ANALOG INTEGRATED NEURAL-LIKE CIRCUITS FOR NONLINEAR PROGRAMMING

Angel Rodríguez-Vázquez [1], Rafael Domínguez-Castro [1], José L. Huertas [1] and Edgar Sánchez-Sinencio [2]

[1] Departamento de Electrónica y Electromagnetismo, Universidad de Sevilla, 41012-Sevilla, SPAIN
[2] Department of Electrical Engineering, Texas A&M University, College Station, TX 77843, USA

Abstract

A systematic approach is presented for the design of analog "neural" nonlinear programming solvers using Switched-Capacitor (SC) integrated circuit techniques. The method is based on formulating a dynamic gradient system whose state evolves in time towards the solution point of the corresponding programming problem. A "neuron" cell for the linear and the quadratic problem suitable for monolithic implementation is introduced. The design of this "neuron" and its corresponding "synapses" using SC techniques are considered in detail. A SC circuit architecture based on a reduced set of basic building blocks with high modularity is presented. Simulation results using a mixed-mode simulator (DIANA) [1] and experimental results from breadboard prototypes are included, illustrating the validity of the proposed techniques.

I. Introduction.

Linear and Nonlinear *Programming Solvers* are a class of analog "neural" optimizers intended for the solution of *constrained optimization* problems (also called nonlinear programming problems) in *real time*. Tank and Hopfield [2] proposed a linear programming solver with a "neural" organization. In more recent papers, [3] [4], it is demonstrated that the network of Tank and Hopfield obeys the same unifying stationary theorem as the canonical nonlinear programming circuit previously reported by Chua and Lin [5]. Kennedy and Chua's analysis, [3] [4], also yields a relationship between the solution of the network and that of the optimization problem, thus providing the foundations of a synthesis procedure for "neural" nonlinear programming solvers.

The main drawback of the circuits by Kennedy and Chua and Tank and Hopfield is that they rely on *conventional RC-active* design techniques which are not the best suited for monolithic implementation, specially taking into account that accurate resistor ratios and large RC values are required. In this paper we try to overcome this drawback by focusing on the design of programming solvers using *switched-capacitor* techniques. The inherent programmability and reconfigurability of SC circuits together with the maturity of this technique [6] in the field of analog VLSI justify the interest of our approach.

II. Synthesis technique.

The unconstrained problem.

Multiparameter optimization can be defined as the process of finding, in the parameter space, a point $\bar{x}^T=\{x_1, x_2,\dots, x_N\}$ where a *cost function* $\Psi(\bar{x})$ is minimized. Optimization in classical analog computers, [7] [8], was accomplished by implementing the following companion *dynamic gradient system*,

$$\frac{d\mathbf{x}}{dt} = -\frac{1}{\tau}\nabla\Psi(\mathbf{x}) \qquad , \tau > 0 \tag{1}$$

When $d\bar{x}/dt=\bar{0}$ for this system, implies that $\nabla\Psi=\bar{0}$, that is, the equilibrium of the gradient system coincides either with a local extreme (minimum or maximum) or with an inflection point of the corresponding cost function. On the other hand, since, for $\tau>0$, $d\bar{x}/dt$ and $\nabla\Psi$ are opposite vectors, the time evolution of $\bar{x}$ will result in $\Psi(\bar{x})$ becoming smaller and smaller as time goes on. Therefore, the process of seeking for equilibrium of the companion gradient system of a cost function, $\Psi(\bar{x})$, actually yields minimization of this cost function.

The constrained problem.

Consider the following general nonlinear programming problem [9],

$$\begin{array}{ll} \text{Minimize} & \Phi(x_1,x_2,\dots,x_N) \\ \text{subject to the constraints,} & F_k(x_1,x_2,\dots,x_N) \geq 0 \ ; \ 1\leq k\leq Q \end{array} \tag{2}$$

where N and Q are two independent integers.

To solve this problem by a gradient scheme, like that in (1), we convert it in an equivalent unconstrained problem. The way to do this is to define a *pseudo-cost function* $\Psi(\bar{x})$ as follows,

$$\Psi(\mathbf{x}) = \Phi(\mathbf{x}) + \mu P(\mathbf{x}) \tag{3}$$

where $\Phi(\bar{x})$ is the original cost function, $P(\bar{x})$ is referred to as the *penalty function* and μ is a real parameter called *penalty multiplier*.

Different penalty function alternatives can be used in practice. For a function to qualify as a valid penalty function it must monotonically increase as we move away from the region defined by the constraints (*feasibility region*), that is, it must monotonically increase as the constraints $F_k(\bullet)$ decrease from zero [10].

A drawback of the pseudocost function in (3) is that the penalty multiplier μ has to be made large enough to yield penalty dominance outside the feasibility region and thus to guarantee that the minimum of the pseudo-cost function is inside the feasibility region. Here, we propose an alternative pseudo-cost function which does not require very large penalty parameter values,

$$\Psi_m(\mathbf{x}) = U(\mathbf{F})\Phi(\mathbf{x}) + \mu P(\mathbf{x}) \tag{4a}$$

where we define,

$$U(\mathbf{F}) = \begin{cases} 1, & \text{if } F_k \geq 0 \quad \text{for every } k \\ 0, & \text{otherwise} \end{cases} \tag{4b}$$

In (4a), the function $\Phi(\bar{x})$ is disconnected outside the feasibility region. Hence, penalty dominance outside this region is guaranteed no matter what the actual value of μ is. Besides, since the penalty gradient is different from zero at any point, the solution point is forced to be either inside the feasibility region or just at the boundaries.

III. SC circuits for "neural" programming solvers.

SC building blocks.

The basic SC "neuron": Figure 1a shows the basic switched-capacitor "neuron" for quadratic programming. The clock signals controlling the switches are shown in Fig.1b.

Analysis of Fig.1 by using charge conservation principles gives,

$$\begin{aligned} y^o_{out}(n+\tfrac{1}{2}) &= \sum_{j=1}^{M} U_j(s_j)\, h_j\,[y_j^{e+}(n) - y_j^{e-}(n)] + y^e_{out}(n) \\ y^e_{out}(n+1) &= y^o_{out}(n+\tfrac{1}{2}) \end{aligned} \tag{5}$$

both polarities of the weighting factors can be obtained by grounding the appropriate input terminals. For instance, taking $y_j^+ = 0$ yields a *j-th* negative weight.

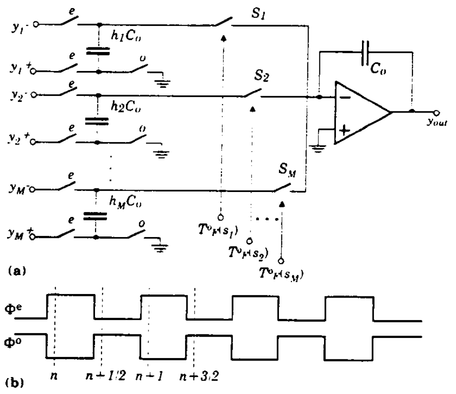
FIGURE 1: a) Basic switched-capacitor integrating "neuron"; b) Required clock signals.

Constraint evaluation summers: Constraint evaluation requires weighted summation. Figure 2 shows a switched-capacitor circuit to carry out such an evaluation. Observe that the input signals for this circuit are sensed during the odd clock phase while the output are defined only during the even clock phase. Analysis of Fig.2 allows us to obtain,

$$y^e_{out}(n+1) \doteq \sum_{j=1}^{M} h_j \left| y_j^{o+}(n+\tfrac{1}{2}) - y_j^{o-}(n+\tfrac{1}{2}) \right| \tag{6}$$

$$y^o_{out}(n+\tfrac{1}{2}) = 0$$

As for Fig.1, both negative and positive weighting factors can be easily obtained.

Threshold generation and feasibility region encoder: This block is intended to generate the threshold functions that codify constraint violations. Figure 3 shows one implementation for this block. The threshold signals codifying single constraint violations are obtained at the outputs of the comparators. On the other hand, the threshold signal that codifies whether the point is or not inside the feasibility region results as follows,

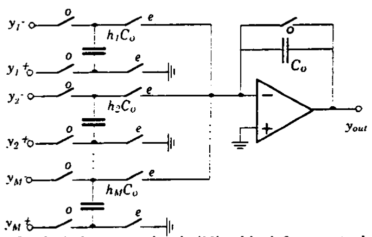
FIGURE 2: Switched-capacitor building block for constraint evaluation.

$$T^o_F(\mathbf{F}) = \overline{\sum_{k=1}^{Q} T_F(-F_k)} \wedge \theta^o = \begin{cases} 1_D \,, & F_k > 0 \quad \forall k \\ 0_D \,, & otherwise \end{cases} \tag{7}$$

where Λ here holds for the logical-AND operation.

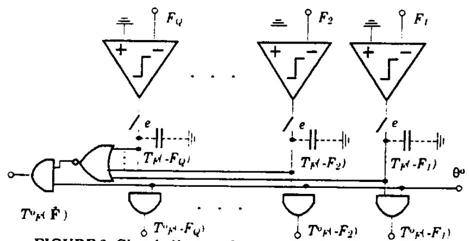
FIGURE 3: Circuit diagram for the threshold generator and feasibility region encoder.

A "neural" SC quadratic programming solver.

Figure 4 shows a switched-capacitor implementation of the quadratic programming solver "neural" network. To simplify the circuit diagram, only the *i-th* "neuron" cell and the *k-th* constraint evaluator are shown. Furthermore, we have arbitrarily used absolute value penalties and have assumed that all the coefficients of both the scalar function

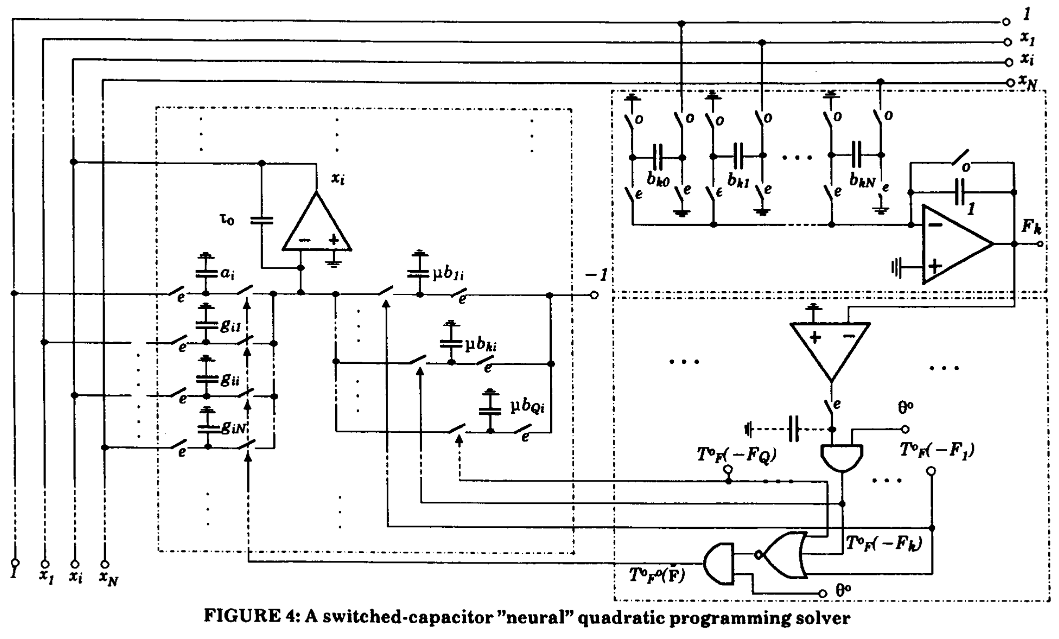
FIGURE 4: A switched-capacitor "neural" quadratic programming solver

and the constraint functions are positive. Analysis of Fig.4 by using (5), (6) and (7) gives,

$$x_i^e(n+1)=x_i^e(n)-\frac{1}{\tau_o}\left| U(\mathbf{F})\left\{a_i+\sum_{j=1}^{N} g_{ij}x_j^e(n)\right\}-\mu\sum_{k=1}^{Q} U(-F_k)b_{ki}\right| \qquad (8)$$

which corresponds to the *Forward-Euler simulation* [11] of the gradient system for the quadratic problem using the pseudo-cost function of (4) with absolute value penalties,

$$\Psi_m(\mathbf{x}) = U(\mathbf{F})\Phi(\mathbf{x})+\mu\sum_{k=1}^{Q} U(-F_k)|F_k(\mathbf{x})| \qquad (9)$$

Other types of pseudo-cost functions and penalties can be considered in a simple way [10].

Parameter μ in (8)-(9) controls the strength of the penalty. On the other hand, parameter τ_o controls the integrator gain and hence the convergence speed of the solver. In practice, choosing τ_0 requires a tradeoff between convergence speed and stability. Although decreasing τ_o increases the rate of convergence, it may produce instabilities.

Alternative SC building blocks.

Parasitic-insensitive "neuron" and summer: All previously shown SC circuits are sensitive to parasitics [11]. Figure 5 shows a threshold-controlled "neuron" that is insensitive to both parasitic capacitors and opamp offset voltage [12]. Along with the figure we include a formula describing the operation of the circuit. Since appropriate operation of the "neural" architecture requires a half-period delay in the operation of the "neuron", only positive weights are allowed for the circuit in Fig.5. Negative weights can be realized by changing the sign of the input signals. It requires generation of the inverted version of the output of each neuron, what is done by the second amplifier in Fig.5.

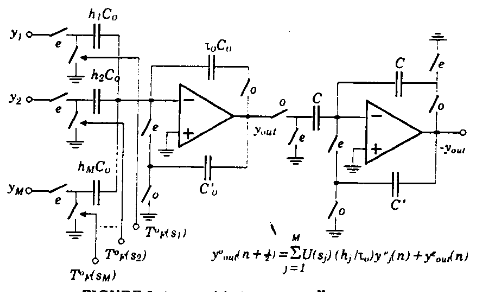

FIGURE 5: A parasitic-insensitive "neuron".

Figure 6 shows a switched-capacitor summer which can be used for parasitic- insensitive constraint evaluation. As for Fig.5, only positive weights are allowed for this summer.

The output signal of the "neuron" of Fig.5 is only valid at odd time instances. This imposes the necessity of changing the "synapses" strategy. The new strategy is illustrated in Fig.7. The integrator-summer Σ_2 has to be implemented by using Fig.5, while Fig.6 must be used for Σ_1. The rest of the architectural concepts leading to Fig.4 are still valid for the parasitic-insensitive building blocks.

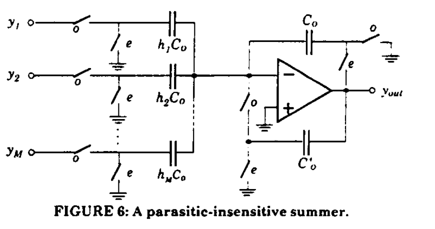

FIGURE 6: A parasitic-insensitive summer.

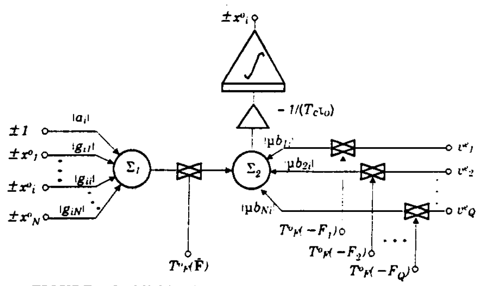

FIGURE 7: Stablishing "synapses" with parasitic-insensitive switched-capacitor building blocks.

IV. Stability Properties

Computation in a "neural" programming solver is a dynamical process of seeking equilibrium. For proper computation, the equilibrium points of a programming solver must correspond to solutions of the problem being computed; moreover, these equilibria must be stable.

The dynamic behavior of a switched-capacitor programming solver is determined by two factors, namely,

1) The reactive parasitics associated with the active components and interconnection wires.
2) The parasitics introduced by the discretizing numerical integration algorithm.

However, since the proposed architectures do not exhibit any global closed-loop during the computation cycle, there is no possibility for instabilities due to continuous-time parasitics to appear. Hence, only discrete-time parasitics must be taken into account for stability analysis.

A switched-capacitor programming solver is a discrete-time dynamic system which can be described by the following difference *state equation,*

$$\mathbf{x}(n+1)=\mathbf{G}(\mathbf{x}(n),\tau_o,\mu) \qquad (10)$$

where $\bar{G}$ is a vector function and τ_o and μ are positive real parameters (these parameters are used in Fig.4 as capacitor sizing parameters).

Equilibrium [13] occurs at the point $\bar{x}^*$ which is a constant solution of (10),

$$\mathbf{x}^*=\mathbf{G}(\mathbf{x}^*,\tau_o,\mu) \qquad (11)$$

In general, the equilibrium points of (10) can be either: 1) inside the feasibility region or 2) on the boundary of this region.

Solution inside the feasibility region. *Local asymptotic stability* can be calculated by first linearizing (10) around the equilibrium point and then taking the z-transform. In order to obtain the equilibrium point to be stable, the root of the characteristic equation must be inside the unit circle [14]. Hence, parameter τ_o has to be selected large enough for the local stability condition to be fulfilled.

Solution on the boundary the feasibility region. It is clear in this case that the system cannot stay static at $\bar{x}^*$. However, the solution $\bar{x}(n)$, $\bar{x}(n+1)$, $\bar{x}(n+2)$....can be made to stay inside an arbitrary small interval around this point. In other words, this system can be made to be stable in the sense that the variations of the solution remain bounded, although it is not asymptotically stable.

From (8), it can be seen that the maximum deviation from the equilibrium point towards the exterior of the acceptability region, for the quadratic problem, is given by

$$|\Delta x_i|=\frac{1}{\tau_0}\left| a_i+\sum_{j=1}^{N} g_{ij}x_j^*\right| \qquad \forall i \qquad (12)$$

while the maximum deviation towards the interior of this region is given by,

$$|\Delta x_i| = \frac{\mu}{\tau_0}\left|\sum_{k=1}^{M} b_{ki}\right| \qquad \forall i \tag{13}$$

where M is the number of constraints that are simultaneously violated.

From the previous equations, it can be seen that oscillations can be made as small as desired by simply varying, in a suitable way, the values of τ_o and μ and provided that these values fulfill the condition of stability in the acceptability region.

V. Practical results

Linear Programming.

The architecture of Fig.4 has been breadboarded for the following two-dimensional linear problem

$$\text{Minimize} \qquad \Phi = a_1x_1 + a_2x_2$$

subject to the constraints,

$$F_1 = -\frac{5}{12}x_1 + x_2 + \frac{35}{12} \geq 0 \qquad F_2 = -\frac{5}{2}x_1 - x_2 + \frac{35}{2} \geq 0 \tag{14}$$

$$F_3 = x_1 + 5 \geq 0 \qquad F_4 = x_2 + 5 \geq 0$$

Fig.8a,b correspond to this problem with $a_1 = -1$, $a_2 = 1$. The theoretical solution is at $x_1 = 7$, $x_2 = 0$. The evolution towards this point from two different initial points is illustrated. In a similar way Fig.8c,d correspond to $a_1 = 1$, $a_2 = -1$, the theoretical solution being at $x_1 = -5$, $x_2 = 5$. As before, the dynamic route of the actual circuit towards the theoretical equilibrium is observed for two different initial points.

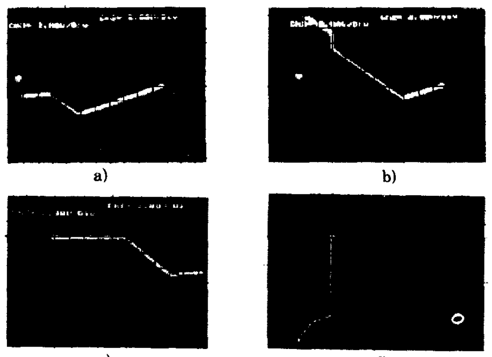

FIGURE 8: a,b) Oscilloscope pictures showing two trajectories for the problem in (14) with $a_1 = -1$, $a_2 = 1$
c,d) The same for the problem in (14) with $a_1 = 1$, $a_2 = -1$
(horizontal signal: x_1, 2V/dv; vertical signal: x_2, 2V/dv)

A Quadratic programming problem.

We have used parasitic-insensitive building blocks to solve the following quadratic problem

$$\text{Minimize} \qquad \Phi = x_1^2 + x_2^2 + x_1x_2 + 3x_1 + 3x_2$$

subject to the constraints,

$$F_1 = -\frac{5}{12}x_1 + x_2 + \frac{35}{12} \geq 0 \qquad F_2 = \frac{5}{2}x_1 - x_2 + \frac{35}{2} \geq 0 \tag{15}$$

$$F_3 = x_1 + 5 \geq 0 \qquad F_4 = -x_2 + 5 \geq 0$$

Table 1 shows the DIANA-simulated values of the variables x_1 and x_2 as functions of different parameters, namely: the *time instant* (measured as the number of clock periods from the initial point, rightmost column in Table 1); the *DC gain of the opamps* used in the integrating "neuron" (parameter A_o in the first row); and the *size of the integrating capacitors* (parameter τ_0 at the left of the Table). In all the cases the initial point was at $x_1 = x_2 = 0$. As can be seen, the convergence speed increases as τ_0 decreases. Furthermore, decreasing the opamp DC gain does not significantly influence the performance of the circuit.

	$A_o=30000$ x_1, x_2	$A_o=100$ x_1, x_2	*Number of clock periods*
	-1.27, -0.71	*-1.26, -0.72*	100
$\tau_0 = 10$	*-1.03 , -0.97*	*-1.03, -0.97*	200
	-1.002 , -0.995	*-1.002, -0.995*	300
	-1.03 , -0.97	*-1.025 , -0.97*	100
$\tau_0 = 5$	*-0.999 , -0.998*	*-0.999, -0.998*	200
	-0.998 , -0.998	*-0.998, -0.998*	300

TABLE 1: Evolution of the solution point as function of different circuit parameters

VI. Conclusions

A unified systematic approach has been presented for the design of "neural" quadratic programming solvers using SC techniques. This architecture is based on a reduced set of basic cells that are interconnected following a dense regular pattern. The resulting circuits exhibit a very high modularity and in this sense can be considered as members of the general family of analog "neural" networks. The proposed method is valid for both linear and quadratic programming problems, and the basic network architecture can be adapted to more general nonlinear problems.

References

[1] G. Arnout, Ph. Reynaert, L. Claesen and D. Dumlugol: *"DIANA V7E user's guide"*. ESAT Laboratory, Katholieke Universiteit Leuven, 1983.

[2] D.A. Tank and J.J. Hopfield: "Simple Neural Optimization Networks: An A/D Converter, Signal Decision Circuit, and a Linear Programming Circuit". *IEEE Trans. Circuits Systems*, Vol. 33, pp 533-541, May 1986.

[3] M.P. Kennedy and L.O. Chua: "Neural Networks for Nonlinear Programming". *IEEE Trans. Circuits and Systems*, Vol. 35, pp 554-562, May 1988.

[4] M.P. Kennedy and L.O. Chua: "Unifying the Tank and Hopfield Linear Programming Circuit and the Canonical Nonlinear Programming Circuit of Chua and Lin". *IEEE Trans. Circuits and Systems*, Vol. 34, pp 210-214, February 1987.

[5] L.O. Chua and G.N. Lin.: "Nonlinear Programming without computation". *IEEE Trans. Circuits Systems*, Vol. 31, pp 182-188, Feb. 1984.

[6] C.W. Solomon: "Switched-Capacitor Filters". *IEEE Spectrum*, Vol 25, pp 28-32, June 1988.

[7] A. Hausner: *"Analog and Analog/Hybrid Computer Programming"*. Prentice-Hall 1971.

[8] G.A. Korn and T.M. Korn: *"Electronic Analog and Hybrid Computers- 2nd Edition"*. Mc Graw-Hill 1972.

[9] G. V. Vanderplaats: *"Numerical Optimization Techniques for Engineering Design: with Applications"*. McGraw-Hill 1984.

[10] R. Domínguez-Castro: *"Switched-Capacitor Neural Networks for Optimization Problems"*. Master Thesis, University of Seville 1989.

[11] R. Gregorian and G.C. Temes: *"Analog MOS Integrated Circuits for Signal Processing"*. Wiley-Interscience 1986.

[12] F. Maloberti: "Switched-Capacitor building blocks for Analogue Signal Processing", Electronics Letters, Vol. 19, pp 263-265, 1983

[13] B.C. Kuo: *"Digital Control Systems"*. Holt, Rinehart and Winston 1980.