WAVEFORM RELAXATION: THEORY AND PRACTICE

by

J. White, F. Odeh, A. S. Vincentelli

and A. Ruehli

Memorandum No. UCB/ERL M85/65

30 July 1985

WAVEFORM RELAXATION: THEORY AND PRACTICE

by

J. White, F. Odeh, A. S. Vincentelli, and A. Ruehli

Memorandum No. UCB/ERL M85/65

30 July 1985

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

WAVEFORM RELAXATION: THEORY AND PRACTICE

J. White, F. Odeh, A. S. Vincentelli, A. Ruehli

Abstract: This paper surveys the family of Waveform Relaxation Methods for solving large systems of ordinary nonlinear differential equations. The basic WR algorithm will be reviewed, and many of the derivative algorithms will be presented, along with new convergence proofs. In addition, examples will be analyzed that illustrate several of the implementation techniques used to improve the efficiency of the basic WR algorithm, along with theoretical results that indicate the strengths or limitations of these techniques.

INTRODUCTION

The tremendous increase in complexity of engineering design and availability of computing resources has made computer simulation an important and heavily used tool for both research and engineering design. Since many simulation problems are formulated as large systems of nonlinear ordinary differential equations (ODE's), much research work has been devoted to solving ODE systems efficiently.

The standard approach to solving ODE systems is based on three techniques [1], [2]:

i) Stiffly stable implicit integration methods, such as the Backward Difference formulas, to convert the differential equations which describe the system into a sequence of nonlinear algebraic equations.

ii) Modified Newton methods to solve the algebraic equations by solving a sequence of linear problems.

iii) Sparse Gaussian Elimination to solve the systems of linear equations generated by the Newton method.

The above approach can become inefficient for large systems where different state variables are changing at very different rates. This is because the direct application of the integration method forces every differential equation in the system to be discretized identically, and this discretization must be fine enough so that the fastest changing state variable in the system is accurately represented. If it were possible to pick different discretization points, or timesteps, for each differential equation in the system so that each could use the largest timestep that would accurately reflect the behavior of its associated state variable, then the efficiency of the simulation would be greatly improved.

Several modifications of the direct method have been used that allow the individual equations of the system to use different timesteps[3][4][5][6][7][8]. The approach that will be discussed in this paper is the family of Waveform Relaxation algorithms [9A,9B,9C]. WR algorithms have captured considerable attention due to their favorable numerical properties and to the success in applying the WR algorithms to the solution of Metal-Oxide-Semiconductor (MOS) digital circuits.

In this paper we will both survey the current state of research in WR algorithms, and present new theoretical and practical results The paper is organized as two parts. In the first part we will present the theoretical background for the basic WR algorithm and it derivatives. We will start by introducing waveform relaxation with a simple example, and follow with the basic algorithm. Then a new proof of the convergence, one that demonstrates that the WR algorithm is a contraction mapping in a particular norm, will be presented. Extensions to the basic algorithm that allow for modified iteration equations (including discrete approximations) will be presented and it will be shown that the convergence of such extensions follows *directly* from the proof that the WR algorithm is a contraction mapping. The extension of the Newton Method to function spaces will then be presented, and its convergence proved using lemmas from the

basic theorem. Finally, discretization approximations will be considered in more detail, by comparing relaxation and explicit integration methods for a sample stiff problem.

In the second part we will analyze examples that illustrate several of the implementation techniques used to improve the efficency of the basic WR algorithm, and prove theorems that indicate the strengths or limitations of these techniques. We will start by considering approaches for partitioning large systems into loosely coupled subsystems. We will then examine how breaking the simulation interval into pieces, called windows, can be used to reduce the number of relaxation iterations required to achieve convergence. Two techniques for reducing the iteration computation will then be presented. The first is based on performing one iteration of a Newton method with each relaxation iteration, and the second is based on exploiting piecewise linearity. Because the WR algorithm has proved to be an efficient technique for simulating MOS digital circuits, the examples used throughout this paper are drawn from this area. In order to more clearly demonstrate both the practicality of the WR algorithm, and the specific nature of its efficiencies, we will end the second section, and the paper, by examining in detail application of the WR algorithm to the simulation of MOS digital circuits.

SECTION 1 - THE WR ALGORITHM AND ITS DERIVATIVES

SECTION 1.1 - THE BASIC WR ALGORITHM

We will start this section with a simple illustrative example, and then present the general WR algorithm. Consider the first-order two-dimensional differential equation in: $x(t) \in \mathbb{R}^2$ on $t \in [0,T]$.

$$\dot{x}_1 = f_1(x_1, x_2, t) \qquad x_1(0) = x_{10} \tag{1.1.1a}$$

$$\dot{x}_2 = f_2(x_1, x_2, t) \qquad x_2(0) = x_{20} \tag{1.1.1b}$$

The basic idea of the waveform-relaxation algorithm is to fix the waveform $x_2: [0,T] \rightarrow \mathbb{R}$ and solve Eqn. (1.1.1a) as a one dimensional differential equation in

$x_1(t)$. The solution thus obtained for $x_1(t)$ can be substituted into Eqn. (1.1.1b) which will then reduce to another first-order differential equation in one variable, $x_2(t)$. Eqn. (1.1.1a) is then re-solved using the new solution for $x_2(t)$ and the procedure is repeated.

Alternately, fix the waveform $x_2(t)$ in Eqn. (1.1.1a) and fix $x_1(t)$ in Eqn. (1.1.1b) and solve both one dimensional differential equations simultaneously. Use the solution obtained for x_2 in Eqn. (1.1.1b) and the solution obtained for x_1 in Eqn. (1.1.1a) and re-solve both equations.

In this fashion, iterative algorithms have been constructed. Either replaces the problem of solving a differential equation in two variables by one of solving a sequence of differential equations in one variable. As described above, these two waveform relaxation algorithms can been seen as the analogues of the Gauss-Seidel and the Gauss-Jacobi techniques for solving nonlinear algebraic equations. Here, however, the unknowns are waveforms (elements of a function space), rather than real variables. In this sense, the algorithms are techniques for time-domain decoupling of differential equations.

The most general formulation of a system of nonlinear differential equations is the following implicit formulation:

$$F(\dot{x}(t), x(t), u(t)) = 0 \qquad x(0) = x_0 \tag{1.1.2}$$

where $x(t) \in \mathbb{R}^n$ on $t \in [0,T]$; $u(t) \in \mathbb{R}^r$ on $t \in [0,T]$ is piecewise continuous; and $F: \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^r \rightarrow \mathbb{R}^n$ is continuous.

In order to guarantee that WR applied to Eqn. (1.1.2) will converge to the system's solution, we first must guarantee that Eqn. (1.1.2) has a solution. If we require that there exists a transformation of Eqn. (1.1.2) to the form $\dot{y} = f(y,u)$ where f is Lipshitz continuous with respect to y for all u, then a unique solution for the system exists[22]. Although there are many sets of broad constraints on F that

guarantee the existence of such a transformation, the conditions can be difficult to verify in practice. In addition, for the above system, it is difficult to determine how to assign variables to equations when applying the WR algorithm. That is, when solving the F_i equation of the system in the iteration process, what x_j variable should be solved for implicitly. If a poor choice is made, the relaxation may not converge[9B].

Rather than carefully considering the existence and assignment questions, which will complicate the analysis that follows without lending much insight, we will consider the following less general form, in which many practical problems, particularly circuit simulation, can be described.

$$C(x(t), u(t))\,\dot{x}(t) = f(x(t), u(t)) \qquad x(0) = x_0 \tag{1.1.3}$$

where $x(t) \in \mathbb{R}^n$ on $t \in [0,T]$; $u(t) \in \mathbb{R}^r$ on $t \in [0,T]$ is piecewise continuous; $C: \mathbb{R}^n x \mathbb{R}^r \rightarrow \mathbb{R}^{nxn}$ is such that $C(x, u)^{-1}$ exists and is uniformly bounded with respect to x, u ; and $f: \mathbb{R}^n x \mathbb{R}^r \rightarrow \mathbb{R}^n$ is globally Lipschitz continuous with respect to x for all $u(t) \in \mathbb{R}^r$.

The fact that $C(x, u)$ has a well-behaved inverse guarantees the existence of a normal form for Eqn (1.1.3), and that $x(t) \in \mathbb{R}^n$ is the vector of state variables for the system. Then as f is globally Lipschitz with respect to x for all u, $C(x, u)^{-1}$ is uniformly bounded, and $u(t)$ is piecewise continuous, there exists a unique solution to Eqn. (1.1.3).

The WR algorithm for solving the above system is as follows:

<u>Algorithm 1.1.1 (WR Gauss-Seidel Algorithm for solving Eqn. (1.1.3))</u>

Comment:
The superscript k denotes the iteration count,
the subscript i denotes the component index
of a vector and ϵ is a small positive number.

$k \leftarrow 0$;
guess waveform $x^0(t)$; $t \in [0,T]$
such that $x^0(0) = x_0$
(for example, set $x^0(t) = x_0$, $t \in [0,T]$);

repeat {

$k \leftarrow k+1$

foreach ($i \in \{1,\ldots,n\}$ {

solve

$$\sum_{j=1}^{i} C_{ij}(x_1^k, \cdots, x_i^k, x_{i+1}^{k-1}, \cdots, x_n^{k-1}, u)\dot{x}_j^k +$$

$$\sum_{j=i+1}^{n} C_{ij}(x_1^k, \cdots, x_i^k, x_{i+1}^{k-1}, \cdots, x_n^{k-1}, u)\dot{x}_j^{k-1} -$$

$$f_i(x_1^k, \cdots, x_i^k, x_{i+1}^{k-1}, \cdots, x_n^{k-1}, u) = 0$$

for ($x_i^k(t)$; $t \in [0,T]$), with the initial condition $x_i^k(0) = x_{i_0}$.

}

} **until** ($\max_{1 \leqslant i \leqslant n} \max_{t \in [0,T]} |x_i^k(t) - x_i^{k-1}(t)| \leqslant \epsilon$)

that is, until the iteration converges. ■

Note that the differential equation in Algorithm (1.1.1) has only one unknown variable x_i^k . The variables $x_{i+1}^{k-1}, \cdots, x_n^{k-1}$ are known from the previous iteration and the variables $x_1^k, \cdots, x_{i-1}^k$ have already been computed. Also, the Gauss-Jacobi version of the WR Algorithm for Eqn. (1.1.3) can be obtained from Algorithm (1.1.1) by replacing the **foreach** statement with the **forall** statement and adjusting the iteration indices.

SECTION 1.2 - CONVERGENCE PROOF FOR THE BASIC WR ALGORITHM

If the matrix $C(x,u)$ is diagonally dominant and Lipschitz continous with respect to x for all u then both the Gauss-Seidel and the Gauss-Jacobi versions of Algorithm (1.1.1) are guaranteed to converge. In [9A], it was shown that the WR algorithm converges when applied to Eqn. (1.1.3) if $C(x,u)$ is diagonally dominant and independent of x. As many systems that are modeled in the form of Eqn. (1.1.3) include a dependence of C on x, we will present a more general convergence proof that extends the original theorem to include these systems. In addition, we will prove the WR method is a contraction in a simpler norm than the one used in the original theorem.

We will prove the theorem by first showing that if $C(x,u)$ is diagonally dominant, then there exists a bound on the $\dot{x}^k$'s generated by the WR algorithm that is independent of k. Using this bound, we will show that the assumption that $C(x,u)$ is Lipschitz continuous implies there exists a norm on $\mathbb{R}^n$ such that for arbitrary positive integers j and k:

$$\|\dot{x}^{k+1}(t)-\dot{x}^{j+1}(t)\| \leqslant \gamma\|\dot{x}^k(t)-\dot{x}^j(t)\| + l_1\|x^{k+1}(t)-x^{j+1}(t)\| + l_2\|x^k(t)-x^j(t)\|$$

for some $\gamma<1$ and $l_1, l_2 < \infty$ for all $t \in [0,T]$. In the properly chosen norm $\|\bullet\|_b$ on $C([0,T],\mathbb{R}^n)$, the above equation implies that

$$\|\dot{x}^{k+1}-\dot{x}^{j+1}\|_b < \|\dot{x}^k-\dot{x}^j\|_b$$

and therefore the sequence $\{\dot{x}^k\}$ converges by the contraction mapping theorem. As $x^k(0) = x_0$ for all k, $\{x^k\}$ converges as well.

Before formally proving this basic WR convergence theorem we will state the well-known contraction mapping theorem[16], and a few lemmas which will be used in the course of the proof.

<u>The Contraction Mapping Theorem:</u> Let Y be a Banach space and $F:Y \to Y$. If F is such that $\|F(y)-F(x)\| \leqslant \gamma\|y-x\|$ for all $x,y \in Y$, for some $\gamma \in [0,1)$, then F has a unique fixed point $\tilde{y}$ such that $F(\tilde{y}) = \tilde{y}$. Furthermore, for any initial guess $y^0 \in Y$ the sequence $\{y^k \in Y\}$ generated by the fixed point algorithm $y^k = F(y^{k-1})$ converges uniformly to $\tilde{y}$.

<u>Lemma 1.2.1:</u> If $C(x,u) \in \mathbb{R}^{n\times n}$ is diagonally dominant uniformly over all $x \in \mathbb{R}^n$, $u \in \mathbb{R}^r$ then given any collection of vectors $\{x^1,\cdots,x^n\}$, $x^i \in \mathbb{R}^n$, and any $u \in \mathbb{R}^r$, the matrix $C^p(x^1,\cdots,x^n,u) \in \mathbb{R}^{n\times n}$ defined by $C^p_{ij}(x^1,\cdots,x^n,u) = C_{ij}(x^i,u)$ is also diagonally dominant. In other words, let C^p be the matrix constructed by setting the ith row of C^p equal to the ith row of the given matrix $C(x^i,u)$. Then this new matrix is also diagonally dominant.

Lemma 1.2.1 follows directly from the definition of diagonal dominance.

Lemma 1.2.2: Let $C \in \mathbb{R}^{nxn}$ be any strictly diagonally dominant matrix. Let L, strictly lower triangular, U, strictly upper triangular, and D, diagonal, be such that $C = L + D + U$. Then $\|D^{-1}(L+U)\|_\infty < 1$ and $\|(D+L)^{-1}U\|_\infty < 1$.

Lemma 1.2.2 is a standard result in matrix theory[24].

Lemma 1.2.3: Let $x, y \in C([0,T],\mathbb{R}^n)$. If there exists some norm on $\mathbb{R}^n$ such that

$$\|\dot{x}(t)\| \leqslant \gamma\|\dot{y}(t)\| + l_1\|x(t)\| + l_2\|y(t)\| \tag{1.2.1}$$

for some positive numbers $l_1, l_2 < \infty$ and $\gamma < 1$ then there exists a norm $\|\bullet\|_b$ on $C([0,T],\mathbb{R}^n)$ and a positive number $\alpha < 1$ such that

$$\|\dot{x}\|_b \leqslant \alpha\|\dot{y}\|_b + l_1\|x(0)\| + l_2\|y(0)\| \tag{1.2.2}$$

Proof of Lemma 1.2.3:

Substituting $\int_0^t \dot{x}(\tau)d\tau + x(0)$ for $x(t)$ in Eqn. (1.2.1), performing an analogous substitution for $y(t)$, multiplying the entire equation by e^{-bt}, and moving the norms inside the integral yields:

$$e^{-bt}\|\dot{x}(t)\| \leqslant \gamma e^{-bt}\|\dot{y}(t)\| + l_1 e^{-bt}\int_0^t \|\dot{x}(\tau)\|d\tau + l_1 e^{-bt}\|x(0)\| + l_2 e^{-bt}\int_0^t \|\dot{y}(\tau)\|d\tau + l_2 e^{-bt}\|y(0)\|. \tag{1.2.3}$$

Let $\|\bullet\|_b$ be defined by $\|f\|_b = \max_{[0,T]} e^{-bt}\|f(t)\|$. This is a norm on $C([0,T],\mathbb{R}^n)$ for any finite positive number $b > 0$ and is equivalent to the uniform norm on $C([0,T],\mathbb{R}^n)$. Then Eqn. (1.2.3) implies

$$\|\dot{x}\|_b \leqslant \gamma\|\dot{y}\|_b + \max_{[0,T]}[\, l_1 e^{-bt}\int_0^t e^{b\tau}d\tau\, \|\dot{x}\|_b + l_1 e^{-bt}\|x(0)\| + l_2 e^{-bt}\int_0^t e^{b\tau}d\tau\, \|\dot{y}\|_b + l_2 e^{-bt}\|y(0)\|\,] \tag{1.2.4}$$

And since $e^{-bt}\int_0^t e^{b\tau}d\tau \leqslant \frac{1}{b}$, then for $b > l_1$ we can write

$$\|\dot{x}\|_b \leqslant \frac{\gamma+l_2 b^{-1}}{1-l_1 b^{-1}}\|\dot{y}\|_b + l_1\|x(0)\| + l_2\|y(0)\|. \tag{1.2.5}$$

In this case γ is less than 1, so there exists a finite B for which $\frac{(\gamma+l_2 B^{-1})}{1-l_1 B^{-1}} = \alpha < 1$.

Let the b in Eqn. (1.2.5) be set equal to this B to get

$$\|\dot{x}\|_B \leqslant \alpha\|\dot{y}\|_B + l_1\|x(0)\| + l_2\|y(0)\| \tag{1.2.6}$$

which completes the proof. ■

Now we prove the following WR convergence theorem for systems of equations of the form of Eqn (1.1.3).

Theorem 1.2.1: If, in addition to the assumptions of Eqn. (1.1.3), $C(x(t),u(t)) \in \mathbb{R}^{nxn}$ is strictly diagonally dominant uniformly over all $x(t) \in \mathbb{R}^n$ and $u(t) \in \mathbb{R}^r$ and Lipschitz continuous with respect to $x(t)$ for all $u(t)$, then the sequence of waveforms $\{ x^k \}$ generated by the Gauss-Seidel or Gauss-Jacobi WR algorithm will converge uniformly to the solution of Eqn. (1.1.3) for all bounded intervals $[0,T]$.

Proof of Theorem 1.2.1:

We will present the proof only for the Gauss-Seidel WR algorithm, as the proof for the Gauss-Jacobi case is almost identical. The equations for one iteration of the Gauss-Seidel WR algorithm applied to Eqn. (1.1.3) can be written in matrix form as

$$\hat{C}(x^{k+1}, x^k, u)\dot{x}^{k+1} = \hat{f}(x^{k+1}, x^k, u)$$

where $\hat{C}_{ij}(x^{k+1}, x^k, u) = C_{ij}(x_1^{k+1}, \cdots, x_i^{k+1}, x_{i+1}^k, \cdots, x_n^k, u)$ and $\hat{f}_i(x^{k+1}, x^k, u) = f_i(x_1^{k+1}, \cdots, x_i^{k+1}, x_{i+1}^k, \cdots, x_n^k, u)$. Let $\hat{C}(x^{k+1}, x^k, u) = L_{k+1} + D_{k+1} - U_{k+1}$ where L_{k+1} is strictly lower triangular, U_{k+1} is upper triangular, and D_{k+1} is diagonal (Note that by Lemma 1.2.1, the matrix $\hat{C}$ is diagonally dominant because C is diagonally dominant). Rearranging the iteration equation yields:

$$\dot{x}^{k+1} = (L_{k+1}+D_{k+1})^{-1}\,[\,U_{k+1}\dot{x}^{k} + \hat{f}\,(x^{k+1}, x^{k}, u)\,]. \tag{1.2.7}$$

Taking the difference between Eqn. (1.2.7) at iteration $k+1$ and at iteration $j+1$ yields

$$\dot{x}^{k+1}-\dot{x}^{j+1} = (L_{k+1}+D_{k+1})^{-1}U_{k+1}\dot{x}^{k} - (L_{j+1}+D_{j+1})^{-1}U_{j+1}\dot{x}^{j} + \tag{1.2.8}$$
$$(L_{k+1}+D_{k+1})^{-1}\hat{f}\,(x^{k+1}, x^{k}, u) - (L_{j+1}+D_{j+1})^{-1}\hat{f}\,(x^{j+1}, x^{j}, u)$$

Using the Lipschitz continuity of $\hat{f}$ and that $\|(L_{k+1}+D_{k+1})^{-1}\| < K$ for some $K<\infty$ independent of x and k, (because $C(x,u)$ is uniformly diagonally dominant with respect to x) in Eqn. 1.2.8 leads to

$$\|\dot{x}^{k+1}(t)-\dot{x}^{j+1}(t)\| \leqslant l_1K\|x^{k+1}(t)-x^{j+1}(t)\| + l_2K\|x^{k}(t)-x^{j}(t)\| + \tag{1.2.9}$$
$$\|(L_{k+1}+D_{k+1})^{-1}-(L_{j+1}+D_{j+1})^{-1}\|\;\|\hat{f}\,(x^{j+1}, x^{j}, u)\| +$$
$$\|(L_{k+1}+D_{k+1})^{-1}U_{k+1}\dot{x}^{k}(t)-(L_{j+1}+D_{j+1})^{-1}U_{j+1}\dot{x}^{j}(t)\|$$

where l_1 is the Lipschitz constant of $\hat{f}$ with respect to its first argument, and l_2 is the Lipschitz constant of $\hat{f}$ with respect to its second argument. That $C(x,u)$ is uniformly diagonally dominant and Lipschitz continuous with respect to x for all u implies $(L_k+D_k)^{-1}$ and $(L_k+D_k)^{-1}U_k$ are also Lipschitz continuous in the same manner. It then follows that there exist some positive finite numbers k_1, k_2, k_3, k_4 such that

$$\|\dot{x}^{k+1}(t)-\dot{x}^{j+1}(t)\| = l_1K\|x^{k+1}(t)-x^{j+1}(t)\| + l_2K\|x^{k}(t)-x^{j}(t)\| + \tag{1.2.10}$$
$$[k_3\|x^{k+1}(t)-x^{j+1}(t)\| + k_4\|x^{k}(t)-x^{j}(t)\|]\,\|\hat{f}\,(x^{j+1}, x^{j}, u)\| +$$
$$[k_1\|x^{k+1}(t)-x^{j+1}(t)\| + k_2\|x^{k}(t)-x^{j}(t)\|]\,\|\dot{x}^{k}(t)\| + \gamma\|\dot{x}^{k}(t)-\dot{x}^{j}(t)\|$$

where k_1 is the Lipschitz constant of $(L_k+D_k)^{-1}U_k$ with respect to its first x argument (see definition of L_k, U_k and D_k above), k_2 is the Lipschitz constant with respect to the second x argument, k_3 and k_4 are the Lipschitz constants for $(L_k+D_k)^{-1}$ with respect to its first and second x arguments, and γ is such that $\|(L_k+D_k)^{-1}U_k\| < \gamma < 1$ independent of k (by Lemma 1.2.2).

To establish a bound on the terms in Eqn. (1.2.10) involving $\|\dot{x}^{k}(t)\|$ and $\|\hat{f}\,(x^{j+1}, x^{j}, u)\|$ it is necessary to show that the $\dot{x}^{k}$'s and therefore the x^{k}'s and $\hat{f}\,(\bullet)$'s are bounded *a priori*. We prove such a bound exists in the following lemma.

Lemma 1.2.4: If $C(x,u)$ in Eqn. (1.1.3) is strictly diagonally dominant and Lipschitz continuous then the $\dot{x}^k(t)$'s produced by Algorithm (1.1.1) are bounded independent of k .

Proof of Lemma 1.2.4

If $\|\bullet\|$ is the l_∞ norm on $\mathbb{R}^n$, by Lemma 1.2.1 $\|(L_{k+1}+D_{k+1})^{-1}U_{k+1}\| < 1$. From Eqn. (1.2.7),

$$\|\dot{x}^{k+1}(t)\| < \gamma\|\dot{x}^k(t)\| + \|(L_{k+1}+D_{k+1})^{-1}\| \, \|\hat{f}(x^{k+1}(t), x^k(t), u)\| \quad (1.2.11)$$

for some positive number $\gamma < 1$. As $f(x,u)$ is globally Lipschitz continuous with respect to x , there exist finite positive constants l_1, l_2 such that

$$\|\hat{f}(x,y,u) - \hat{f}(w,z,u)\| < l_1\|x-w\| + l_2\|y-z\| \quad (1.2.12)$$

for all $u,x,y,w,z \in \mathbb{R}^n$. From Eqn. (1.2.11) and Eqn. (1.2.12) and using the fact that $\|(L_{k+1}+D_{k+1})^{-1}\|$ is bounded by some $K < \infty$ for all k :

$$\|\dot{x}^{k+1}(t)\| \leqslant \gamma\|\dot{x}^k(t)\| + l_1K\|x^{k+1}(t)\| + l_2K\|x^k(t)\| + K\|\hat{f}(0,0,u)\| \quad (1.2.13)$$

Eqn. (1.2.13) is in the form to apply a slightly modified Lemma 1.2.3. Therefore there exists some $\|\bullet\|_b$ such that

$$\|\dot{x}^{k+1}\|_b \leqslant \alpha\|\dot{x}^k\|_b + (l_1K + l_2K)\|x(0)\| + K\|\hat{f}(0,0,u)\| \quad (1.2.14)$$

where $\alpha < 1$. This implies that

$$\|\dot{x}^{k+1}\|_b \leqslant \frac{1}{1-\alpha}[(l_1K + l_2K)\|x(0)\| + K\|\hat{f}(0,0,u)\|] + (\alpha)^k\|\dot{x}^0\|_b \quad (1.2.15)$$

for all k. Then, since $\|\dot{x}^0\|_b$ must be bounded given a finite $x(0)$, and $\|\dot{x}^{k+1}\|_b = \max_{[0,T]} e^{-bt}\|\dot{x}^{k+1}(t)\|$,

$$\|\dot{x}^{k+1}(t)\| < e^{bT}\left[\frac{1}{1-\alpha}[(l_1K + l_2K)\|x(0)\| + K\|\hat{f}(0,0,u)\|] + \|\dot{x}^0\|_b\right] = \tilde{M} \quad (1.2.16)$$

which proves the lemma. ■

In Lemma 1.2.4 it was proved that $\|\dot{x}^k(t)\|$ is bounded *a priori* by some $\tilde{M}$. This implies $x^k(t)$ is bounded on $[0,T]$. Using the Lipschitz continuity property of $\hat{f}$, a bound, $\tilde{N}$, can be derived for $\|\hat{f}(x^{k+1}(t), x^k(t), u)\|$. Applying these bounds to Eqn. 1.2.10 we get

$$\|\dot{x}^{k+1}(t)-\dot{x}^{j+1}(t)\| \leqslant \gamma\|\dot{x}^{k}(t)-\dot{x}^{j}(t)\| + \tag{1.2.17}$$

$$(l_1K+k_1\tilde{M}+k_3\tilde{N})\,\|x^{k+1}(t)-x^{j+1}(t)\| + (l_2K+\tilde{M}k_2+k_4\tilde{N})\,\|x^{k}(t)-x^{j}(t)\|$$

where $\gamma < 1$. Eqn. (1.2.17) is of the form to apply Lemma 1.2.3. As $x^{k+1}(0)-x^{j+1}(0) = 0$ for all k, j, Lemma 1.2.3 implies

$$\|\dot{x}^{k+1}-\dot{x}^{j+1}\|_b \leqslant \alpha\|\dot{x}^{k}-\dot{x}^{j}\|_b \tag{1.2.18}$$

for some norm on $C([0,T],\mathbb{R}^n)$ and for some $\alpha < 1$. As $C([0,T],\mathbb{R}^n)$ is complete in any one of the B norms, by the contraction mapping theorem $\dot{x}^k$ converges to some $\dot{x} \in C([0,T],\mathbb{R}^n)$ which is a fixed point of Eqn. (1.2.5). Any fixed point $\dot{x}$ of Eqn. (1.2.5) is a solution to Eqn. (1.1.3) if $x(0) = x_0$. $x^k(0) = x_0$ for all k, therefore $\dot{x}^k$ converges to the unique solution of Eqn. (1.1.3). The sequence $\{ x^k \}$ converges because integration from 0 to T, which maps $\dot{x}(t)$ to $x(t)$, is a bounded continuous function ■.

SECTION 1.3 - NONSTATIONARY WR ALGORITHMS

Algorithm 1.1.1 is stationary in the sense that the equations that define the iteration process do not change with the iterations. A straight-forward generalization is to allow these iteration equations to change, and to consider under what conditions the relaxation still converges [9]. There are two major reasons for studying nonstationary algorithms. The solution of the ordinary differential equations in the inner loop of Algorithm 1.1.1 cannot be obtained exactly. Instead numerical methods compute the solution with some error which is in general controlled, but which cannot be eliminated. However, the discrete approximation can be interpreted as the exact solution to a perturbed system. Since the approximation changes with the solutions, the perturbed system changes with each iteration. Hence, practical implementations of WR that must compute the solution to the iteration equations approximately can be interpreted as nonstationary methods.

The second reason for studying nonstationary methods is that they can be used to improve the computational efficiency of the basic WR algorithm. An approach would

be to improve the accuracy of the computation of the iteration equations as the relaxation approaches convergence. In this way, accurate solutions to the original system would still be obtained, but unnecessarily accurate computation of the early iteration waveforms, which are usually far from the final solution, is avoided.

In this section we show that nonstationary WR algorithms converge as a direct consequence of the contraction mapping property of the original WR algorithm. That is, given mild assumptions about the relationship between a general stationary contraction map and a nonstationary map, the nonstationary map will produce a sequence that will converge to within some tolerance. And if in the limit as $k \to \infty$ the nonstationary map approaches the stationary map, then the sequence generated by the nonstationary map will converge to the fixed point of the original map. In later sections we will lean on these results to guarantee the convergence of implementations of WR-based algorithms.

<u>Theorem 1.3.1:</u> Let Y be a Banach space and $F, F^k : Y \to Y$. Define $y^{k+1} = F(y^k)$ and $\tilde{y}^{k+1} = F^k(\tilde{y}^k)$. If F is a contraction mapping with contraction factor γ (See section 1.2), $\|F(y) - F^k(y)\| \leqslant \delta^k$ for all $y \in Y$, and $z \in Y$, is such that $z = F(z)$, then for any $\epsilon > 0$ there exists a $\delta < 1$ such that if $\delta^k < \delta$ for all k then $\lim_{k\to\infty}\|\tilde{y}^k - \tilde{y}^{k-1}\| < \epsilon$ and $\lim_{k\to\infty}\|z - \tilde{y}^k\| < \frac{\delta}{1-\gamma}$. Futhermore, if $\lim_{k\to\infty}\delta^k \to 0$ then $\lim_{k\to\infty}\|\tilde{y}^k - \tilde{y}^{k-1}\| \to 0$ and $\lim_{k\to\infty}\|z - \tilde{y}^k\| \to 0$.

<u>Proof of Theorem 1.3.1</u>

Taking the norm of the difference between the kth and $k+1^{st}$ iteration of the nonstationary algorithm we get:

$$\|\tilde{y}^{k+1} - \tilde{y}^k\| < \|F^{k+1}(\tilde{y}^k) - F^k(\tilde{y}^{k-1})\| \tag{1.3.1}$$

Given that $\|F^k(y) - F(y)\| < \delta^k$ for all $y \in Y$

$$\|\tilde{y}^{k+1} - \tilde{y}^k\| \leqslant \|F(\tilde{y}^k) - F(\tilde{y}^{k-1})\| + \delta^k + \delta^{k+1}. \tag{1.3.2}$$

Using the contraction property of F ,

$$\|\tilde{y}^{k+1}-\tilde{y}^{k}\| < \gamma\|\tilde{y}^{k}-\tilde{y}^{k-1}\| + \delta^{k} + \delta^{k+1}. \tag{1.3.3}$$

Unfolding the iteration equation into direct sum form,

$$\|\tilde{y}^{k+1} - \tilde{y}^{k}\|_{b} \leqslant \delta^{k+1} + \delta^{k} + \sum_{i=1}^{k} \gamma^{k-i}(\delta^{i} + \delta^{i-1}). \tag{1.3.4}$$

If $\delta^{k} < \delta$ for all k then from Eqn. (1.3.4)

$$\lim_{k\to\infty}\|\tilde{y}^{k+1} - \tilde{y}^{k}\| \leqslant 2\delta(1+\frac{1}{1-\gamma}). \tag{1.3.5}$$

As $\gamma<1$, $\lim_{k\to\infty}\|\tilde{y}^{k+1} - \tilde{y}^{k}\|$ can be made as small as desired by reducing δ , which proves the first part of Theorem 1.3.1.

Let y be the fixed point of F . The difference between the computed and the exact solution at the $k+1^{th}$ iteration is

$$\|\tilde{y}^{k+1}-y\| = \|F^{k}(\tilde{y}^{k})-F(y)\|. \tag{1.3.6}$$

Again using the contractive property of F and that $\|F(y) - F^{k}(y)\| \leqslant \delta^{k}$.

$$\|\tilde{y}^{k+1}-y\| = \gamma\|\tilde{y}^{k}-y\| + \delta^{k}. \tag{1.3.7}$$

Summing and taking the limit,

$$\lim_{k\to\infty}\|\tilde{y}^{k+1} - y\|_{b} \leqslant \frac{\delta}{1-\gamma}. \tag{1.3.8}$$

which completes the proof of the first statement of Theorem 1.3.1. The second statement of the theorem follows from almost identical arguments. ■

In Section 1.2 we proved the WR iteration was a contraction mapping in the appropriate norm $\|\bullet\|_{b}$ on $C([0,T],\mathbb{R}^{n})$ where B depended on the problem. To repeat the result from that section, it was shown that:

$$\|\dot{x}^{k+1}-\dot{x}^{j+1}\|_{b} \leqslant \alpha\|\dot{x}^{k}-\dot{x}^{j}\|_{b}$$

This WR convergence result and Theorem 1.3.1 imply that using any "reasonable" approximation method to solve the WR iteration equations will not affect the convergence *provided* the errors in the approximation are driven to zero. In addition, Theorem 1.3.1 indicates that it will be difficult to determine *a priori* how accurately the iteration equations must be solved to guarantee convergence to within a given

tolerance, because an estimate of the contraction factor of the WR algorithm is required.

As can be seen from Eqn. (1.3.8), the WR algorithm is a contraction mapping with respect to $\dot{x}(t)$ in a B norm. Theorem 1.3.1 then implies that the WR iteration equations must be solved accurately with respect to $\dot{x}(t)$ in this B norm if the iterations are to converge. There is a more cumbersome proof of the WR convergence theorem in which it is shown that the WR algorithm is a contraction in $x(t)$, but in a larger B norm than the one used in the proof of Theorem 1.2.1, and the size of this B is a function of the magnitude of the off-diagonal terms of $C(x,u)$. With such a result, Theorem 1.3.1 implies that it is only necessary to control errors in the computation of $x(t)$ to guarantee iteration convergence. However, convergence in a larger B norm is in some sense a weaker type of convergence. So, in the case where $C(x,u)$ has non-zero off-diagonal terms, it is expected that more rapid convergence would be achieved if the $x^k(t)$'s are computed in a way that also guarantees that the $\dot{x}^k(t)$'s are globally accurate.

SECTION 1.4 - WAVEFORM NEWTON METHODS

The WR algorithm is an extension to function spaces of the popular relaxation methods used to solve nonlinear algebraic problems. Another popular method for solving nonlinear algebraic problems is the Newton-Raphson method, and its function space extension also has practical applications. In this section we will derive the function-space Newton method applied to systems of the form of Eqn. (1.1.3) and prove that the method has *global* convergence properties, which is not true in general for the Newton-Raphson algorithms[22].

In order to derive a function-space extension to the Newton-Raphson algorithm, let $F(x)$ (from Eqn. (1.1.3)) be defined as

$$F(x) = C(x,u)\dot{x} - f(x,u) = 0 \qquad x(0) = x_0 \tag{1.4.1}$$

where $x:[0,T]\rightarrow \mathbb{R}^n, u:[0,T]\rightarrow \mathbb{R}^r$ and is piecewise continuous; $C: \mathbb{R}^n x \mathbb{R}^r \rightarrow \mathbb{R}^{nxn}$ is such that $C(x,u)^{-1}$ exists and is uniformly bounded with respect to x,u ; and $f: \mathbb{R}^n x \mathbb{R}^r \rightarrow \mathbb{R}^n$ is globally Lipschitz continuous with respect to x for all u . Applying the Newton-Raphson algorithm to find an x such that $F(x)=0$ given some initial guess x^0 we get

$$x^{k+1} = x^k - J_F^{-1}(x^k)F(x^k) \tag{1.4.2}$$

where $J_F(x)$ is the Frechet derivative of $F(x)$ with respect to x . Note that in this case $J_F(x)$ is a matrix-valued function on $[0,T]$. That is, $J_F(x)$ is a matrix of waveforms.

Using the definition of the Frechet derivative, we can compute $J_F(x)$,

$$\lim_{h\rightarrow 0} (1/\|h\|)\|F(x+h) - F(x) - J_F(x)(h)\| \rightarrow 0. \tag{1.4.3}$$

Evaluating this limit for the $F(x)$ given in Eqn. (1.4.1) we get

$$F(x+h) - F(x) = C(x+h,u)(\dot{x}+\dot{h}) - C(x,u)\dot{x} - f(x+h,u) + f(x,u)$$

and approximating to order $\|h\|^2$

$$F(x+h)-F(x) = C(x,u)\dot{h} + \frac{\partial C(x,u)}{\partial x}(h)\dot{x} - \frac{\partial f(x,u)}{\partial x}h + O(\|h\|^2) \tag{1.4.4}$$

As Eqn. (1.4.3) applies only in the limit as $h\rightarrow 0$, Eqn. (1.4.4) implies

$$J_F(x)h = C(x,u)\dot{h} + \frac{\partial C(x,u)}{\partial x}h\dot{x} - \frac{\partial f(x,u)}{\partial x}h \tag{1.4.5}$$

Substituting the computed derivative into Eqn. (1.4.2) and rearranging we get

$$C(x^k,u)\dot{x}^{k+1} + \frac{\partial C(x^k,u)}{\partial x}(x^{k+1}-x^k)\dot{x}^k = \tag{1.4.6}$$

$$f(x^k,u) + \frac{\partial f(x^k,u)}{\partial x}(x^{k+1}-x^k)$$

$$x^{k+1}(0) = x_0$$

We will refer to Eqn. (1.4.6) as the Waveform-Newton(WN) algorithm for solving Eqn (1.1.3). It is, however, just the function-space extension of the classical Newton-Raphson algorithm.

Newton-Raphson algorithms converge quadratically when the iterated value is close to the correct solution, but they do not in general have global convergence properties. However, the WN algorithm represented by Eqn. (1.4.6) does converge for any initial guess, given mild assumptions on the behavior of $\frac{\partial C(x,u)}{\partial x}$ as in the following theorem.

Theorem 1.4.1: For any system of the form of Eqn. (1.1.3) in which $\frac{\partial C(x,u)}{\partial x}$ is Lipschitz continuous with respect to x for all u and f is continuously differentiable, the sequence $\{ x^k \}$ generated by the WN algorithm converges uniformly to the solution of Eqn. (1.4.1).

Proof of Theorem 4.1

For this proof of the convergence of the Waveform-Newton method we will assume that $C(x,u)$ is the identity, as the proof for the general case is much more involved, and does not provide much further insight into the nature of the convergence. For the case $C(x,u) = I$ Eqn. (1.4.6) can be simplified to

$$\dot{x}^{k+1} = f(x^k,u) + \frac{\partial f(x^k,u)}{\partial x}(x^{k+1}-x^k). \tag{1.4.7}$$

Taking the difference between Eqn. (1.4.7) at iteration $k+1$ and the exact solution and substituting $(x^{k+1}-x) + (x-x^k)$ for $x^{k+1}-x^k$ yields

$$\dot{x}^{k+1}-\dot{x} = f(x^k,u)-f(x,u) + \frac{\partial f(x^k,u)}{\partial x}[(x^{k+1}-x) + (x-x^k)]. \tag{1.4.8}$$

As f is continuously differentiable on $[0,T]$ and Lipschitz continuous, $\frac{\partial f(x,u)}{\partial x}$ is bounded by the Lipschitz constant l_1. With this bound,

$$\|\dot{x}^{k+1}-\dot{x}\| < l_1\|x^{k+1}-x\| + l_1\|x^{k+1}-x\| + l_1\|x^k-x\|. \tag{1.4.9}$$

Lemma 1.2.3 can be applied to Eqn. (1.4.9) (with $\gamma = 0$). Therefore there exists some $b < \infty$ and $\gamma < 1$ such that

$$\|\dot{x}^{k+1}-\dot{x}\|_b \leqslant \alpha\|\dot{x}^k-\dot{x}\|_b. \tag{1.4.10}$$

Therefore $\{ \dot{x}^k \}$ converges to $\dot{x}$, the fixed point of Eqn. (1.4.1). Given $x^k(0) = x_0$

for all k, $\{ x^k \}$ converges to the solution of Eqn. (1.4.1) on any bounded interval. ■

SECTION 1.5 - DISCRETIZED WR ALGORITHMS

To compute the iteration waveforms for the WR algorithm it is usually necessary to solve systems of nonlinear ordinary differential equations. The most popular techniques for solving these systems are the multistep integration formulas (such as the Backward Difference or Trapezoidal formulas[1]). These methods approximate the original differential equation by a sequence of algebraic equations corresponding to a collection of discrete points in time. The error in this discretization approximation is a function of the timesteps, which are usually chosen small enough so that the waveforms are computed to some *a priori* accuracy.

The convergence theorem presented in Section 1.2 is not immediately applicable to the convergence of this **discretized** WR algorithm because the differential equations that describe the decomposed systems are not solved exactly. However, one can view the discretized WR algorithm as a nonstationary method. The theorems presented in Section 1.3 can then be applied to guarantee WR convergence to the solution of the given system of ODE's when the global discretization error, a function of the timesteps, is driven to zero with the WR iterations.

If the global discretization error is not driven to zero, one may expect that the WR algorithm will still converge to an approximate solution of the given system of ODE's. In this section we show that unless the timesteps used in the numerical method are kept below some problem-dependent bound, the WR algorithm may not converge. We will start by analyzing a simple example that demonstrates a possible breakdown of the WR method under discretizations. Subsequently we will prove that the discretized WR algorithm converges if the timesteps used are "small enough". Finally, we will end this section by comparing explicit and implicit integration methods for WR.

Consider the two node inverter circuit in Fig. 1. The current equations at each node can be written by inspection, and are:

$$C\dot{x}_1+g_1x_1 + g_2(x_1-x_2) = 0 \tag{1.5.1}$$

$$C\dot{x}_2+g_2(x_2-x_1) + i_{m1}(x_1,x_2) + i_{m2}(x_1) = 0$$

$$x_1(0) = x_2(0) = 0.$$

In order to generate a simple linear example, i_{m1}, i_{m2} were linearized about the point where the input and output voltages were equal to half the supply voltage. Time is normalized to seconds to get the following 2x2 example:

$$\dot{x}_1 = -x_1 + 0.1x_2 \tag{1.5.2}$$

$$\dot{x}_2 = -\lambda x_1 + -x_2$$

$$x_1(0) = x_2(0) = 0.$$

Note that the initial conditions given for the above example identify a stable equilibrium point.

The Gauss-Seidel WR iteration equations for the linear system example are:

$$\dot{x}_1^{k+1} = -x_1^{k+1} + 0.1x_2^k \tag{1.5.3}$$

$$\dot{x}_2^{k+1} = -\lambda x_1^{k+1} - x_2^{k+1}$$

$$x_1^{k+1}(0) = x_1^k(0) = x_2^{k+1}(0) = x_2^k(0) = 0.$$

Applying the Implicit-Euler numerical integration method with a fixed timestep h, ($\dot{x}(nh) = \frac{1}{h}[x(nh)-x((n-1)h)]$) to solve the decomposed equations yields the following recursion equation for $x_2^k(n)$:

$$x_2^{k+1}(n) = \frac{1}{1+h}\, x_2^{k+1}(n-1) \tag{1.5.4}$$

$$- \frac{\lambda h}{(1+h)^2}\left[\frac{x_1(0)}{(1+h)^n} + 0.1h\sum_{j=1}^{n}(1+h)^{j-n}x_2^k(j)\right].$$

As an example, let λ=200, h=0.5 and as an initial guess use $x_2^0(nh) = nh$, which is far from the exact solution $x_2^0(nh) = 0$. The computed sequences for the initial guess and first, second and third iterations of Eqn. (1.5.4) are presented in Table 1.

Table 1 - Values for x_2 for Several Implicit-Euler Computed WR Iterations

STEP	TIME	INITIAL	ITER #1	ITER #2	ITER #3
0	0	0	0	0	0
1	0.5	0.5	-1.111	2.469	-5.487
2	1.0	1.0	-3.704	11.52	-32.92
3	1.5	1.5	-7.778	31.55	-111.6
4	2.0	2.0	-13.17	66.21	-281.3
5	2.5	2.5	-19.66	117.9	-587.5
6	3.0	3.0	-27.02	187.9	-1075
7	3.5	3.5	-35.07	276.0	-1786
8	4.0	4.0	-43.64	381.5	-2751
9	4.5	4.5	-52.60	502.9	-3992
10	5.0	5.0	-61.85	638.4	-5519

As the Table 1 indicates, the WR algorithm diverges for this example. In fact, Eqn. (1.5.4) indicates that the WR algorithm will converge only if

$$\frac{h}{(1+h)} < \frac{1}{\sqrt{0.1\lambda}} \tag{1.5.5}$$

To understand this nonconvergence phenomenon consider the Gauss-Seidel WR algorithm applied to Eqn. (1.1.3) with $C(x,u) = C$. The WR iteration equation is (identical to Eqn. 1.2.9):

$$\dot{x}^{k+1} = (L + D)^{-1}U\dot{x}^{k} + \hat{f}(x^{k+1}, x^{k}, u). \tag{1.2.9}$$

Applying Implicit-Euler yields:

$$x^{k+1}(n+1) - x^{k+1}(n) = (L + D)^{-1}U(x^{k}(n+1) - x^{k}(n)) + h\,\hat{f}(x^{k+1}(n+1), x^{k}(n+1), u). \tag{1.5.6}$$

In the limit as $h \to \infty$, Eqn. (1.5.6) becomes equivalent to solving $\hat{f}(x^{k+1}(n+1), x^{k}(n+1), u) = 0$. Since little is assumed about f other than Lipschitz continuity, it is unlikely that this problem can be solved, in general, with a simple Gauss-Seidel relaxation. However, in the limit as the timestep becomes small, Eqn. (1.5.6) becomes

$$x^{k+1}(n+1) - x^{k+1}(n) = (L + D)^{-1}U(x^{k}(n+1) - x^{k}(n)).$$

and from the lemma in Section 1.2, the norm of $\|(L + D)^{-1}U\| < 1$ so the relaxation

is certain to converge. The timestep h can be viewed as a parameterization of this algebraic problem. As the timestep decreases, the problem is continuously deformed from one that may not be solvable by relaxation to one that is guaranteed to be solvable by relaxation. We formalize this observation in the following theorem:

Theorem 1.5.1: If, in addition to the assumptions of Theorem 1.2.1, the WR iteration equations are solved using a stable and consistent multistep method with a fixed timestep h, for a finite number of points, then there exists an $h' > 0$ such that the sequences $\{ x^k(n) \}$ generated by the Gauss-Seidel or Gauss-Jacobi discretized WR algorithm will converge for all $0 < h < h'$.

The proof of this theorem can be found in [26].

Now consider solving Eqn. (1.5.3) using the computationally simpler Explicit-Euler integration formula ($\dot{x}(nh) = \frac{1}{h}[x((n+1)h)-x(nh)]$). The recursion equation for the $x_2^k(n)$'s is:

$$x_2^{k+1}(n+1) = (1-h)x_2^{k+1}(n) - \qquad (1.5.7)$$

$$0.1\lambda h^2[\ [(1-h)^n x_1^k(0)] + \sum_{j=1}^{n-1}(1-h)^{n-1-j} x_2^k(j)\ ]$$

The computed sequences $\{x_2^{k+1}\}$'s for the initial guess and first, second and third iterations of Eqn. (1.5.7) are given in Table 2, for the case of $\lambda=200$, $h=0.5$ and $x_2^0(nh) = nh$.

As the table indicates, the Explicit-Euler discretized WR algorithm converges for this example. In general, if explicit integration methods are used in the WR algorithm, the iterations will converge for any fixed timestep.

Theorem 1.5.2: If, in addition to the assumptions of Theorem 1.2.1, the WR iteration equations are solved using an explicit multistep method with a fixed timestep h, for a finite number of timesteps, then the sequences $\{ x^k(n) \}$ generated by the Gauss-Seidel or Gauss-Jacobi discretized WR algorithm will converge for all $h > 0$.

Table 2 - Values for x_2 for several Explicit-Euler computed WR iterations					
STEP	TIME	INITIAL	ITER #1	ITER #2	ITER #3
0	0	0	0	0	0
1	0.5	0.5	0	0	0
2	1.0	1.0	0	0	0
3	1.5	1.5	-0.625	0	0
4	2.0	2.0	-1.875	0	0
5	2.5	2.5	-3.594	0.7813	0
6	3.0	3.0	-5.625	3.125	0
7	3.5	3.5	-7.852	7.422	-0.977
8	4.0	4.0	-10.19	13.67	-4.883
9	4.5	4.5	-12.61	21.63	-13.92
10	5.0	5.0	-15.06	30.96	-29.79

<u>Proof of Theorem 1.5.2</u>

The proof of this theorem follows from a simple inductive argument [21]. Let $x(n)$ be the exact solution to the system discretized using an explicit method. Assume $x^k(m) = x(m)$ for all $m < n$. Since the integration method is explicit, $x^k(n)$ and $x(n)$ are the same function of u and $x^k(m)$, $m < n$. Therefore, $x^k(n) = x(n)$. $x^k(0) = x(0)$ for all k by assumption, which completes the proof. Note that this proof guarantees that the discretized WR algorithm converges precisely to $x(n)$, the exact solution to the original **discretized** system, in n iterations. ■

It should be noted that the above proof does not show the explicit discretized WR algorithm is a contraction, nor does it show convergence on a fixed time interval independent of h. A more general proof can be found in [20].

Insuring relaxation convergence puts no constraints on the timesteps for the Explicit-Euler method, or for explicit methods in general. But since these methods have small regions of absolute stability, the timestep may be limited not by accuracy considerations but to insure stability. For example, consider the differential equation of Eqn. (1.5.3), but with a perturbed initial condition, $x_1^k(0) = x_2^k(0) = 0.1$. The exact solution will decay asymptotically to zero, but the numerical solution produced by the Explicit-Euler algorithm will decay asymptotically to zero only if:

$$h < \frac{2}{1+0.1\lambda}. \tag{1.5.8}$$

Of course, if it is really reasonable to solve a system using an explicit integration method with fixed timesteps, WR is a not a good algorithm to use. As the convergence proof indicates, the WR algorithm will converge for at least one additional time step, and probably no more, with each relaxation iteration. For that reason, it is inefficient to compute more than one additional time step with each WR iteration. Given that, it follows that there is no reason to recompute the old timepoints because the relaxation will have converged for sure. The WR algorithm is then reduced to an explicit integration algorithm applied to the entire system directly.

Still, it is interesting to examine the case when the timestep necessary to insure convergence of the Implicit-Euler discretized WR algorithm is as small as the timestep necessary for stability of the Explicit-Euler. Because, if implicit methods do not allow use of much larger timesteps, the extra computations required to use them is not worthwhile. From Eqns. (1.5.5) and (1.5.8), if $\lambda = 10$ then the Implicit-Euler timesteps are unconstrained and the Explicit-Euler timestep must be less than 1.0. If $\lambda = 100$ then the Implicit-Euler timesteps must be less than 0.37 , and the Explicit-Euler timesteps must be less than 0.18. In addition, Implicit-Euler will continue to allow larger timesteps than Explicit-Euler for very large λ , because its timestep constraint decreases as $\frac{1}{\sqrt{\lambda}}$, where as the Explicit-Euler timestep constraint decreases as $\frac{1}{\lambda}$.

One can infer from the above example that the WR algorithm allows the use of larger timesteps than a direct explicit method in most cases, but constrains the timesteps more than a direct implicit method. The difference between the direct implicit method timestep constraint and that for the WR algorithm is smallest if the system to be solved is very loosely coupled. Digital integrated circuits, for which the WR

algorithm was originally developed, are not always loosely coupled. The coupling can be quite strong, but is usually so only for short intervals. The WR algorithm is efficient for these problems because small timesteps are required (to insure WR convergence) only during those intervals when the coupling is strong, and, because implicit integration is used, the timestep can safely be made much larger for the rest of the interval[19].

Most of the above analysis does not extend readily to the case where different timesteps are used for different nodes of the system. Examining the multiple timestep case is in general a very difficult problem in numerical analysis, even for standard methods, and nothing has been published examining this case. Since the major advantage of WR is that only those variables in the system that are changing rapidly use small timesteps, this is an important missing piece of the theory about WR methods.

SECTION 2 - IMPLEMENTATION TECHNIQUES FOR WR METHODS

SECTION 2.1 - PARTITIONING METHODS

In Algorithm 1.1.1, the system equations are solved as single differential equations in one unknown, and these solutions are iterated until convergence. If this kind of node-by-node decomposition strategy is used for systems with even just a few tightly coupled nodes, the WR algorithm will converge very slowly. As an example, consider the three node circuit in Fig. 2a, a two inverter chain separated by a resistor-capacitor network. In this case, the resistor-capacitor network is intended to model wiring delays, so the resistor has a large conductance compared to the other conductances in the circuit. The current equations for the system can be written down by inspection and are:

$$C\dot{x}_1 + i_{m1}(x_1, vdd) + i_{m2}(x_1, u) + g(x_1 - x_2) = 0$$

$$C\ \dot{x}_2\ g\ (x_2 - x_1) = 0$$

$$C\ \dot{x}_3\ i_{m3}(x_3, x_2) + i_{m4}(x_3, vdd) = 0$$

Linearizing and normalizing time (so that the simulation interval $[0,T]$ is converted to $[0,1]$) yields a 3x3 linear equation:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} -10 & 9.5 & 0 \\ 9.5 & -9.5 & 0 \\ 0 & -1 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 5 \\ 0 \\ 0 \end{bmatrix}$$

$$x_1(0)=x_2(0)=0 \quad x_3(0)=5$$

Algorithm 1.1.1 was used to solve the original nonlinear system. The input $u(t)$, the exact solution for x_2, and the first, fifth and tenth iteration waveforms generated by the WR algorithm for x_2 are plotted in Fig. 2b. As the plot indicates, the iteration waveforms for this example are converging very slowly. The reason for this slow can be seen by examining the linearized system. It is clear x_1 and x_2 are tightly coupled by the small resistor modeling the wiring delay.

If Algorithm 1.1.1 is modified, so that x_1 and x_2 are lumped together and solved directly, we get the following iteration equations:

$$\begin{bmatrix} \dot{x}_1^{k+1} \\ \dot{x}_2^{k+1} \end{bmatrix} = \begin{bmatrix} -10 & 9.5 & 0 \\ 9.5 & -9.5 & 0 \end{bmatrix} \begin{bmatrix} x_1^{k+1} \\ x_2^{k+1} \\ x_3^{k} \end{bmatrix} + \begin{bmatrix} 5 \\ 0 \end{bmatrix}$$

$$\dot{x}_3^{k+1} = -x_2^{k+1} - x_3^{k+1}$$

The modified WR algorithm now converges in one iteration, because x_3 only depends on the "block" of x_1 and x_2, and that block is independent of x_3.

As the example above shows, lumping together tightly coupled nodes and solving them directly can greatly improve the efficiency of the WR algorithm. For this reason, the first step in almost every WR-based program is to *partition* the system, to scan all the nodes in the system and determine which should be lumped together and solved directly. Partitioning "well" is difficult for several reasons. If too many nodes are lumped together, the advantages of using relaxation will be lost, but if any tightly coupled nodes are not lumped together then the WR algorithm will converge very slowly. And since the aim of WR is to perform the simulation rapidly, it is important

that the partitioning step not be computationally burdensome.

Three approaches have been applied to solve this partitioning problem, and all have been used successfully in programs for solving large systems. The first approach is to require the user to partition the system [10] . This technique is reasonable for the simulation of large digital integrated circuits because usually the large circuit has already been broken up into small, fairly independent pieces to make the design easier to understand and manage. However, what is a sensible partitioning from a design point of view may not be a good partitioning for the WR algorithm. For this reason programs that require the user to partition the system sometimes perform a "sanity check" on the partitioning [11] . A warning is issued if there are tightly coupled nodes that have not been lumped together.

A second approach to partitioning, also tailored to digital integrated circuits, is the functional extraction method[12]. In this method the equations that describe the system are carefully examined to try to find functional blocks (i.e. a *nand* gate or a *flip-flop*). It is then assumed that nodes of the system that are members of the same functional block are tightly coupled, and are therefore grouped together. This type of partitioning is difficult to perform, since the algorithm must recognize broad classes of functional blocks, or nonstandard blocks may not be treated properly. However, the functional extraction method can produce very good partitions because the relative importance of the coupling of the nodes can be accurately estimated.

The most general, and perhaps the most obvious, approach to the partitioning problem is the "diagonal dominant loop" method [13] [12] . In this method tightly coupled nodes are determined by examining 2x2 submatrices of the Jacobian of $f(x,u)$ and $C(x,u)$. If the magnitude of the product of the diagonal terms is not greater than the product of the off-diagonal terms by some factor α, (a good choice will depend on the application), then the two nodes corresponding to the submatrix are

lumped together. The precise algorithm is as follows:

<u>Algorithm 1.2.1 - Diagonal Dominant Loop Partitioning</u>

Step 1: Compute matrices L^f, L^C defined by

for all (i, j **in** N) {

$$L_{ij}^f \ (i \neq j) = \max_{x,u} \left| \frac{\partial f_i(x,u)}{\partial x_j} \right|$$

$$L_{ii}^f = \min_{x,u} \left| \frac{\partial f_i(x,u)}{\partial x_i} \right|$$

$$L_{ij}^C \ (i \neq j) = \max_{x,u} |C_{ij}(x,u)|$$

$$L_{ii}^C = \min_{x,u} |C_{ii}(x,u)|$$

}

Step 2: Partition the system.

for all (i, j **in** N) {

if ($L_{ii}^C L_{jj}^C \geqslant \alpha L_{ij}^C L_{ji}^C$ *or* $L_{ii}^f L_{jj}^f \geqslant \alpha L_{ij}^f L_{ji}^f$) {

x_i is lumped with x_j

}

}

■

The diagonal dominant loop method has the advantage of simplicity and generality, but it is often too conservative in practice. Unnecessarily large subsystems can be generated because only the worst-case coupling is considered when lumping nodes together. There are also cases for which the method is not conservative enough. A poor partitioning will be generated for systems that include sets of nodes that are extremely tightly coupled to each other and are also tightly coupled to other nodes in the system. (A somewhat complicated modification to this algorithm eliminates this difficulty[28]). The functional extraction method is much less general, and if it is to capture a wide variety of functional blocks, can become a very complicated algorithm. However, the functional extraction methods better estimate the effective coupling between nodes, and therefore are likely to generate smaller subsystems.

In the case where a functional extraction method exists, but is too complicated to apply to a large system directly, then a good mixed approach is to use the two methods sequentially. First partition the system by applying the diagonal dominant loop method. Then apply the functional extraction method only to any overly large

subsystems. In this way a reasonable partition can be generated quickly.

SECTION 2.2 - WINDOWING

The convergence theorem presented in Section 1.2 guarantees that the WR algorithm is a contraction mapping in an exponentially weighted norm. In this section, we will demonstrate by example the practical implications of this choice of norm. We will then examine how to reduce the number of iterations required to achieve convergence by breaking the simulation interval into small pieces, or "windows". First we will prove WR convergence in an unweighted norm for short intervals. As this proof must take into account worst-case behavior, the estimate of the interval the proof provides is too short to be practical. This will lead us to the conclusion that an adaptive approach to choosing the windows will be more useful, and is a *safe* alternative because the basic convergence theorem guarantees that regardless of the interval chosen, the WR algorithm converges.

Consider the following nonlinear ordinary differential equation in $x_1(t), x_2(t) \in \mathbb{R}$ with input $u \in \mathbb{R}$ that approximately describes the cross-coupled *nor* logic gate in Fig. 3a (the approximate equations represent a normalization that converts the simulation interval $[0,T]$ to $[0,1]$).

$$\dot{x}_1 = (5-x_1) - x_1(x_2)^2 - 5x_1u \tag{2.2.1}$$

$$\dot{x}_2 = (5-x_2) - x_2(x_1)^2$$

$$x_1(0) = 5.0 \qquad x_2(0) = 0.0$$

The Gauss-Seidel WR Algorithm given in Section 1.2 was used to solve the for the behavior of the cross-coupled *nor* gate circuit approximated by the above small system of equations. In Fig. 3b plots of the input $u(t)$, the exact solution for $x_1(t)$, and the relaxation iteration waveforms for $x_1(t)$ for the 5th, 10th and 20th iterations are shown. The plots demonstrate a property typical of the WR algorithm when applied to systems with strong coupling: the difference between the iteration

waveforms and correct solution is not reduced at every time point in the waveform. Instead, each iteration lengthens the interval of time, starting from zero, for which the waveform is close to the exact solution.

As an example of "better" convergence, consider the following differential equation in x_1, x_2, x_3 with input u that approximately describs the shift register in Fig. 4a (here the simulation interval $[0,T]$ has been normalized to $[0,1]$)

$$\dot{x}_1 = (5.0 - x_1) - x_1(u)^2 - (x_1 - x_2) \tag{2.2.2}$$

$$\dot{x}_2 = (x_1 - x_2)$$

$$\dot{x}_3 = (5.0 - x_3) - x_3(x_2)^2$$

$$x(0) = 0.$$

The Gauss-Seidel WR Algorithm given in Section 1.2 was used to solve the original system approximated by the above system of equations. The input $u(t)$, the exact solution for $x_1(t)$, and the waveforms for $x_1(t)$ computed from the first, second, and third iterations of the WR algorithm are plotted in Fig. 4b. As the plots for this example show, the difference between the iteration waveforms and the correct solution is reduced throughout the entire waveform.

Perhaps surprisingly, the behavior of the first example is consistent with the WR convergence theorem, even though that theorem states that the iterations converge uniformly. This is because it was proved that the WR method is a contraction map in the following nonuniform norm on $C([0,T],\mathbb{R}^n)$:

$$\max_{[0,T]} e^{-bt} \|f(t)\|$$

where $b > 0$, $f(t) \in \mathbb{R}^n$, and $\| \bullet \|$ is a norm on $\mathbb{R}^n$. Note that $\|f(t)\|$ can increase as e^{bt} without increasing the value of this function space norm. If $f(t)$ grows slowly, or is bounded, it is possible to reduce the function space norm by reducing $\|f(t)\|$ only on some small interval in $[0,T]$, though it will be necessary to increase this interval to decrease further the function space norm. The waveforms in the more

slowly converging example above, converge in just this way; the function space norm is decreased after every iteration of the WR algorithm because significant errors are reduced over larger and larger intervals of time. The examples above lead to the following definition:

Definition 2.2.1: A differential system of the form given in Eqn. (1.1.3) is said to have the *strict WR contractivity property* on $[0,T]$, if the WR algorithm applied to the system is a contraction map in a uniform norm on $[0,T]$, i.e.

$$\max_{[0,T]} \|x^{k+1}(t) - x^k(t)\| \quad < \quad \max_{[0,T]} \|x^k(t) - x^{k-1}(t)\| \qquad (2.2.3)$$

where $x(t) \in \mathbb{R}^n$ on $t \in [0,T]$ is the solution to Eqn. (1.1.3); $x^k(t) \in \mathbb{R}^n$ on $t \in [0,T]$ is the kth iterate of Algorithm 1.1.1; and $\| \bullet \|$ is any norm on $\mathbb{R}^n$. If the WR algorithm applied to the system is a contraction in a uniform norm on $[0,T]$ for any $T > 0$ then we say that the system has the strict WR contractivity property on $[0,\infty)$. ■

For a system of equations to have the strict WR contractivity property on $[0,\infty)$ it must be more than just loosely coupled. In addition, the decomposed equations solved at each iteration of the waveform relaxation must be well-damped, so that errors due to the decomposition die off in time, instead of accumulating or growing. As the crossed *nand* gate example indicates, many systems of interest do not have the strict WR contractivity property on $[0,T)$ for all $T < \infty$. However, we will prove that any system that satisfies the WR convergence theorem will also have the strict WR contractivity property on some nonzero interval.

Theorem 2.2.1: For any system of the form of Eqn. (1.1.3) which satisfies the assumptions of the WR convergence theorem (Theorem 1.2.1) there exists a $T > 0$ such that the system has the strict WR contractivity property on $[0,T]$.

Proof of Theorem 2.2.1

We will prove the theorem only for the Gauss-Seidel WR algorithm but, as before, the theorem holds for the Gauss-Jacobi case. Starting with Eqn. (1.2.8) and substituting x^k for x^j,

$$\dot{x}^{k+1}(t)-\dot{x}^k(t) = \tag{1.2.8}$$

$$(L_{k+1}(t)+D_{k+1}(t))^{-1}U_{k+1}(t)\dot{x}^k(t) - (L_k(t)+D_k(t))^{-1}U_k(t)\dot{x}^{k-1}(t) +$$

$$(L_{k+1}+D_{k+1})^{-1}\hat{f}(x^{k+1}, x^k, u) - (L_k+D_k)^{-1}\hat{f}(x^k, x^{k-1}, u)$$

To simplify the notation, let $A_k(t), B_k(t) \in \mathbb{R}^{nxn}$ be defined by $A_k(t) = (L_k(t)+D_k(t))^{-1}U_k(t)$, $B_k(t) = (L_k(t)+D_k(t))^{-1}$. It is be important to keep in mind that $(L_k(t)+D_k(t))^{-1}U_k(t)$, and $(L_k(t)+D_k(t))^{-1}$ are functions of x^k, and by definition, so are $A_k(t)$ and $B_k(t)$. Expanding the above equation and integrating,

$$\int_0^t (\dot{x}^{k+1}(\tau)-\dot{x}^k(\tau))d\tau = \int_0^t A_{k+1}(\tau)(\dot{x}^k(\tau) - \dot{x}^{k-1}(\tau))d\tau + \tag{2.2.4}$$

$$\int_0^t [A_{k+1}(\tau)-A_k(\tau)]\,\dot{x}^{k-1}(\tau)\,d\tau +$$

$$\int_0^t B_{k+1}(\tau)[\hat{f}(x^{k+1}(\tau), x^k(\tau), u(\tau))-\hat{f}(x^k(\tau), x^{k-1}(\tau), u(\tau)]d\tau +$$

$$\int_0^t [B_{k+1}(\tau)-B_k(\tau)]\,\hat{f}(x^k(\tau), x^{k-1}(\tau), u(\tau))d\tau$$

Integrating by parts and using the fact that $x^k(0) - x^{k-1}(0) = 0$,

$$x^{k+1}(t)-x^k(t) = A_{k+1}(t)\,[x^k(t)-x^{k-1}(t)] - \tag{2.2.5}$$

$$\int_0^t \frac{d}{d\tau}A_{k+1}(\tau)[x^k(\tau)-x^{k-1}(\tau)]d\tau + \int_0^t [A_{k+1}(\tau) - A_k(\tau)]\,\dot{x}^{k-1}\,d\tau +$$

$$\int_0^t B_{k+1}(\tau)(\hat{f}(x^{k+1}(\tau), x^k(\tau), u(\tau))-\hat{f}(x^k(\tau), x^{k-1}(\tau), u(\tau))d\tau +$$

$$\int_0^t [B_{k+1}(\tau)-B_k(\tau)]\,\hat{f}(x^k(\tau), x^{k-1}(\tau), u(\tau))d\tau$$

Taking norms, and using the Lipschitz continuities of f, $A_k(t)$, and $B_k(t)$, and the uniform boundedness of $B_k(t)$ in x (see Thm 1.2.1):

$$\|x^{k+1}(t)-x^k(t)\| - \int_0^t (l_1K+k_1\tilde{M}+k_3\tilde{N})\|x^{k+1}(\tau)-x^k(\tau)\| \leqslant \tag{2.2.6}$$

$$\gamma\|x^k(t)-x^{k-1}(t)\| + \int_0^t (l_2K+k_1\tilde{M}+2k_2\tilde{M}+k_4\tilde{N})\|x^k(\tau)-x^{k-1}(\tau)\|d\tau$$

where l_1, l_2 are the Lipschitz constants of $\hat{f}$ with respect to x^{k+1} and x^k respectively; k_1, k_2, k_3, k_4 are the Lipschitz constants for $A_{k+1}(t)$, $B_{k+1}(t)$ with respect to their x_{k+1} and x_k arguments respectively; $\gamma = \max_{[x,k]}[(L_k+D_k)U_k] < 1$; and $\tilde{M}$ and $\tilde{N}$ are the *a priori* bounds on $\dot{x}^k$ and $\hat{f}$ found in the proof of Theorem 1.2.1. Note that $\frac{d}{d\tau}A_{k+1}(\tau) = \frac{d}{dx^{k+1}}A_{k+1}\dot{x}^{k+1} + \frac{d}{dx^k}A_{k+1}\dot{x}^k < k_1\tilde{M}+k_2\tilde{M}$. Moving the max (over t) norms outside the integrals and integrating yields

$$\max_{[0,T]}\|x^{k+1}(t)-x^k(t)\| \leqslant \tag{2.2.7}$$

$$\frac{\gamma+T\ (Kl_2+k_1\tilde{M}+2k_2\tilde{M}+k_4\tilde{N})}{1-T(Kl_1+k_1\tilde{M}+k_3\tilde{N})}\max_{[0,T]}\|(x^k(t)-x^{k-1}(t))\|.$$

Since $\gamma < 1$, a $T'>0$ exists such that $\frac{\gamma+T'(Kl_2+k_1\tilde{M}+2k_2\tilde{M}+k_4\tilde{N})}{1-T'(l_1K+k_1\tilde{M}+k_3\tilde{N})} = \alpha < 1$. With this T', Eqn. (2.2.17) becomes

$$\max_{[0,T']}\|x^{k+1}-x^k\| \leqslant \alpha\max_{[0,T']}\|x^k-x^{k-1}\| \tag{2.2.8}$$

for $\alpha < 1$, which proves the theorem. ■

Theorem 2.2.1 guarantees that the WR algorithm will be a contraction mapping in a uniform norm for any system, provided the interval of time over which the waveforms are computed is made small enough. This suggest that the interval of simulation $[0,T]$ should be broken up into *windows*, $[0,T_1], [T_1,T_2], \cdots, [T_{n-1},T_n]$ where the size of each window is small enough so that the WR algorithm contracts uniformly throughout the entire window. The smaller the window is made, the faster the convergence. However, as the window size becomes smaller, the advantages of the waveform relaxation are lost. Scheduling overhead increases when the windows become smaller, since each subsystem must be processed at each iteration in every win-

dow. If the windows are made very small, timesteps chosen to calculate the waveforms will be limited by the window size rather than by the local truncation error, and unnecessary calculations will be performed.

The lower bound for the region over which WR contracts uniformly given in Theorem 2.2.1 is too conservative in most cases to be of direct practical use. As mentioned above, in order for the WR algorithm to be efficient it is important to pick the largest windows over which the iterations actually contract uniformly, but the theorem only provides a worst-case estimate. Since it is difficult to determine *a priori* a reasonable window size to use for a given nonlinear problem, window sizes are usually determined dynamically, by monitoring the computed iterations[13]. Since Theorem 1.2.1 guarantees the convergence of WR over *any* finite interval, a dynamic scheme does not have to pick the window sizes very accurately. The only cost of a bad choice of window is loss of efficiency, the relaxation will *still* converge.

SECTION 2.3 - RELAXATION-NEWTON METHODS

In Section 1.3 we discussed a general class of methods for improving the computational efficiency of WR algorithms. The approach taken in these methods was to approximately solve the iteration equations when the computed x^k 's were far from convergence. We proved that if the WR iteration equations are only solved approximately, but the accuracy of the approximation is improved with each iteration, then these methods have convergence properties similar to the canonical WR algorithm. The practical question is then what approximation should be used initially, and by how much should the accuracy be improved with each iteration. In this section we will present a modified WR algorithm that automatically adjusts the accuracy of the computation to how close the iterations are to convergence. The method is an extension to function spaces of relaxation-Newton algorithms used for solving nonlinear algebraic systems[14],[15],[16]. In these algorithms the nonlinear iteration equations are not

solved exactly, but are solved approximately, by performing one step of a Newton method. Since the accuracy of the one Newton step improves as the x^k 's approach the exact solution, the iteration equations will be solved more accurately with each iteration *if* the sequence $\{ x^k \}$ converges.

Using the waveform-Newton method derived in Section 1.4, and performing one step of this Newton method with each waveform relaxation iteration, yields the following Waveform-Newton-Relaxation algorithm (WNR).

<u>Algorithm 2.3.1 (WNR Gauss-Seidel Algorithm for solving Eqn. (1.3))</u>

Comment:
The superscript k denotes the iteration count, the subscript $i \in 1, \cdots, N$ denotes the component index of a vector and ϵ is a small positive number.

$k \leftarrow 0$;
guess waveform $x^0(t)$; $t \in [0,T]$
such that $x^0(0) = x_0$
(for example, set $x^0(t) = x_0$, $t \in [0,T]$);

repeat {
$k \leftarrow k+1$

for all (i **in** N) {

solve

$$\sum_{j=1}^{i-1} C_{ij}(x_1^k, \cdots, x_{i-1}^k, x_i^{k-1}, \cdots, x_n^{k-1}, u)\dot{x}_j^k + \frac{\partial C_{ii}(x_1^k, \cdots, x_{i-1}^k, x_i^{k-1}, \cdots, x_n^{k-1}, u)}{\partial x_i}(x_i^k - x_i^{k-1})\dot{x}_i^k + \sum_{j=i+1}^{n} C_{ij}(x_1^k, \cdots, x_{i-1}^k, x_i^{k-1}, \cdots, x_n^{k-1}, u)\dot{x}_j^{k-1} - f_i(x_1^k, \cdots, x_{i-1}^k, x_i^{k-1}, \cdots, x_n^{k-1}, u) - \frac{\partial f_i(x_1^k, \cdots, x_{i-1}^k, x_i^{k-1}, \cdots, x_n^{k-1}, u)}{\partial x_i}(x_i^k - x_i^{k-1}) = 0$$

for ($x_i^k(t)$; $t \in [0,T]$), with the initial condition $x_i^k(0) = x_{i_0}$.

}

}

until ($\max_{1 \leqslant i \leqslant n} \max_{t \in [0,T]} |x_i^k(t) - x_i^{k-1}(t)| \leqslant \epsilon$)

■ .

Again, each equation has only one unknown variable x_i^k , like Algorithm 1.1.1, but each of the nonlinear equations has been replaced by a simpler time-varying linear problem.

Given the global convergence properties of both the original WR and the WN algorithms, it is not surprising that the WNR algorithm has global convergence properties. We will state the convergence theorem, but will not present the proof because it quite similar to the proof of the basic WR and WN convergence theorems.

Theorem 2.3.1: If, in addition to the assumptions of Eqn. 1.1.3, $C(x,u) \in \mathbb{R}^{n \times n}$ is strictly diagonally dominant uniformly over all $x \in \mathbb{R}^n$ and $\frac{\partial C(x,u)}{\partial x}$ is Lipschitz continuous with respect to x for all u ; then the sequence $\{ x^k \}$ generated by the Gauss-Seidel or Gauss-Jacobi WNR algorithm will converge to the solution of Eqn 1.1.3 for all bounded intervals $[0,T]$.

The linear time-varying systems generated by the WNR algorithm are easier to solve numerically than the nonlinear iteration equations of the basic WR algorithm, but the iteration equations could be futher simplified if the time-varying Jacobian is replaced by a a time-invariant approximation. Approximating the Jacobian will of course destroy the local quadratic convergence of the WN method. But, as an examination of the convergence proof in Section 1.4 indicates, approximating the Jacobian will not destroy the global WN convergence. In addition, loss of quadratic convergence may not be a significant consideration when the Newton method is used in conjunction with a relaxation method, because the relaxation converges linearly and will dominate the rate of convergence of the combined method.

The Modified WNR method then converts the basic WR iteration equations to much simpler linear time-invariant equations. Such systems can be solved with a variety of efficient numerical techniques other than the standard multistep methods. Since the problem is linear, Laplace transform techniques could be used (see Section 2.4). Also, it is possible to use methods based on replacing the solution of the differential equation with a series of orthogonal functions (e.g. Chebyshev polynomials) with unknown coefficients. The problem of finding a solution to the differential

equation is then reduced to determining the coefficients[17].

SECTION 2.4 - TECHNIQUES FOR PIECEWISE-LINEAR SYSTEMS

Large systems of ODE's are usually constructed by analyzing an interconnected network of nonlinear elements. Often, to reduce computation time, the nonlinear elements are not evaluated exactly, but approximated by a linearly interpolated table of values. Not only does this approach reduce the computation time needed to evaluate the nonlinear elements, but it also converts the original system into a piecewise-linear system. Inside the "bounding box" formed by the table entries, the system is linear.

In this section an approach is given for solving piecewise-linear systems using the WR algorithm presented in Section 1.2, and Laplace transforms[18]. This technique not only takes advantage of a loosely coupled original system through the use of the WR algorithm, but as the iterations are computed using Laplace transforms, the piecewise-linearity of the problem is also exploited.

We will start this section by deriving the iteration equations for the WR algorithm applied to a linear differential system. Following, the steps required to compute the WR iteration waveforms using Laplace transform techniques will be described. We will then extend the approach to piecewise-linear systems and introduce a new algorithm.

Consider the following autonomous linear ODE:

$$\dot{x} = Ax \qquad x(0)=x_0 \tag{2.4.1}$$

where $x(t) \in \mathbb{R}^n$ on $t \in [0,T]$ and $A \in \mathbb{R}^n x \mathbb{R}^n$. A is linear; therefore it is Lipschitz continuous with respect to x , and the basic WR convergence theorem guarantees Eqn. (2.4.1) can be solved using Algorithm 1.1.1.

Let $A = L+D+U$ where L is strictly lower triangular, D is diagonal, and U strictly upper triangular. The Gauss-Jacobi WR iteration equations applied to Eqn. (2.4.1) are:

$$\dot{x}^{k+1} = Dx^{k+1} + (L+U)x^k \qquad x(0) = x_0 \tag{2.4.2}$$

Solving Eqn. (2.4.2) by Laplace transforms,

$$x^{k+1}(s) = (sI - D)^{-1}[\,(L+U)x^k(s) + x_0\,] \tag{2.4.3}$$

If $x^k(s)$ is a rational (vector-valued) function with real poles, so is $x^{k+1}(s)$. And by induction, if $x^0(s)$ is a rational function with real poles, then so is $x^k(s)$.

Given that $x^0(s)$ is a rational function with real poles (for example, if $x^0(t)=x_0$, $x^0(s) = \frac{1}{s}x_0$), it is easy to compute the $x^{k+1}(s)$ term from $x^k(s)$. If x^k has the following partial fraction expansion:

$$x^k(s) = \sum_{i=1}^{M^k}(s-\lambda_i)^{-m_i}v^i \tag{2.4.4}$$

where $v^i \in \mathbb{R}^n$, $\lambda_i \in \mathbb{R}$, and m_i and M^k are positive integers, then x^k can be calculated from Eqn. (2.4.2) as

$$x^{k+1}(s) = \sum_{i=1}^{M^k}(s-\lambda_i)^{-m_i}(sI-D)^{-1}(L+U)v^i + (sI-D)^{-1}x_0 \tag{2.4.5}$$

which can be expressed as another partial fraction expansion

$$x^{k+1}(s) = \sum_{i=1}^{M^{k+1}}(s-\lambda'_i)^{-m'_i}w^i. \tag{2.4.6}$$

When necessary, the time domain expression for Eqn. (2.4.6) can be obtained from

$$x^{k+1}(t) = \sum_{i=1}^{M^{k+1}} e^{\lambda'_i t}\frac{t^{m'_i-1}}{(m'_i-1)}w^i. \tag{2.4.7}$$

As indicated above, the partial fraction expansion of x^{k+1} is computed from the partial fraction expansion of x^k in two simple steps. First, M^k multiplications of the matrix $(L+U)$ by the vector v^i must be performed. For large systems, the $(L+U)$ matrix is usually sparse, so the number of scalar multplications required to perform the matrix-by-vector multiplication is $K \bullet n$, where K is the average number of nonzero terms in each row of the matrix. The next step is to compute the w^i vectors

as in Eqn. (2.4.6). This involves performing the partial fraction expansion of the terms of the form $(s-\lambda^i)^{-m_i}(s-d^j)^{-1}$ and $s^{-1}(s-d_j)^{-1}$ where d_j is the jth entry of the diagonal matrix D. The partial fraction expansion can be computed by evaluating $\sum_{i=1}^{M^k} W_i$ residues, where W_i is the number of nonzero elements in $(L+U)v_i$.

The only complication incurred by extending the above technique to a piecewise-linear systems is that the solution will cross into many different linear regions. However, the points in time at which the solution passes from one linear region to the next can be thought of as defining beginning and ending points of windows in time (see Section 2.2). Inside each window the problem is linear, with initial conditions specified by the solution's value at the time it crosses the boundary of the region. The algorithm can then proceed as above inside each window, with only the additional difficulty of determining the boundary crossing times.

Before describing the algorithm, we will formally define a piecewise-linear differential system so that we can precisely define the notion of boundary crossings.

<u>Definition 6.1:</u> Let R^j, $j \in [1, \cdots, r]$ be a collection of closed sets with disjoint interiors, and $U = \bigcup_{i=1}^{r} R^j$. Let $A_j \in \mathbb{R}^{nxn}$ and $b_j \in \mathbb{R}^n$. Then $p: U \rightarrow \mathbb{R}^n$ is such that $p(z) = A_j z + b_j$ is piecewise-linear.

Consider the following differential equation

$$\dot{x}(t) = p(x(t)) \qquad x(0) = x_0 \tag{2.4.8}$$

where $x(t) \in \mathbb{R}^n$ and $p: \mathbb{R}^n \rightarrow \mathbb{R}^n$ is piecewise-linear. We assume that the $A_j \in \mathbb{R}^{nxn}$ and $b_j \in \mathbb{R}^n$ of p are known, and are such that $p(\bullet)$ is everywhere continuous. The WR algorithm for solving systems of the form of Eqn. (2.4.8) using a region by region application of the Laplace transform technique described above is as follows:

<u>Algorithm 2.4.1 (Region by region solution of Eqn. 2.4.8)</u>

$T' = 0$

repeat {

Find $j \in [1, \cdots, n]$ such that $x^k(T') \in R^j$

$x^0(s) = \frac{1}{s} x(T')$

$k \leftarrow 0$.

repeat {

$k \leftarrow k+1$

$x^k(s) = (sI - D)^{-1}[(L+U)x^{k-1}(s) + \frac{1}{s} b_j + x_0]$

Partial fraction expand $x^k(s)$ and collect like terms.

$$x^k(t+T') = \sum_{i=1}^{M^{k+1}} e^{\lambda'_i t} \frac{t^{m'_i - 1}}{(m'_i - 1)} w^i$$

} **until**($\max_{1 \leqslant i \leqslant n} \max_{t \in [T', T]} |x_i^k(t) - x_i^{k-1}(t)| \leqslant \epsilon$)

i.e. until sufficient convergence is obtained.

$\tau = \inf [t > T' \mid x^k(t) \in R^j]$.

Find the time $x^k(t)$ leaves region R^j

$T' = T' + \tau$

} **until** $T' > T$

In order to check convergence it is necessary to compute the time domain expressions for the iterations. The computation of the time domain expression is a relatively expensive operation, and is only required to check WR convergence. It is possible to improve the efficiency of Algorithm 2.4.1 by checking the convergence only every few iterations. Another method for improving the efficiency is somewhat more subtle. Since the WR algorithm usually converges in a nonuniform manner (see Section 2.2), insisting that the relaxation converge to the end of the interval of interest only to then toss away the solution after the boundary crossing time, will usually require many unnecessary WR iterations to be performed. Even though it is impossible to know when the boundary crossings will occur without knowing the exact solution, finding those times for the partially converged solutions can provide good approximations to the boundary crossing times. Since in many cases evaluating the boundary crossing time is very expensive, a fast approximation should be used to provide some reasonable upper bound on the boundary crossing time. This approximation can then be used to shorten the interval over which WR convergence must be assured. Hence, the number of WR iterations required to achieve satisfactory convergence can be reduced at the cost of computing these approximate boundary crossings.

SECTION 2.5 - TECHNIQUES FOR MOS DIGITAL CIRCUITS

In the previous sections of this paper, we have presented several relatively general techniques for improving the efficiency of WR along with corresponding examples related to simulating MOS circuits. We chose examples from this area because, as mentioned in the introduction, WR has proved to be an efficient techique for solving the large nonlinear ODE systems that describe MOS digital circuits. This is due, in part, to characteristics of these ODE systems that are exploited by the general properties of the WR algorithm mentioned above. Specifically, these problems are easily broken up into loosely coupled subsystems across which relaxations converge rapidly, and different state variables change at very different rates, so the ability of the WR algorithm to use different timesteps for different nodes is of great practical advantage.

There are also other properties of MOS circuits that can be exploited by the WR algorithm, and they are much more specific to the circuit simulation problem. In order to complete our presentation of the WR algorithm we will discuss some of these techniques, and will end the section with experimental results that demonstrate the strengths and weaknesses of the WR algorithm in this important area of application.

2.5.1 - THE ODE DESRIPTION OF MOS DIGITAL SUBCIRCUITS

As mentioned previously, the physical behavior of MOS digital circuits can be represented as a system of differential equations of the following form:

$$C(v(t), u(t))\,\dot{v}(t) = f(v(t), u(t)) \qquad v(0) = v_0 \tag{2.5.1}$$

where $v(t) \in \mathbb{R}^n$ is a vector of time-varying voltages, $u(t) \in \mathbb{R}^r$ is a vector of time-varying inputs to the circuit, $C : \mathbb{R}^n x \mathbb{R}^r \rightarrow \mathbb{R}^{nxn}$ is a matrix of nonlinear capacitances, and $f : \mathbb{R}^n v \mathbb{R}^r \rightarrow \mathbb{R}^n$ is a a vector function of the voltages (the currents). For most circuits of practical interest, $C(v, u)$ is strictly diagonally dominant and therefore $C(v, u)^{-1}$ exists and is uniformly bounded with respect to v, u; and f is globally Lipschitz continuous with respect to x for all $u(t) \in \mathbb{R}$.

2.5.2 - ORDERING OF THE SUBCIRCUITS

Consider using relaxation to solve a large system of linear algebraic equations of the form

$$Ax = b$$

where $x, b \in \mathbb{R}^n$ and $A \in \mathbb{R}^{nxn}$. Let L, D, U be strictly lower triangular, diagonal, and strictly upper triangular matrices respectively, such that $A = L + D + U$. If the classical Gauss-Seidel relaxation algorithm is used to solve the above system, the iteration equations can be written in matrix form as:

$$(L + D)\, x^{k+1} - U\, x^{k} = b \tag{2.5.2}$$

Taking the difference between iteration $k+1$ and k yields the following relation:

$$(x^{k+1} - x^{k}) = (L + D)^{-1} U\, (x^{k} - x^{k-1}). \tag{2.5.3}$$

assuming $L + D$ is nonsingular (i.e. the entries in D are nonzero). By the contraction mapping theorem, the relaxation converges if there exists some induced norm on $\mathbb{R}^n$ such that $\| (L + D)^{-1} U \| < 1$.

As an example, suppose that A is lower triangular, that is $U = 0$. Then the relaxation converges in one iteration because the above norm is zero. Of course, the order of the single equations represented by the rows of A in Eqn. 2.5.1 is not unique. One could reverse order the equations of the system, so that b_i becomes $\tilde{b}_{n+1-i}$; $Ax = b$ then becomes $\tilde{A}x = \tilde{b}$ where $\tilde{A}$ is a row permutation of A. In this case, $\tilde{A}$ is upper triangular because A is lower triangular. If the Gauss-Seidel relaxation algorithm is used to solve this new problem, the above norm will no longer be zero, and the relaxation will not converge in one iteration. If fact, it may not converge at all.

Although the above is an extreme example, it does indicate that if the Gauss-Seidel relaxation algorithm is used, it is possible to reorder the system so that the number of iterations required to achieve convergence can be significantly reduced. In particular, a reordering should attempt to move as many of the large elements of the

matrix into the lower triangular portion as possible.

The Gauss-Seidel WR algorithm shares this property with the algebraic relaxation scheme. That is, the Gauss-Seidel WR algorithm will converge more rapidly if the ODE system can be made lower triangular. In the case of MOS digital circuits the function $f(v,u)$ that represents the currents in the circuit can be made mostly lower triangular by a careful reordering of the equations*. This is because the MOS transistor is a highly directional device. The transistor currents at the drain and source terminals are a strong function of the voltage at the gate terminal, but the gate current is almost unaffected by the drain and source voltages. Therefore, if the differential equations of the circuit can be ordered so that the equation that is solved to determine the voltage at the gate of a transistor can be placed before the equation that is solved to determine the voltage at the drain and source of a the transistor, then $f(v,u)$ will be mostly lower triangular.

It is not, in general, possible to so order the equations completely. Flip-flops (See Fig. 3a), and many other types of digital circuits, contain loops that require some transistor's drain or source voltage equation to precede its gate voltage equation. In these cases, convergence is still improved if the equations are ordered so that $f(v,u)$ is as lower triangular as possible.

2.5.3 - TRUNCATED RELAXATION SCHEMES

One of the practical difficulties of applying the WR algorithm to large systems is that the entire waveform for every node in the system must be stored during the iteration process. For systems with many nodes and long waveforms, the required data storage may exceed a computer's available memory. Breaking the simulation interval into "windows" (section 2.2) reduces the storage required for each of the individual

*When we say we wish $f(v,u)$ to be mostly lower triangular, we mean that we would like the terms $\frac{\partial f_i}{\partial v_j}$ that are large for any v_j to be in the lower triangular portion.

waveforms and allows larger systems to be simulated without exceeding available memory, but very large systems will still require extra storage. One approach to solving this problem without changing the WR algorithm is to store the waveforms on a mass storage medium (e.g. a magnetic disk). Then, since only a few waveforms are used at any one time, those waveforms can be moved into a computer's memory, and then moved back out when no longer needed (in much the same way as virtual memory). Another approach to reducing the memory requirement of the WR algorithm is to introduce a truncation approximation, which will work well for certain classes of MOS digital circuits.

Consider the inverter chain example in Fig. 5a. The input to the first inverter and its output for the cases of chains with two, three and four inverters are plotted in Fig. 5b. As the plots indicate, the impact on the output of the first inverter of additional inverters diminishes with the number of inverters. This suggests that the output of each inverter in a long chain could be computed accurately by considering only a few inverters at a time, effectively truncating the computation. For example, compute the output of the first inverter by solving a reduced system which ignores all the inverters after the fifth; then compute the output of the second inverter, using the computed output from the first inverter, by solving a reduced system ignoring all the inverters after the sixth; and continue in this fashion until all the inverter outputs have been computed. The advantage of this approach is that at any point in the procedure, only the waveforms of six inverters are needed. And, as the above simple example indicates, the error due to this truncation will be small.

Of course this algorithm will only produce accurate results if the system of equations are almost unidirectional, like the inverter chain, and the truncation algorithm follows this direction. Combinational circuits, a large class of MOS digital integrated circuits, do share the mostly unidirectional property of the inverter chain. And since

these circuits can be quite large (several thousand nodes), using the truncation approach avoids the difficultly of storing all the waveforms.

2.5.4 - PARTIAL WAVEFORM CONVERGENCE

If the WR algorithm is used to compute the time domain behavior for very large circuits, it is often the case that some pieces of the circuit will converge much more rapidly than others. The overall efficiency of the WR method can be improved if the waveforms that have already converged are not recomputed every subsequent iteration.

To take advantage of partial waveform convergence requires a simple modification to Algorithm 1.1.1. Before giving the exact algorithm we present the following useful definition.

<u>Definition 2.5.1</u> Let

$$\sum_{j=1}^{n} C_{ij}(v(t), u(t))\, v_j(t) = f_i(v(t), u(t)) \qquad v^i(0) = v_{i_0} \tag{2.5.4}$$

be the ith equation of the system in Eqn. 2.5.1. We say $v_j(t)$ is an input to this equation if there exists some $\alpha, t \in \mathbb{R}$ and $z, y \in \mathbb{R}^n$ such that $\sum_{j=1}^{n} C_{ij}(z, u(t))y_j \neq \sum_{j=1}^{n} C_{ij}(z+\alpha e_j, u(t))y_j$ or $f_i(z, u(t)) \neq f_i(z+\alpha e_j, u(t))$, where e_j is the jth unit vector. The input set for the ith equation is the set of $j \in [1, \cdots, n]$ such that $v_j(t)$ is an input.

The WR algorithm is then modified slightly using this notion of the set of inputs to a given ODE.

<u>Algorithm 2.5.1 - WR Algorithm with Partial Waveform Convergence</u>

Comment:
The superscript k denotes the iteration count,
the subscript i denotes the component index
of a vector and ϵ is a small positive number.

$k \leftarrow 0$;
guess waveform $x^0(t)$; $t \in [0,T]$
such that $x^0(0) = x_0$

(for example, set $x^0(t) = x_0, t \in [0,T]$);

repeat {

$k \leftarrow k+1$

foreach (i **in** N) {

Partialflag = *TRUE*

if ($k = 1$) *Partialflag* = *FALSE*

For each ($j < i, j \in$ *input set of* v_i)

if ($\max_{[0,T]} |v_j^k - v_j^{k-1}| > \epsilon$) *Partialflag* = *FALSE*

For each ($j \geqslant i, j \in$ input set of v_i)

if ($\max_{[0,T]} |v_j^{k-1} - v_j^{k-2}| > \epsilon$) *Partialflag* = *FALSE*

if (*Partialflag* = *TRUE*) $v_i^{k+1} = v_j^k$

else solve

$$\sum_{j=1}^{i} C_{ij}(v_1^k, \cdots, v_i^k, v_{i+1}^{k-1}, \cdots, v_n^{k-1}, u)\dot{v}_j^k +$$

$$\sum_{j=i+1}^{n} C_{ij}(v_1^k, \cdots, v_i^k, v_{i+1}^{k-1}, \cdots, v_n^{k-1}, u)\dot{v}_j^{k-1} +$$

$$f_i(v_1^k, \cdots, v_i^k, v_{i+1}^{k-1}, \cdots, v_n^{k-1}, u) = 0$$

for ($v_i^k(t)$; $t \in [0,T]$), with the initial condition $v_i^k(0) = v_{i_0}$.

}

} **until** ($\max_{1 \leqslant i \leqslant n} \max_{t \in [0,T]} |v_i^k(t) - v_i^{k-1}(t)| \leqslant \epsilon$)

that is, until the iteration converges. ■

2.5.5 - EXPERIMENTAL RESULTS

The degree to which the WR algorithm improves circuit simulation efficiency can be traced to two properties of a circuit. The first, mentioned before, is the differences in the rates of change of voltages in the system, as this will determine how much efficiency is gained by solving the subsystems with independent integration timesteps. The second is the amount of coupling between the subsystems. If the subsystems are tightly coupled, then many relaxation iterations will be required to achieve convergence, and the advantage gained by solving each subsystem with its own timestep will be lost. To show this interaction for a practical example, we will use the Relax2.2[13] program to compare the computation time required to simulate a 141-node CMOS memory circuit using standard direct methods and using the WR algorithm. In order to demonstrate the effect of tighter coupling, the CMOS memory circuit will be simulated using several values of a parameter XQC, which is the percent of the gate oxide

capacitance that is considered as gate-drain or gate-source overlap capacitance.

Table 3 - Direct vs WR on a Memory Circuit with Different Couplings

METHOD	XQC	TIMEPOINTS	# WR ITERS	CPU TIME
Direct	0.01	124,539	1	933s
WR	0.01	17,728	2.5	304s
Direct	0.05	122,988	1	945s
WR	0.05	19,199	3	410s
Direct	0.2	118,335	1	917s
WR	0.2	19,193	4	530s
Direct	0.33	115,233	1	895s
WR	0.33	19,315	6.5	707s

The results in Table 3 are exactly as expected. As the coupling increases, the number of WR iterations required increases, and the difference in the simulation time for WR and the direct method decreases.

It is possible to verify for this example our claim of the nature of the efficiencies of using WR. Consider the number of timepoints computed by the direct method versus the number of computed timepoints for the WR method in the final iteration. By comparing these two numbers, a bound can be put on the maximum speed increase that can achieved by solving different subsystems using different timesteps (Note that we are only considering the number of timepoints computed by the WR method in the final iteration, because we are only interested in the number of timepoints needed to accurately compute the given waveform).

The total number of timepoints computed for each of the simulation cases of the memory circuit example is also given in Table 3. This number is the sum of the computed timepoints over all the waveforms in the circuit. If most of the efficency of a decomposition method stems from solving each of the subsystems with its own timestep, then the maximum improvement that could be gained from a decomposition integration method would be the ratio of the number of timepoints computed using the

direct method compared to the number of timepoints computed in the final WR iteration. As can be seen from the Table 3, for the CMOS memory example this ratio is approximately six. In order to compute the actual efficiency of the WR method, the average number of WR iterations performed must be considered, because for each WR iteration the set of timepoints is recomputed. Then, if our claims above are correct, when the ratio of the number of timepoints for the direct method to the number of WR timepoints is divided into the average number of relaxation iterations, the result should be almost equal to the ratio of WR computation time to direct computation time. And as Table 3 shows, it is.

In the above analysis we have ignored an important advantage of relaxation methods: that they avoid large matrix solutions. This is a reasonable assumption for the above example because the matrix operations account for only a small percentage of the computations, even when direct methods are used. However, for much larger problems, of the order of several thousand nodes, the time to perform the large matrix solutions required by direct methods will dominate. In those cases WR methods should compare even more favorably because they avoid these large matrix solutions.

Finally, in Table 4, we present several circuits that have been simulated using RELAX2.2 with direct and WR methods.

Table 4 - CPU Time for Direct Methods vs WR for Several Industrial Circuits

Circuit	Devices	DIRECT	WR
uP Control	232	90s*	45s*
Cmos Memory	621	995s*	308s*
4-bit Counter	259	540s*	299s*
Inverter Chain	250	98s**	38s**
Digital Filter	1082	1800s*	520s*
Encode-Decode	3295	5000s*	1500s*

*On Vax11/780 running Unix using Shichman-Hodges Mosfet model
**On Vax11/780 running VMS using Yang-Chatterjee Mosfet model

CONCLUSIONS AND AKNOWLEDGEMENTS

In this paper several of the WR algorithms that have been proposed in the literature have been analyzed both from a theoretical and practical point of view. We have, however, treated several of these aspects too lightly. In particular, research is needed to more thoroughly understand the nature of WR convergence under discretization, and to characterize systems for which WR algorithms contract in uniform norm. In addition, theoretical and practical work needs to be continued on breaking large systems into smaller subsystems in such a way that relaxation algorithms converge rapidly. Finally, since the WR algorithm has a tremendous amount of inherent parallelism, its application to solving problems using parallel processors is also an important research question.

The authors wish to thank G. Casinovi, J. Kleckner, K. Kundert, R. Newton, and R. Saleh, and the other members of the CAD research group at Berkeley. In addition, we would like to thank J. Beetem, W. Donath, P. Defebve, and H. Y. Hsieh at IBM's Yorktown Research group for many valuable discussions and suggestions. This research has been sponsored in part by DARPA under contract NESC-N39, and in part by the Air Force Office of Scientific Research (AFSC), United States Air Force, under Contract No. F49620-83-C-0005. The United States Government is authorized to reproduce and distribute reprints for government purposes notwithstanding any copyright notation hereon.

REFERENCES

[1] C. William. Gear, "*Numerical Initial Value Problems for Ordinary Differential Equations*," Prentice Hall, 1974.

[2] L.W. Nagel, "SPICE2: A computer program to simulate semiconductor circuits," Electronics Research Laboratory *Rep. No. ERL-M520, University of California, Berkeley, May 1975.*

[3] K. Sakallah and S.W. Director, "An activity-directed circuit simulation algorithm," *Proc. IEEE Int. Conf. on Circ. and Computers*, October 1980, pp.1032-1035

[4] G. De Micheli, A. Sangiovanni-Vincentelli and A.R. Newton, "New Algorithms for the Timing Analysis of Large Circuits" *Proc. 1980 Int. Symp. on Circ. and Syst.*, Houston, 1980.

[5] B.R. Chawla, H.K. Gummel, and P. Kozah, "MOTIS - an MOS timing simulator," *IEEE Trans. Circuits and Systems, Vol. 22, pp. 901-909, 1975*

[6] C. F. Chen and P. Subramaniam, "The Second Generation MOTIS Timing Simulator-- An Efficient and Accurate Approach for General MOS Circuits" *Proc. 1984 Int. Symp. on Circ. and Syst.*, Montreal, Canada, May 1984.

[7] Keepin, W. N., "Multirate Integration of Two Time-Scale Systems," *Ph.D. Dissertation, University of Arizonia, 1980*

[8] C. William Gear, "Automatic Multirate Methods for Ordinary Differential Equations" *Information Processing 80*, North-Holland Pub. Co., 1980

[9A] E. Lelarasmee, A. E. Ruehli, A. L. Sangiovanni-Vincentelli, "The waveform relaxation method for time domain analysis of large scale integrated circuits," *IEEE Trans. on CAD of IC and Syst.*, Vol. 1, n. 3, pp.131-145, July 1982.

[9B] E. Lelarasmee, "The waveform relaxation method for the time domain analysis of large scale nonlinear dynamical systems", *Ph.D. dissertation, University of California, Berkeley*.

[9C] E. Lelarasmee and A. Sangiovanni-Vincentelli, "Some New Results on Waveform Relaxation Algorithms for the Simulation of Integrated Circuits," *Proc. of 1982 IEEE Int. Large Scale System Symposium*, Virginia Beach, Oct. 1982, pp. 371-376

[10] E. Lelarasmee and A. Sangiovanni-Vincentelli, "Relax: a new circuit simulator for large scale MOS integrated circuits", Proc. 19th Design Automation Conference, Las Vegas, Nevada, pp. 682-690, June 1982.

[11] P. Defebve, J. Beetem, W. Donath, H.Y. Hsieh, F. Odeh, A.E. Ruehli, P.K. Wolff,

Sr., and J. White, "A Large-Scale Mosfet Circuit Analyzer Based on Waveform Relaxation" *International Conference on Computer Design*, Rye, New York, October 1984.

[12] C. H. Carlin and A. Vachoux, "On Partitioning for Waveform Relaxation Time-Domain Analysis of VLSI Circuits" *Proc. 1984 Int. Symp. on Circ. and Syst.*, Montreal, Canada, May 1984.

[13] J. White and A. Sangiovanni-Vincentelli, "Relax2.1 - A Waveform Relaxation Based Circuit Simulation Program" *Proc. 1984 Int. Custom Integrated Circuits Conference* Rochester, New York, June 1984.

[14] M. Guarini and O. A. Palusinski, "Integration of Partitioned Dynamical Systems using Waveform Relaxation and Modified Functional Linearization," *1983 Summer Computer Simulation Proceedings*, Vancouver, Canada, July 1983.

[15] W.M.G. van Bokhoven, "An Activity Controlled Modified Waveform Relaxation Method" *1983 Conf. Proc. IEEE ISCAS*, Newport Beach, CA, May 1983.

[16] J. M. Ortega and W.C Rheinbolt, *Iterative Solution of Nonlinear Equations in Several Variables* Academic Press, 1970.

[17] O. Palusinski "Continuous Expansion Methods in Computer Aided Circuit Analysis" *Proc. 1984 Int. Custom Integrated Circuits Conference* Rochester, New York, June 1984

[18] J. Kaye and A. Sangiovanni-Vincentelli, "Solution of piecewise linear ordinary differential equations using waveform relaxation and Laplace transforms", *Proc. 1982 Int. Conf. on Circ. and Comp.*, New York, Sept. 1982.

[19] W.K. Chia, T.N. Trick, and I.N. Haij, "Stability and Convergence Properties of Relaxation Methods for Hierarchical Simulation of VLSI Circuits", *Proc. 1984 Int. Symp. on Circ. and Syst.*, Montreal, Canada, May 1984.

[20] F. Odeh and D Zein, "A Semidirect Method for Modular Circuits" *1983 Conf. Proc. IEEE ISCAS*, May, 1983, Newport Beach, CA

[21] G. Guardabassi "Subsystemwise Simulation of Discrete-Time Interconnected

Dynamical Systems." Electronics Research Laboratory *Rep. No. ERL-M82/8, University of California, Berkeley, Feb. 1982.*

[22] J. K. Hale, *Ordinary Differential Equations* John Wiley and Sons, Inc., 1969

[23] H. A. Antosiewicz, "Newton's method and boundary value problems", *Journal of Computer System Science* 2(2), 177-202 (1968)

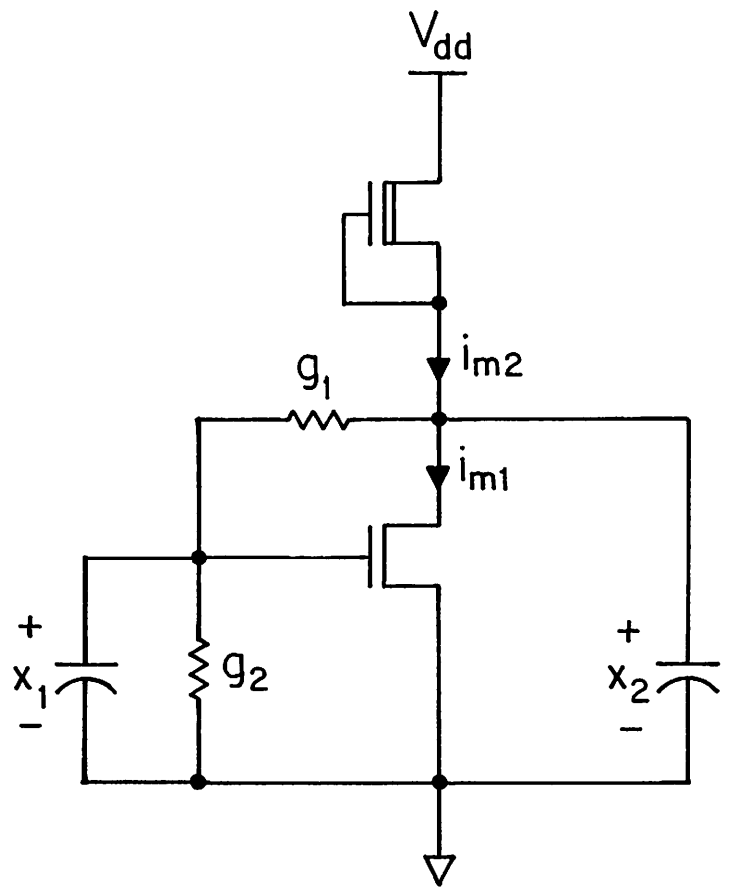
[24] Richard S. Varga *Matrix Iterative Analysis* Prentice-Hall, 1962

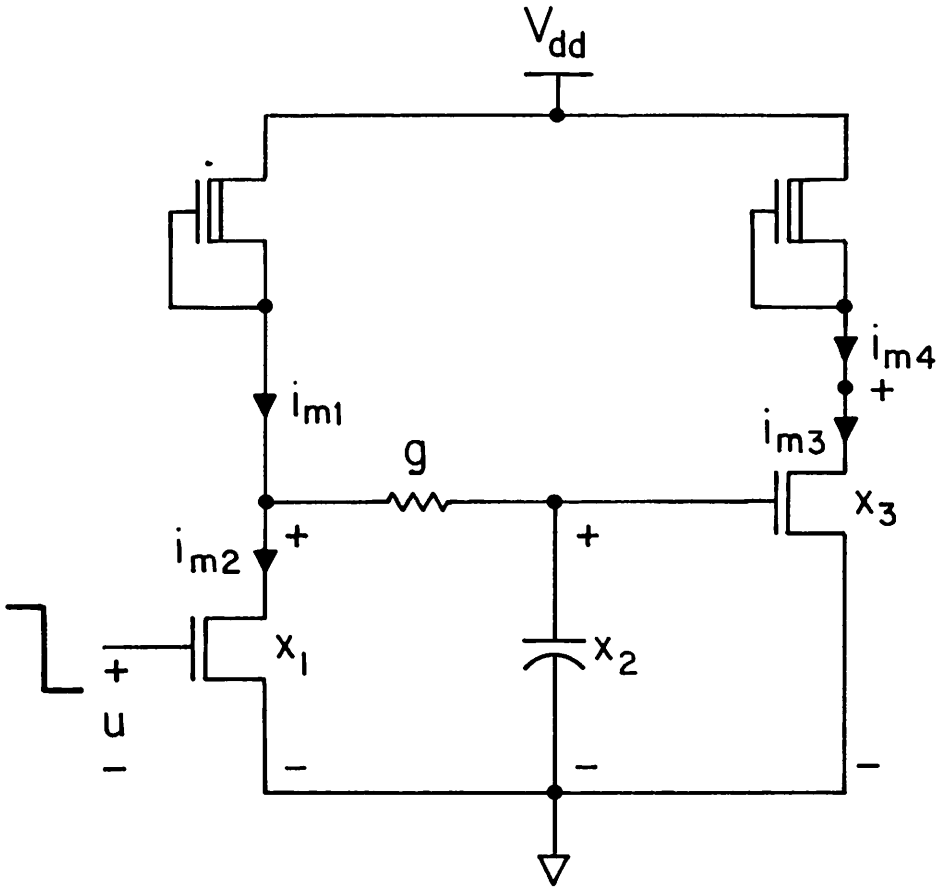
[26] F. Odeh, A. Ruehli, C. H. Carlin "Robustness Aspects of an Adaptive Waveform Relaxation Scheme" *Proc. 1983 Int. Conference on Computer Design*, Port Chester, New York, October, 1983

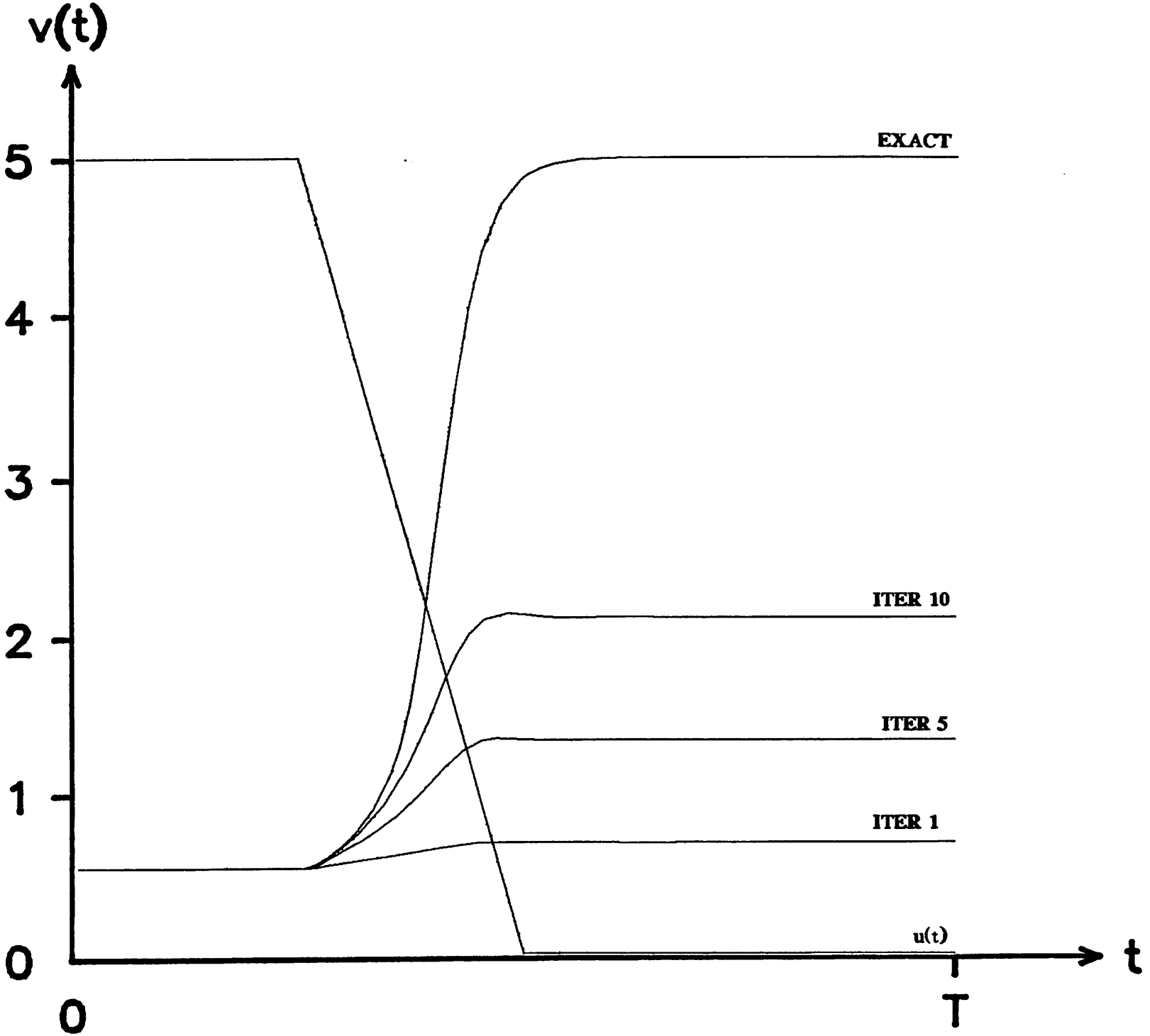
[27] J. More "Nonlinear Generalization of Matrix Diagonal Dominance with Applications to Gauss-Seidel Iterations" *SIAM J. Numer. Anal., Vol. 9, No. 2, June 1972*

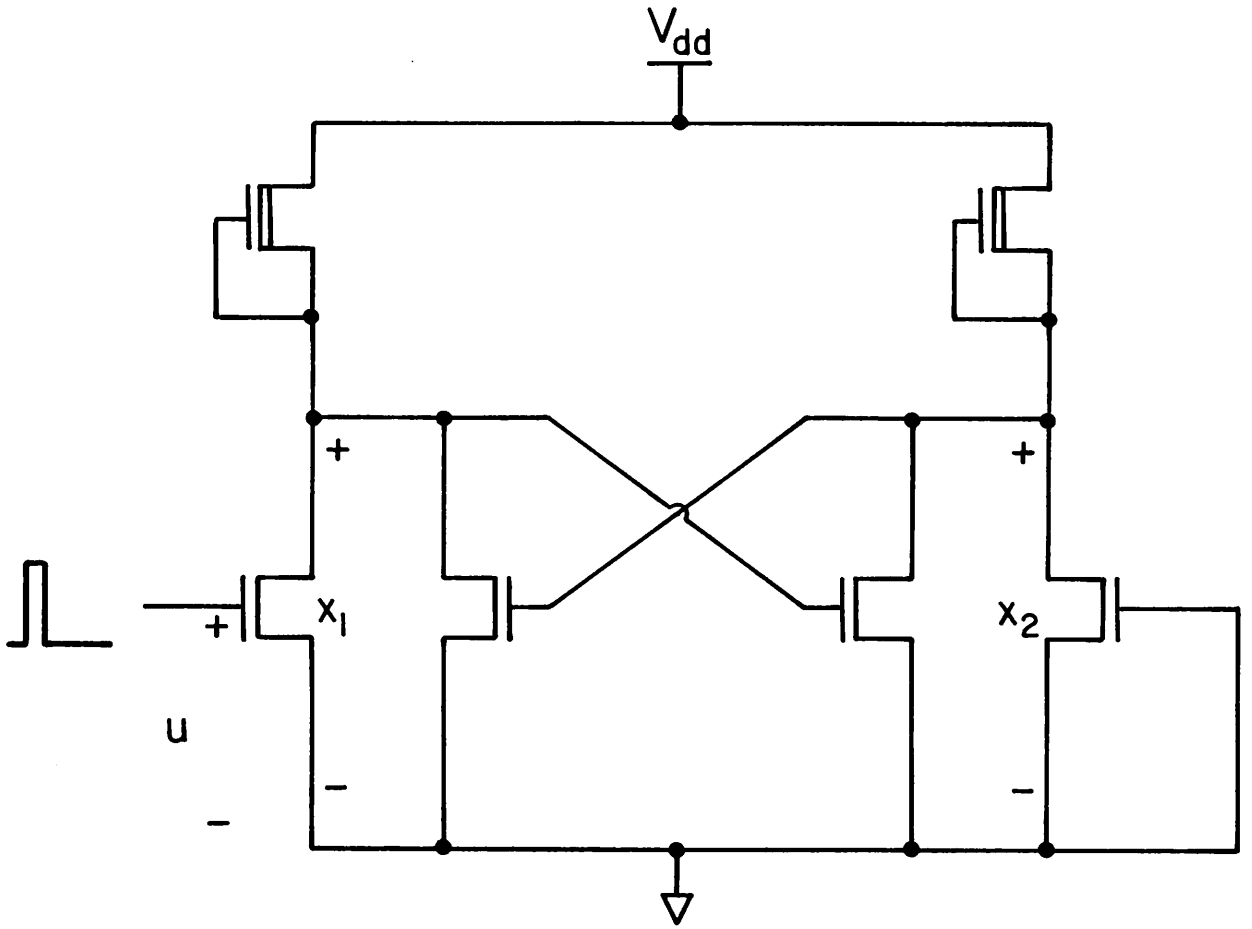
[28] J. White and A.L. Sangiovanni-Vincentelli, "Partitioning Algorithms and Parallel Implementations of Waveform Relaxation Algorithms for Circuit Simulation" *Proc. Int. Symp. on Circ. and Syst.*, Kyoto, Japan, June 1985

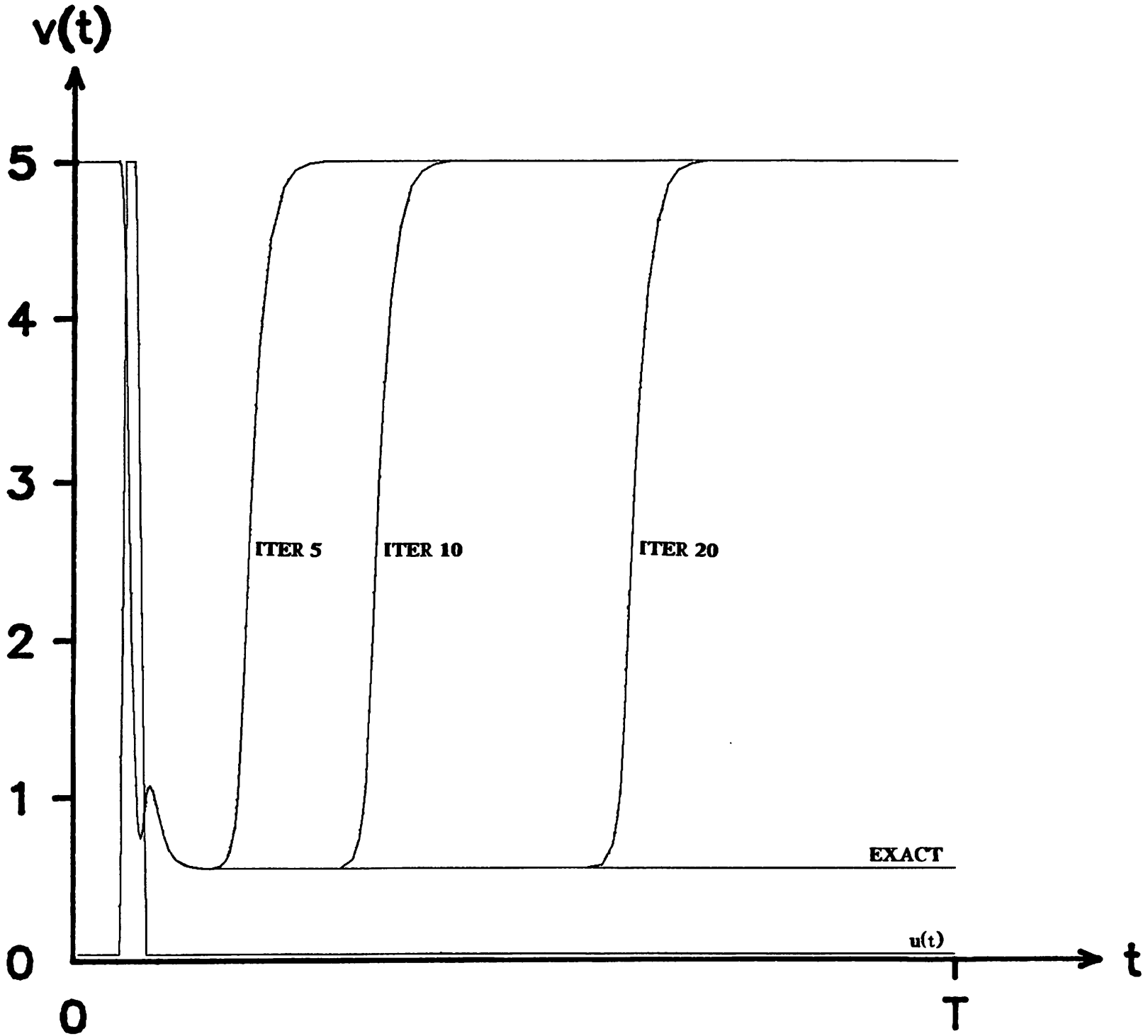
List of Figures

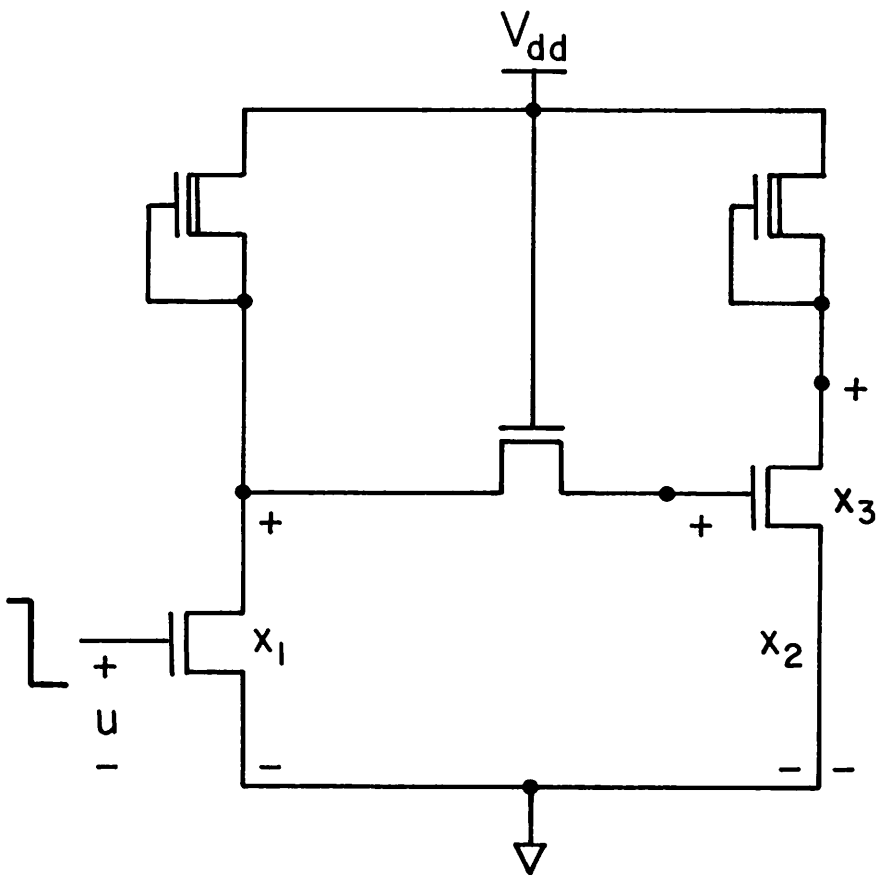

V_{dd}
i_{m2}
g_1
i_{m1}
+
x_1
−
g_2
+
x_2
−

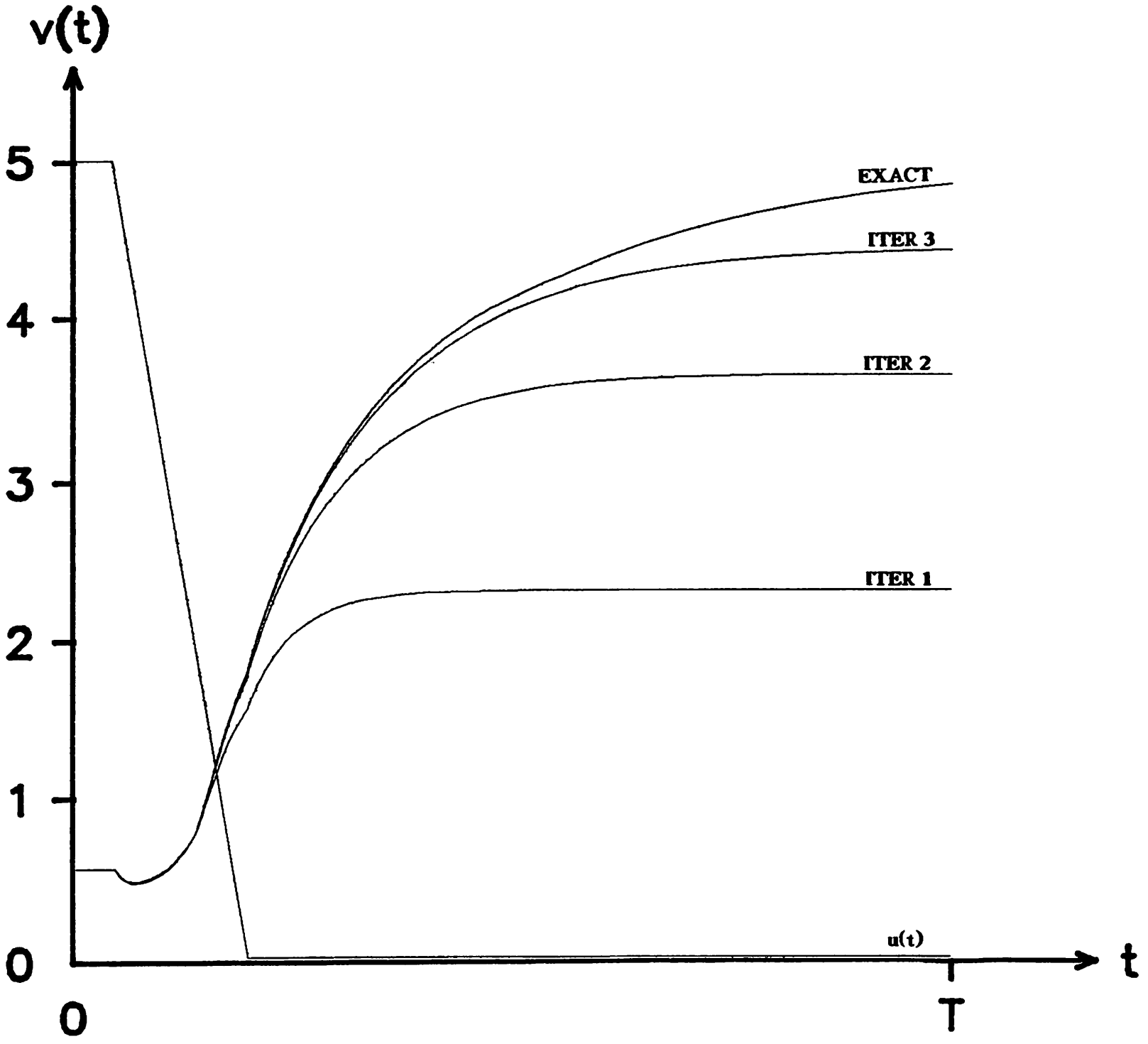

V_{dd}
i_{m1}
i_{m2}
i_{m3}
i_{m4}
g
x_1
x_2
x_3
u
+
−

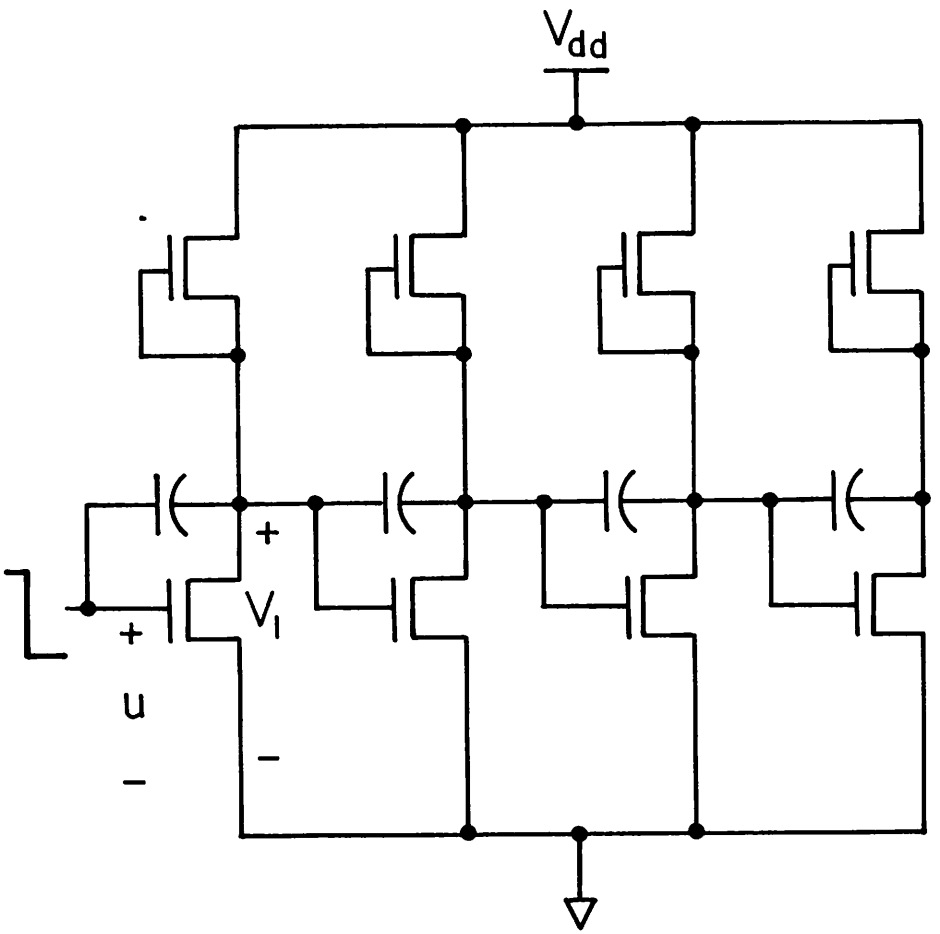

v(t)
5
4
3
2
1
0
0
T
t
EXACT
ITER 10
ITER 5
ITER 1
u(t)

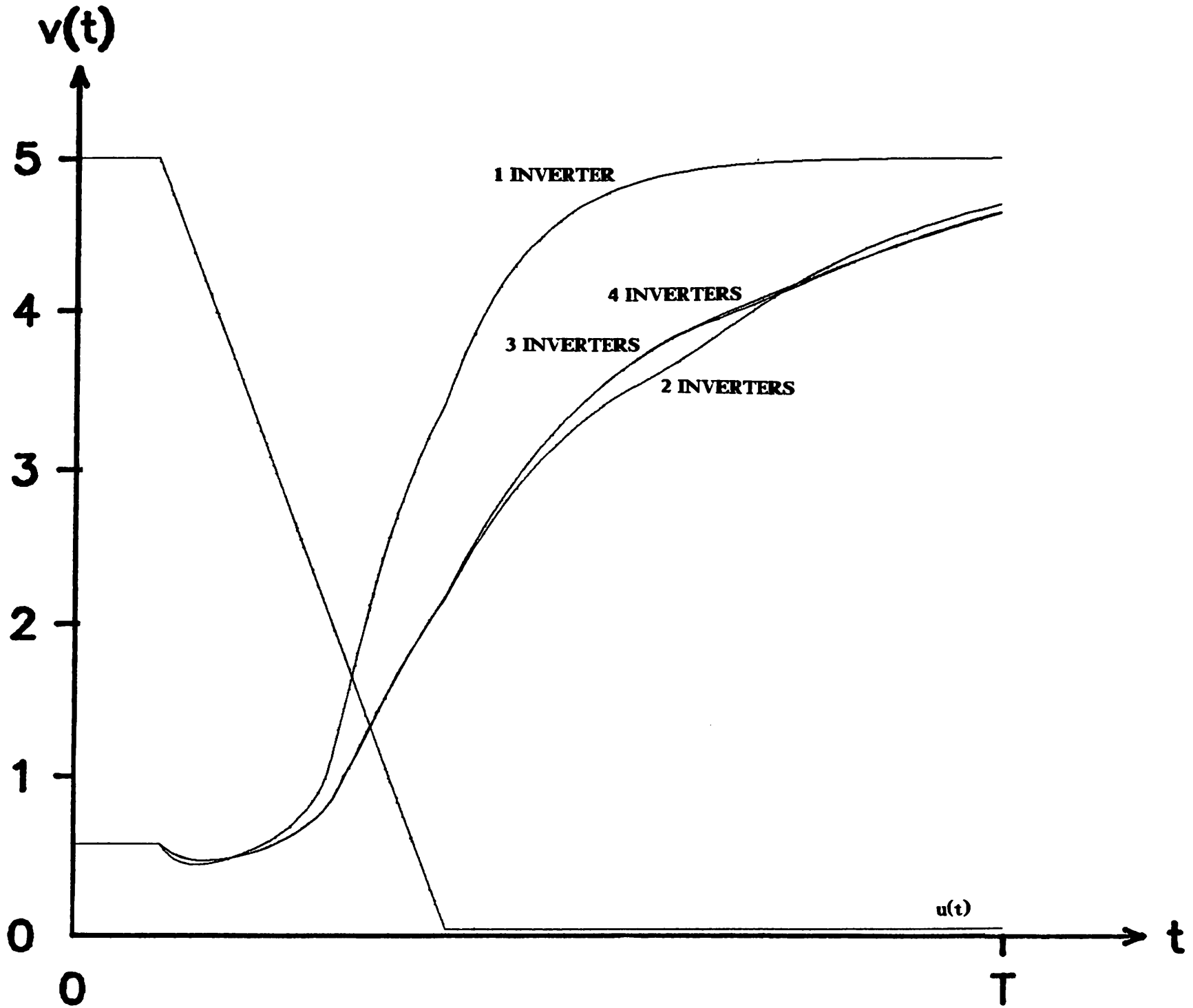

V_{dd}
+
+
x_1
x_2
+
u
−
−
−


v(t)
5
4
3
2
1
0
ITER 5
ITER 10
ITER 20
EXACT
u(t)
t
0
T


V_{dd}
+
+
+
x_3
x_1
x_2
+
u
−
−
−
−


v(t)
5
4
3
2
1
0
0
T
t
EXACT
ITER 3
ITER 2
ITER 1
u(t)


V_{dd}
+
V_1
+
u
−
−


v(t)
5
4
3
2
1
0
0
T
t
1 INVERTER
4 INVERTERS
3 INVERTERS
2 INVERTERS
u(t)