resulting in the update equation:

$$\int_t^{t+\Delta t} \mathbf{f}(\mathbf{x}, \zeta)\, d\zeta \approx \mathbf{k}_1 \Delta t \tag{12}$$

Now, back to RK4, we have to compute the second, third and fourth terms in Eq. (5), and for that we are going to introduce our major restriction to the type of dynamics that we will be able to simulate correctly with this simplified method. The restriction is that neither the feedback and control templates, $\mathbf{A}$ and $\mathbf{B}$, nor the input and bias vectors, $\mathbf{u}$ and $\mathbf{z}$, can vary during the evolution of the network. Fortunately this is the case for the most of the image processing templates. If not, the complete processing can at least be approximated by decomposing the network evolution into discrete steps in which these magnitudes are considered constant. In these conditions we have:

$$\begin{aligned} \mathbf{k}_2 &= \mathbf{k}_1 + (\mathbf{A}-\mathbf{I})\,\mathbf{k}_1(\Delta t/2\tau) \\ \mathbf{k}_3 &= \mathbf{k}_1 + (\mathbf{A}-\mathbf{I})\,\mathbf{k}_2(\Delta t/2\tau) \\ \mathbf{k}_4 &= \mathbf{k}_1 + (\mathbf{A}-\mathbf{I})\,\mathbf{k}_3(\Delta t/\tau) \end{aligned} \tag{13}$$

and operating, we find that Eq. (4) turns into:

$$\begin{aligned} \int_t^{t+\Delta t} \mathbf{f}(\mathbf{x}, \zeta)\, d\zeta \approx \Bigg[\mathbf{I} &+ (\mathbf{A}-\mathbf{I})\left(\frac{\Delta t}{2\tau}\right) + \\ &+ \frac{2}{3}(\mathbf{A}-\mathbf{I})^2\left(\frac{\Delta t}{2\tau}\right)^2 + \\ &+ \frac{1}{3}(\mathbf{A}-\mathbf{I})^3\left(\frac{\Delta t}{2\tau}\right)^3 \Bigg] \mathbf{k}_1 \Delta t \end{aligned} \tag{14}$$

what means that, despite being RK4, the update of the state variable can be computed with only one single evaluation of the derivative. The update equation being now:

$$\int_t^{t+\Delta t} \mathbf{f}(\mathbf{x}, \zeta)\, d\zeta \approx \mathbf{M}\mathbf{k}_1 \Delta t \tag{15}$$

The only difference between equations (12) and (15) is the multiplication of vector $\mathbf{k}_1$ by a matrix $\mathbf{M}$ that can be calculated in advance outside of the integration loop. Even more, this multiplication can be also realized before the integration loop, modifying matrices $\mathbf{A}$ and $\mathbf{B}$, and vector $\mathbf{z}$, as long as they remain constant during the evolution of the network, what was already required for the simplification of the RK4 algorithm. Then, during integration, we can use the modified versions of the matrices to evaluate the derivative. Let us begin by defining:

$$\begin{aligned} \mathbf{A}' &= \mathbf{I} + \mathbf{M}\,(\mathbf{A}-\mathbf{I}) \\ \mathbf{B}' &= \mathbf{M}\mathbf{B} \\ \mathbf{z}' &= \mathbf{M}\mathbf{z} \end{aligned} \tag{16}$$

thus the new update of the state variable is calculated with the same computational effort as the FE's update, as can be seen by calculating:

$$\mathbf{M}\mathbf{k}_1 = \frac{1}{\tau}\left[(\mathbf{A}'-\mathbf{I})\,\mathbf{x} + \mathbf{B}'\mathbf{u} + \mathbf{z}'\right] \tag{17}$$

and comparing it to Eq. (11).

(*) Appendix for further speed improvement

For all methods, and given that $\mathbf{B}$, $\mathbf{u}$ and $\mathbf{z}$ remain constant during the evolution of the network, the vector:

$$\mathbf{C} = \mathbf{B}\mathbf{u} + \mathbf{z} \tag{18}$$

can be evaluated right before the beginning of the integration loop. This eliminates the need for one matrix multiplication plus one sum of vectors for each integration step, for FE and simplified RK4 methods. In the case of the regular RK4, it is four times this.

IV. Tests and Measurements

The tests reported in this section are intended to validate the above described simplification of the RK4 algorithm. The important results are, namely: achieving the same accuracy of the regular RK4 algorithm, achieving the same speed as a FE simulation. We must insist here that the main restriction made on the CNN dynamics to be simulated is the linear time-invariance of the templates (A, B and z, from which matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{z}$ are derived) and the CNN input ($\mathbf{u}$). In these conditions, there is no explicit dependence on time in the time-derivative of the state ($\mathbf{x}$). As the network follows the FSR CNN model, i. e. $\mathbf{x}$ and $\mathbf{y}$ can be identified, the simplification depicted by Eq. (15) can be done. This is a strong restriction for the study of dynamic systems in general, described by sets of ODEs or PDEs, and whose excitations are usually a function of time. But in the case of CNNs, many interesting image processing tasks and pattern formation dynamics are realized with zero or constant inputs and linear time-invariant templates. Therefore this approach is very useful.

All the examples presented in this section have been simulated with MATLAB, version 7.0.8.347, running in a x86-64 Linux computer, using a dual-core AMD Opteron(tm) processor, @2211MHz, 1024KB cache and 2GB RAM shared by other 3 processors like this one. The scripts that we have employed, together with the input images for each of them, can be found in `http://www.imse-cnm.csic.es/vmote/cnnsim`. The `00README` file explains which files are required to reproduce each of the examples reported here. Although exact CPU times will depend on the architecture and characteristics of the processor being employed. In all cases, a RK4 method using a very small integration step is employed as the reference against which the rest of the simulated trajectories are going to be compared. This methodology has been widely accepted since [1]. Then a FE, a regular RK4 and a simplified RK4 simulations are performed. The CPU time dedicated to each one of the integration loops is evaluated with the help of MATLAB's commands `tic` and `toc`. The results obtained are summarized in Tables I and II. As can be seen, the simplified RK4 method reports computing speeds similar to those achieved by the simpler FE while maintaining the full RK4 accuracy.

TABLE I
CPU time for each method (in seconds)

Templates	Image size	FE	RK4	RK4 simp.
Linear diffusion	12×12	7.9650e-02	4.1544e-01	9.7870e-02
	24×25	2.3371e+00	9.2761e+00	2.5232e+00
	48×51	3.4650e+01	1.3855e+02	3.9469e+01
Trigger waves	12×12	6.5395e-02	2.5480e-01	6.7580e-02
	24×24	2.4600e+00	9.6021e+00	3.0578e+00
	48×48	6.5357e+01	2.5722e+02	6.9848e+01
Pattern generation	12×12	1.6640e-01	6.3270e-01	1.3170e-01
	24×24	1.0046e+01	3.8027e+01	1.2162e+01
	48×48	1.4302e+02	5.8275e+02	1.2999e+02
Edge detection	12×12	1.5380e-01	6.9310e-01	1.5470e-01
	24×25	2.4273e+00	9.6703e+00	2.3809e+00
	48×51	6.5181e+01	2.6064e+02	6.1581e+01

TABLE II
Maximum RMSE from reference for each method (in volts)

Templates	Image size	FE	RK4	RK4 simp.
Linear diffusion	12×12	2.6010e-03	5.3662e-08	5.3622e-08
	24×25	2.4876e-03	4.8037e-08	4.8037e-08
	48×51	1.4265e-03	3.3234e-08	3.3234e-08
Trigger waves	12×12	1.5824e-01	4.1361e-02	4.1361e-02
	24×24	2.1156e-01	5.1412e-02	5.1412e-02
	48×48	2.7893e-01	6.2769e-02	6.2769e-02
Pattern generation	12×12	1.6667e-01	1.5938e-01	1.5938e-01
	24×24	8.7878e-02	2.5662e-02	2.5662e-02
	48×48	1.4484e-01	8.9391e-02	8.9391e-02
Edge detection	12×12	5.1647e-03	6.8922e-10	6.8922e-10
	24×25	5.0284e-03	6.7516e-10	6.7516e-10
	48×51	5.0304e-03	7.4916e-10	7.4916e-10

A. Linear diffusion

Linear diffusion can be achieved within the CNN framework by means of the following templates:

$$A = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -3 & 1 \\ 0 & 1 & 0 \end{pmatrix} \quad \begin{matrix} B = 0 \\ z = 0 \end{matrix} \tag{19}$$

The input, being B equal to zero, is irrelevant. From the image processing point of view, the input image is the initial state of the network, $\mathbf{x}(0)$. The time constant of the network is $\tau = 1\mu$s, and the integration step, relatively small, is 0.02τ. We have simulated the linear diffusion of images of different sizes. In order to evaluate the accuracy of the simulated trajectories, the states of all the cells in the network obtained for each integration method are compared to the reference. A measure of the error (RMSE) is obtained for each integration step. Fig. 1 shows some snapshots of the simulations for the diffusion of a 48×51-pixels image of Lena with the different

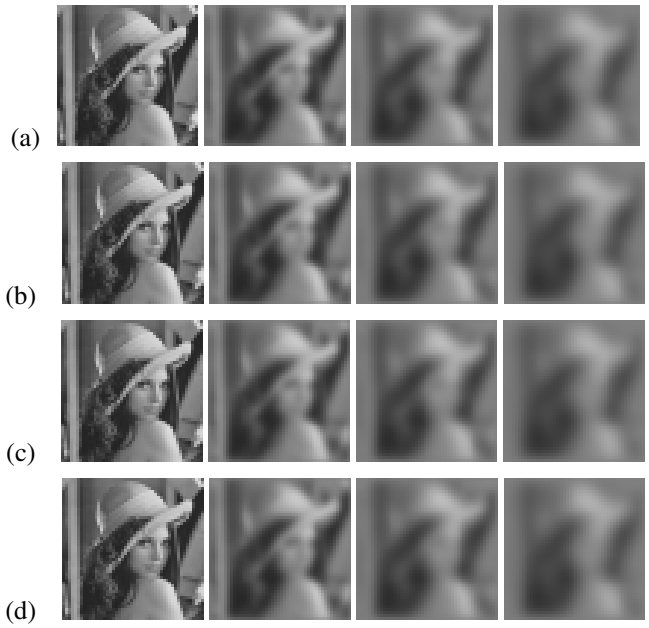

Fig. 1. Evolution of the state for the (a) reference, (b) FE, (c) RK4 and (d) simplified RK4 simulations.

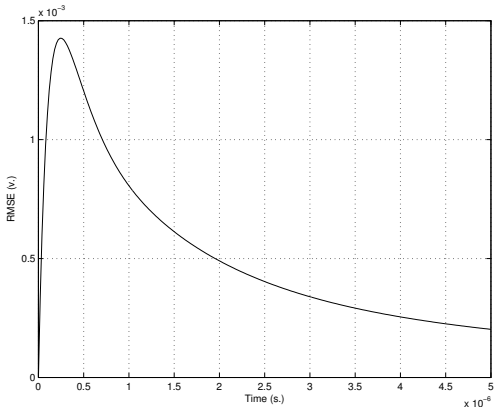

Fig. 2. FE vs. reference (linear diffusion)

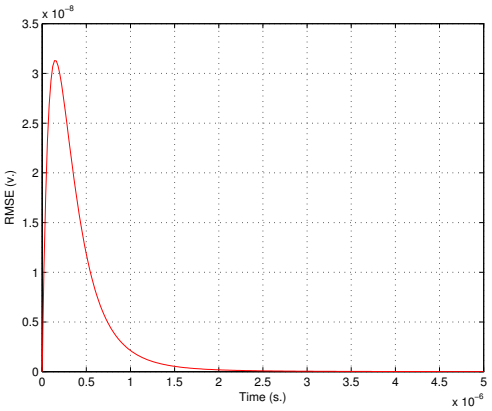

Fig. 3. RK4 vs. reference (linear diffusion)

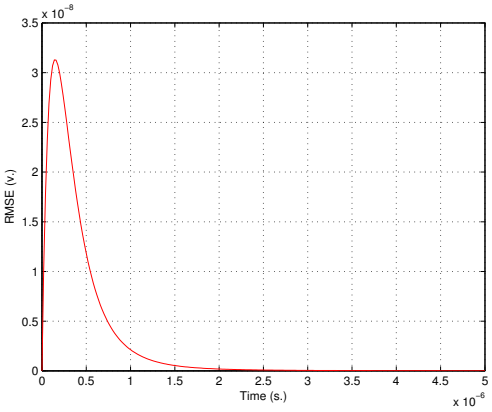

Fig. 4. Simplified RK4 vs. reference (linear diffusion)

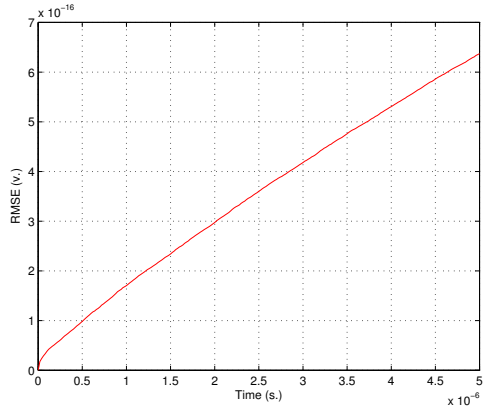

Fig. 5. Simplified vs. regular RK4 (linear diffusion)

integration methods. Differences are not apparent with these pictures.

Fig. 2 displays the RMSE observed when comparing the trajectories predicted by the FE method with the reference. For a relatively small integration step, 0.02τ, the error peaks at 2.6mV, for a signal range of $2V_{p\text{-}p}$. For the RK4 methods, both the regular RK4 and the simplified one, the error is much smaller (Fig. 3 and Fig. 4). There is no appreciable difference between the results obtained by RK4 and the simplified version of the RK4 (Fig. 5. Therefore it is confirmed that the simplification employed renders, at least in this case, RK4 accuracy. Concerning the simulation speed, reported in Table I, the simplified method renders FE speed, and therefore it takes only 25% of the time required by the regular RK4.

B. Trigger-waves

A trigger-wave is a binary wave that expands, from an initial triggering seed, in a direction that is normal to the interface of the active and inactive regions (Fig. 6). This trigger-waves can be employed to perform morphological operations and to implement some image metrics. The templates employed for this operation are:

$$A = \begin{pmatrix} 0.25 & 0.25 & 0.25 \\ 0.25 & 3.00 & 0.25 \\ 0.25 & 0.25 & 0.25 \end{pmatrix} \quad \begin{matrix} B = 0 \\ z = 3.75 \end{matrix} \tag{20}$$

The network time-constant and the integration time step are the same as in the previous example. Again, the distribution of CPU times obtained is of the same nature, FE and simplified RK4 take nearly the same amount of time to finish the integration loop, and they do it in 25-30% of the time employed by the regular RK4. Accuracy of the RK4 methods is higher, again, as can be seen in Fig. 7 for a 48×48-cells network. While the results using either of the RK4 methods, the regular or the simplified version are virtually the same (Fig. 8).

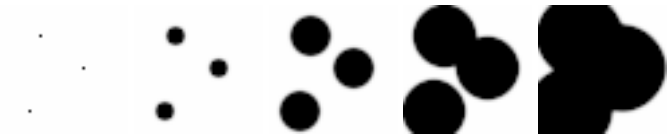

Fig. 6. Evolution of the trigger-wave front

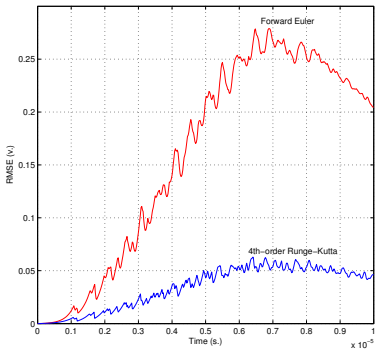

Fig. 7. FE, RK4 and simp. RK4 vs. ref. (trigger-waves)

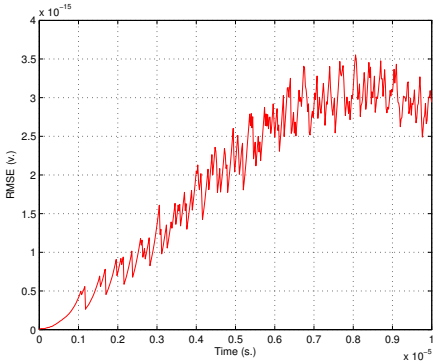

Fig. 8. Simp. method compared to regular RK4 (trigger-waves)

C. Pattern generation

Perhaps this is not as useful in image processing as the previous templates but still can be realized by autonomous CNNs. In this case, starting from a random initial state, the template encodes the organization of the states into some pattern of a prescribed form (Fig. 9). The templates employed for this operation are:

$$A = \begin{pmatrix} -0.50 & 0.00 & -0.50 \\ 0.00 & 2.00 & 0.00 \\ -0.50 & 0.00 & -0.50 \end{pmatrix} \quad \begin{matrix} B = 0 \\ z = 0 \end{matrix} \tag{21}$$

For the same τ and time step, similar results are obtained in this case, concerning the CPU times (Table I) and the maximum error (Table II). Accuracy of the RK4 methods is higher than that of FE (Fig. 10). While both verions of RK4 are equally precise (Fig. 11).

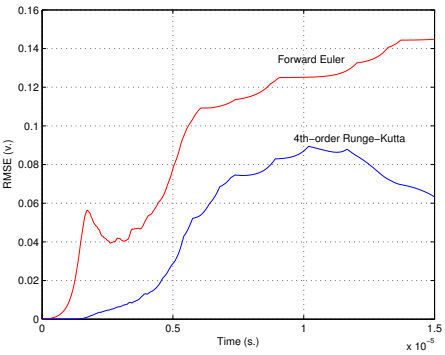

Fig. 10. FE, RK4 and simp. RK4 vs. ref. (pattern generation)

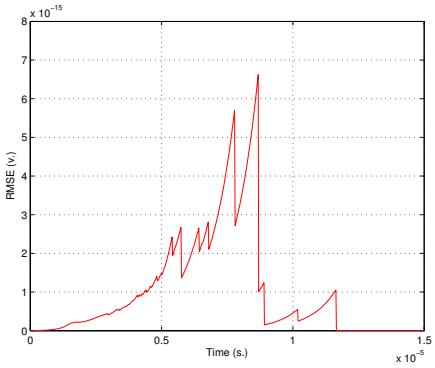

Fig. 11. Simp. vs. regular RK4 (pattern generation)

D. Edge detection

Finally, an operation with a non-zero control template B:

$$A = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad B = \begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix} \quad z = -0.5 \tag{22}$$

In this case the initial state ($\mathbf{x}(0)$) is zero and the input image is the CNN input ($\mathbf{u}$).

Once more, the outcome of the simulations points to the same results: the simplified RK4 method renders FE speed (Table I) with RK4 accuracy (Table II). Figs. 13 illustrate this fact for the simulation of the edge detection in a 48×51 image. Accuracy of both RK4 methods is the same in practice (Fig. 14).

V. Conclusions

In this paper we have presented a simplification of the state calculation in the 4th-order Runge-Kutta method that results

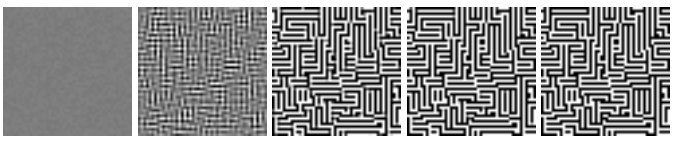

Fig. 9. Pattern formation in a 48×48 CNN from a random initial state

Fig. 12. Evolution of the cells' state

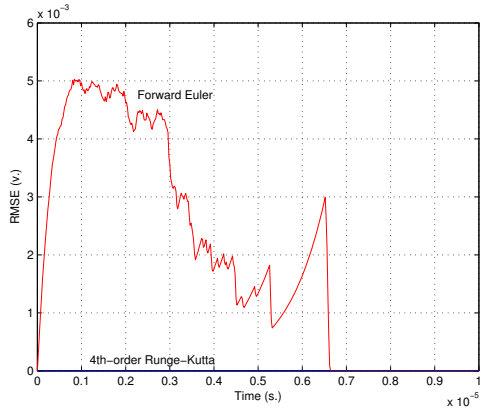

Fig. 13. FE, RK4 and simp. RK4 vs. ref. (edge detection)

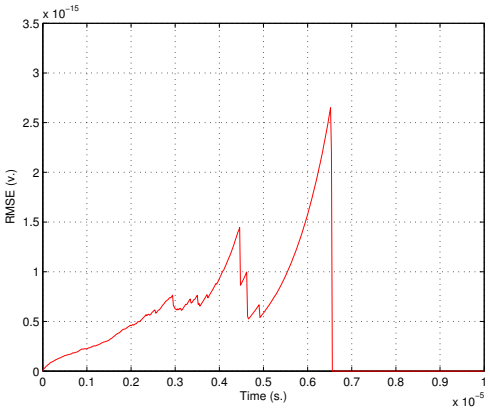

Fig. 14. Simp. method compared to regular RK4 (edge detection)

in FE speed while still maintaining RK4 accuracy. The initial assumptions, necessary in order to apply this methodology, are that input vector, feedback and control templates and bias vector are constant during the evolution of the network; the templates are linear; and that the state and output vectors can be identified. These conditions, although too restrictive for many dynamic networks, are usually met by CNNs following the FSR model and running many different image processing templates. Therefore, simulation of these networks using the proposed simplification can be done, roughly, in 25-30% of the time employed by RK4, without loosing accuracy on the simulated trajectories. Even more, for a prescribed accuracy, the time steps of the integration loops can be made larger than for the light-weighted FE method. These advantanges might not be quite relevant for off-line simulation, but can be of a lot of help for the digital emulation of a CNN under restrictive time requirements or scarce computing resources.

ACKNOWLEDGMENTS

The authors would like to thank J. Fernández-Berni for fruitful discussions on the simulation of dynamic networks. The authors would also like to thank the Office of Naval Research for Mr. Pozas-Flores' student grant, concening CNNA registration and hotel accomodation expenses during the conference. This work is partially funded by grant 2006-TIC-2352 (CICE/JA).

REFERENCES

[1] H. Harrer, A. Schuler, and E. Amelunxen, "Comparison of Different Numerical Integration Methods for Simulating CNNs". *Proc. Int. W. Cellular Neural Networks and Apps.*, pp. 151–159, Dec. 1990.

[2] C.-C. Lee and J. de Gyvez, "Single-Layer CNN Simulator". *IEEE Int. Symp. Circuits and Systems*, Vol. 6,pp. 217–220, May-Jun 1994.

[3] R. Carmona, I. García-Vargas, G. L. nán, R. Domínguez-Castro, S. Espejo, and A. R. Vázquez, "Sirena: A CAD Environment for Behavioral Modeling and Simulation of VLSI CNN Chips". *Int. J. Circuit Theory Appl.*, Vol. 27, No. 1, pp. 43–76, 1999.

[4] R. Kunz, R. Tetzlaff, and D. Wolf, "SCNN: A Universal Simulator for CNNs". *Proc. Int. W. Cellular Neural Networks and Apps.*, pp. 255–260, June 1996.

[5] T. H. Wu, B. J. Sheu, and E. Y. Chou, "Behavioral Simulation of Densely-Connected Analog Cellular Array Processors for High-Performance Computing". *Analog Integr. Circuits Signal Process.*, Vol. 10, No. 1-2, pp. 77–88, 1996.

[6] M. Hanggi, H. Reddy, and G. Moschytz, "A Sampling Theorem in Numerical Integration [Nonlinear Circuit Simulation]". *Proc. IEEE Int. Conf. Electronics, Circuits and Systems*, Vol. 1, pp. 561–564, 1999.

[7] M. Salerno, D. Casali, G. Costantini, and M. Carota, "A Customizable Tool for CNN Simulation". *Proc. European Conf. Circuit Theory and Design*, Vol. 1, pp. I/83–I/86, Aug.-Sept. 2005.

[8] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C: The Art of Scientific Computing*. New York, NY, USA: Cambridge University Press, 1992.

[9] P. Keresztes, kos Zarndy, T. Roska, P. Szolgay, T. Bezk, T. Hidvgi, P. Jns, and A. Katona, "An Emulated Digital CNN Implementation". *The Journal of VLSI Signal Processing*, Vol. 23, No. 2, pp. 291–303, Nov. 1999.

[10] J. Martínez, F. Toledo, and J. Ferrández, "New Emulated Discrete Model of CNN Architecture for FPGA and DSP Applications". *Int. Work-Conf. on Artificial and Natural Neural Networks*, pp. 33–40, Heidelberg: Springer-Verlag, 2003.

[11] S. Malki and L. Spaanenburg, "A CNN-Specific Integrated Processor". *EURASIP Journal on Advances in Signal Processing*, Vol. 2009, p. 14, 2009.

[12] R. Grech, E. Gatt, I. Grech, and J. Micallef, "Digital Implementation of CNNs". *Proc. IEEE Int. Conf. Electronics, Circuits and Systems*,. pp. 710–713, Sept. 2008.

[13] R. Dolan and G. DeSouza, "GPU-Based Simulation of CNNs for Image Processing". *Int. Joint Conf. Neural Networks*, pp. 730–735, June 2009.

[14] S. Espejo, R. Carmona, R. Domínguez-Castro, and A. R. Vázquez, "A VLSI-Oriented Continuous-Time CNN Model". *Int. J. Circuit Theory Appl.*, Vol. 24, No. 3, pp. 341–356, 1996.

[15] L. O. Chua and T. Roska, *Cellular Neural Networks and Visual Computing: Foundations and Applications*. New York, NY, USA: Cambridge University Press, 2002.

[16] L. Chua and T. Roska, "Cellular Neural Networks with Nonlinear and Delay-Type Template Elements and Non-Uniform Grids". *International Journal of Circuit Theory and Applications*, Vol. 20, pp. 449–451, 1992.

[17] F. Corinto and M. Gilli, "Comparison Between the Dynamic behaviour of Chua-Yang and Full-Range Cellular Neural Networks". *Int. Journal of Circuit Theory and Applications*, Vol. 31, No. 5, pp. 423–441, 2003.

[18] A. Rodriguez-Vazquez, G. Linan-Cembrano, L. Carranza, E. Roca-Moreno, R. Carmona-Galan, F. Jimenez-Garrido, R. Dominguez-Castro, and S. Meana, "ACE16k: the Third Generation of Mixed-Signal SIMD-CNN ACE Chips Toward VSOCs". *IEEE Trans. Circuits and Systems I*, Vol. 51, No. 5, pp. 851–863, May 2004.

[19] Z. Voroshazi, A. Kiss, Z. Nagy, and P. Szolgay, "Implementation of Embedded Emulated-Digital CNN-UM Global Analogic Programming Unit on FPGA and its Application". *Int. Journal of Circuit Theory and Applications*, Vol. 36, No. 5-6, pp. 589–603, 2008.