Int. J. Appl. Math. Comput. Sci., 2001, Vol.11, No.5, 1203–1216

MEASURING AND MAINTAINING CONSISTENCY: A HYBRID FTF ALGORITHM

James R. BUNCH*, Richard C. LE BORNE**,
Ian K. PROUDLER***

Due to the versatility as well as its ease of implementation, the Fast Transversal Filters algorithm is attractive for many adaptive filtering applications. However, it is not widely used because of its undesirable tendency to diverge when operating in finite precision arithmetic. To compensate, modifications to the algorithm have been introduced that are either occasional (performed when a predefined condition(s) is violated) or structured as part of the normal update iteration. However, in neither case is any confidence explicitly given that the computed parameters are in fact close to the desired ones. Here, we introduce a time invariant parameter that provides the user with more flexibility in establishing confidence in the consistency of the updated filter parameters. Additionally, we provide evidence through the introduction of a hybrid FTF algorithm that when sufficient time is given prior to catastrophic divergence, the update parameters of the FTF algorithm can be adjusted so that consistency can be acquired and maintained.

Keywords: FTF, numerical stability, consistency

1. Introduction

It is well-known that the Fast Transversal Filters (FTF) algorithm suffers from numerical instability when operating under the effects of finite precision arithmetic. Although the underlying source for this instability has not fully been understood, the symptoms leading toward it have been well-studied (Cioffi and Kailath, 1984; Lin, 1984; Slock and Kailath, 1991): In finite precision arithmetic an updating expression for one of the parameters in the FTF algorithm, the likelihood variable, was found to take on meaningless values just prior to the catastrophic divergence of the algorithm. Since this parameter implicitly defines relationships between itself and other parameters, its loss of physical meaning immediately causes a chain reaction that uncouples the FTF algorithm's parameters, which in turn leads to the divergence.

* Department of Mathematics, University of California, San Diego, La Jolla, California 92093–0112, U.S.A, e-mail: jbunch@ucsd.edu

** Department of Mathematics, Tennessee Technological University, Box 5054, Cookeville, TN 38505, U.S.A., e-mail: rleborne@tntech.edu

*** QinetiQ ltd., Malvern Technology Center, St. Andrews Road, Malvern Worcs., WR14 3PS, United Kingdom, e-mail: i.proudler@signal.qinetiq.com

As a consequence, any practical use of the FTF algorithm needs to monitor any negative consequence from finite precision computations. To address this problem, a so-called rescue variable has been introduced that monitors the quality (consistency) of the algorithm. When the rescue variable takes on a value outside a permissible range of values, some pre-defined action is taken. Another approach, leading to what has been termed the stabilized FTF algorithm (Slock and Kailath, 1991), computes key parameters through multiple algebraic implementations and then defines its updated value from their convex combination.

In this work, we present an alternate means for tracking the finite precision effects. When compared with the rescue variable, the predictor presented here provides an earlier indication of potential FTF divergence. Additionally, we provide evidence through the introduction of a hybrid FTF algorithm that when sufficient time is given prior to catastrophic divergence, the update parameters of the FTF algorithm can be adjusted so that consistency can be acquired and maintained without the need for a complete restart of the algorithm. This algorithm is novel in that it uses the unstable FTF algorithm as often as possible, but when consistency has degraded beyond a threshold, this algorithm relies on a more stable variant for a pre-defined number of update cycles before attempting to use the unstable FTF algorithm again. The possible advantage is in cases in which the unstable FTF algorithm can operate without divergence, providing not only confidence of its solution, but a potential for savings in computational cost.

2. Fast Transversal Filters Algorithm

The FTF algorithm is a result of taking advantage of redundancies arising from the solution of four transversal filtering problems, each related through their use of the same input data; a filter for each of the one-step forward and backward prediction problems, a filter defining the gain vector of the Recursive Least Squares (RLS) algorithm, and a filter to provide the weight vector corresponding to the desired problem being solved. Combined in a special way, these four transversal filters provide the exact solution to the RLS problem at all times and define the FTF algorithm, which is illustrated in Table 1. Parameter definitions can be found in Table 2.

Unfortunately, one of the updated parameters, the likelihood parameter $\gamma_m(n)$ at time index n for a filter of order m, may become negative when its update computations are performed in finite precision arithmetic. It is well-known that in exact arithmetic $0 \leq \gamma_m(n) \leq 1$ and a precursory condition to divergence results from this condition being violated (Haykin, 1991, p.579). The standard approach to monitoring the effect from finite precision arithmetic is the so-called rescue variable (Haykin, 1991, p.707),

$$\zeta_m(n) = \frac{\gamma_{m+1}(n)}{\gamma_m(n)}, \tag{1}$$

Table 1. Summary of the FTF algorithm in which H symbolizes the Hermitian transpose, and updating is performed for a filter of order m, at time-index n.

Prediction:

$$\eta_m(n) = \mathbf{a}_m^H(n-1)\mathbf{u}_{m+1}(n),$$

$$f_m(n) = \gamma_m(n-1)\eta_m(n),$$

$$\mathcal{F}_m(n) = \lambda\mathcal{F}_m(n-1) + \eta_m(n)f_m^*(n),$$

$$\gamma_{m+1}(n) = \lambda\frac{\mathcal{F}_m(n-1)}{\mathcal{F}_m(n)}\gamma_m(n-1),$$

$$\overline{\mathbf{k}}_{m+1}(n) = \begin{bmatrix} 0 \\ \\ \overline{\mathbf{k}}_m(n-1) \end{bmatrix} + \lambda^{-1}\frac{\eta_m(n)}{\mathcal{F}_m(n-1)}\mathbf{a}_m(n-1),$$

$$\mathbf{a}_m(n) = \mathbf{a}_m(n-1) - f_m^*(n)\begin{bmatrix} 0 \\ \\ \overline{\mathbf{k}}_m(n-1) \end{bmatrix},$$

$$\beta_m(n) = \lambda\mathcal{B}_m(n-1)\ \overline{k}_{m+1,m+1}(n),$$

$$\gamma_m(n) = [1 - \beta_m^*(n)\gamma_{m+1}(n)\ \overline{k}_{m+1,m+1}(n)]^{-1}\gamma_{m+1}(n),$$

$$r_m(n) = [1 - \beta_m^*(n)\gamma_{m+1}(n)\ \overline{k}_{m+1,m+1}(n)],$$

$$b_m(n) = \gamma_m(n)\beta_m(n),$$

$$\mathcal{B}_m(n) = \lambda\mathcal{B}_m(n-1) + \beta_m(n)b_m^*(n),$$

$$\begin{bmatrix} \overline{\mathbf{k}}_m(n) \\ \\ 0 \end{bmatrix} = \overline{\mathbf{k}}_{m+1}(n) - \ \overline{k}_{m+1,m+1}(n)\mathbf{c}_m(n-1),$$

$$\mathbf{c}_m(n) = \mathbf{c}_m(n-1) - b_m^*(n)\begin{bmatrix} \overline{\mathbf{k}}_m(n) \\ \\ 0 \end{bmatrix}.$$

Filtering:

$$\xi_m(n) = d(n) - \hat{\mathbf{w}}_m^H(n-1)\mathbf{u}_m(n),$$

$$e_m(n) = \gamma_m(n)\xi_m(n),$$

$$\hat{\mathbf{w}}_m^H(n) = \hat{\mathbf{w}}_m^H(n-1) + \overline{\mathbf{k}}_m(n)e_m^*(n).$$

which is also a positive quantity in exact arithmetic. By utilizing some known relationships between the likelihood parameter and other filter parameters, the rescue variable can alternatively be given as

$$\zeta_m(n) = \frac{\lambda\mathcal{B}_m(n-1)}{\mathcal{B}_m(n)} = 1 - \psi_m(n)\gamma_{m+1}(n)\overline{k}_{m+1,m+1}^*(n), \tag{2}$$

Table 2. Summary of the FTF algorithm parameter definitions: updating is performed for a filter of order m, at time-index n.

Prediction:	
$\eta_m(n)$:	*a priori* forward prediction-error,
$f_m(n)$:	*a posteriori* forward prediction-error,
$\mathcal{F}_m(n)$:	sum of forward predicition-error squares,
$\overline{\mathbf{k}}_{m+1}(n)$:	normalized gain vector,
$\mathbf{a}_m(n)$:	tap-weight: forward prediction filter,
$\beta_m(n)$:	backward *a priori* prediction-error,
$\gamma_m(n)$:	likelihood variable,
$r_m(n)$:	rescue variable: action taken if < 0,
$b_m(n)$:	backward *a posteriori* prediction-error,
$\mathcal{B}_m(n)$:	sum of backward predicition-error squares,
$\mathbf{c}_m(n)$:	backward prediction-error filter tap weight,
Filtering:	
$\xi_m(n)$:	*a priori* prediction-error,
$e_m(n)$:	*a posteriori* prediction-error
$\hat{\mathbf{w}}_m^H(n)$:	*a posteriori* tap-weights.

where λ denotes the forgetting factor while the backward power, the *a priori* backward residual, and the last element of the normalized Kalman gain parameter are given by $\mathcal{B}_m(\cdot)$, $\psi_m(n)$ and $\overline{k}^*_{m+1,m+1}(n)$, respectively. Typically, (2) is the rescue variable implemented in the FTF algorithm. If $\zeta_m(n)$ becomes negative for any n, then the algorithm is "rescued", i.e., it is reinitialized under more relaxed assumptions than the usual pre-windowed case (Lin, 1984). Typically, (2) becomes negative immediately before catastrophic divergence occurs, and hence drastic measures (e.g., reinitialization) are necessary. Conversely, if divergence can be detected before the numerical errors have rendered the filter parameters meaningless, then alternative, less extreme measures other than reinitialization may be possible. It is in this sense that we introduce the hybrid FTF algorithm in Section 4.

3. Tracking Consistency

To motivate the need for tracking consistency during the normal updating of filter parameters, we must first define what is meant by the term consistency, as well as its role in the FTF algorithm. To this end, we will briefly concern ourselves with a generalized formulation of the problem, perturbations involved with its formulation, and how consistency, convergence and stability can be formulated in a single expression.

3.1. Exact Arithmetic

Suppose that a problem $\mathcal{P}$ consists in determining some unknown $x \in \mathbb{R}^n$ which is given by

$$\phi(x) = y, \quad y \in \mathbb{R}^m, \tag{3}$$

i.e., ϕ is the mapping $\phi\colon\ X \longrightarrow Y$. This problem is considered well-posed in the Hadamard sense provided that x exists, it is unique, and the inverse mapping ϕ^{-1} is continuous. It is with the continuity of ϕ^{-1} that the term stability can be applied. If one chooses to formulate the inverse directly, we may simply write

$$\Phi(y) = x, \tag{4}$$

where Φ is a representation of ϕ^{-1}, i.e., only the results are the same; the computational routes, and hence the conditions for continuity, may differ. As is often the case, we do not know y, but instead, an approximation $y_\delta = y(1+\delta),\ 0 \leq \delta < 1$ is given.

Likewise, it is either impossible or undesirable to use (3) (or (4)) directly, but rather an approximation ϕ_δ (or equivalently, Φ_δ) is substituted to give an approximation x_δ to x. In this formulation, assuming that $\mathcal{P}$ is well-posed, the parameter δ defines the family of approximations

$$\phi_\delta(x_\delta) = y_\delta \tag{5}$$

or

$$\Phi_\delta(y_\delta) = x_\delta, \tag{6}$$

for which we are interested in establishing general criteria to ensure the convergence of the solution, i.e., $x_\delta \longrightarrow x$ as $\delta \longrightarrow 0$ (and $y_\delta \longrightarrow y$). As hinted, continuity (pointwise and uniform) is an important cornerstone for this notion of convergence.

When ignoring implementational concerns such as finite precision arithmetic and assuming some means to measure expressions, denoted by $\mathcal{M}$, we can express the direct error (resp. residual error) $\Phi_\delta(y_\delta) - \Phi(y)$ (resp. $\phi_\delta(x_\delta) - \phi(x)$) through

$$\underbrace{\Phi_\delta(y_\delta) - \Phi(y)}_{\mathcal{M}(\text{direct convergence})} = \underbrace{\Phi_\delta(y_\delta) - \Phi_\delta(y)}_{\mathcal{M}(\text{stability})} + \underbrace{\Phi_\delta(y) - \Phi(y)}_{\mathcal{M}(\text{direct consistency})} \tag{7}$$

or

$$\underbrace{\phi_\delta(x_\delta) - \phi(x)}_{\mathcal{M}(\text{res. convergence})} = \underbrace{\phi_\delta(x_\delta) - \phi_\delta(x)}_{\mathcal{M}(\text{stability})} + \underbrace{\phi_\delta(x) - \phi(x).}_{\mathcal{M}(\text{res. consistency})} \tag{8}$$

Hence we see in (7) that if $\Phi_\delta(y) = \Phi(y)$, we have that stability implies convergence. When this is not the case, eqns. (7) and (8) suggest that stability plus consistency implies convergence. Although stability is a necessary condition for convergence, it is not a sufficient one. These notions will now be formalized.

Definition 1. A set of mappings $\{\phi_\delta\colon\ X \longrightarrow Y\}$ is *equicontinuous at* x if, for all $0 \leq \delta < 1$ and for all $\zeta > 0$, there exists an $\eta > 0$, independent of δ, such that $\|\phi_\delta(\hat{x}) - \phi_\delta(x)\| \leq \zeta$ whenever $\|\hat{x} - x\| \leq \eta$.

As an immediate consequence of the continuity of ϕ and its inverse, it is easy to see that the convergence independent of δ is ensured if and only if ϕ is equicontinuous at x, i.e., $x_\delta \longrightarrow x$ as $\delta \longrightarrow 0$ if and only if $\phi(x_\delta) \longrightarrow y$. If, on the other hand, we are interested in the convergence properties of ϕ_δ, then we need a notion of consistency in addition to equicontinuity.

Definition 2. The mapping ϕ_δ is *consistent* with respect to ϕ if for some $\eta > 0$, $\phi_\delta(\hat{x}) \longrightarrow \phi(\hat{x})$ as $\delta \longrightarrow 0$ whenever $\|\hat{x} - x\| < \eta$.

Definition 3. $\Phi_\delta = \phi_\delta^{-1}$ is an approximation to the inverse mapping ϕ^{-1} if and only if ϕ_δ is consistent with respect to the mapping ϕ.

Theorem 1. *For ϕ_δ consistent with respect to ϕ, a sufficient condition for convergence is the equicontinuity of Φ_δ for all $0 \leq \delta < 1$.*

The proof of Theorem 1 can be found in (Chaitin-Chatelin and Frayssé, 1996, p.10).

With respect to the FTF algorithm, let the mapping Φ define the updating process of the parameters. We will see that the second term in (7) may deeply influence the convergence since the derivation of the FTF algorithm assumes, but does not explicitly enforce, certain relationships between the updated parameters. Direct consistency describes the degree in which parameters (that have been perturbed) maintain these relationships (mappings), the loss of which can be enough to cause divergence. If we assume that the FTF algorithm provides usable results for sufficiently small perturbations in its parameters (i.e., stability is retained for small perturbations in its input), the central issue becomes the maintenance of consistency.

Consider now the rescue variable defined in (2). We see that if only a large inconsistency between parameters can be detected, then only a small amount of information from the filter's parameters can be retained. On the other hand, lacking sufficient warning of an impending divergence, one can try to enforce consistency at every update iteration. Slock and Kailath (1991) suggested that multiple evaluations of certain parameters be made (through different computational routes) in which the final update is defined via a convex combination. The idea here is to use these redundantly computed parameters to control the growth of the term $\Phi_\delta(y) - \Phi(y)$ and hence to retain consistency. Since there has not been a means to measure this term, this procedure must be done at each iterative update cycle.

3.2. Iterative Measurement of Consistency

Rather than a static criterion such as that given by (2), we seek a dynamic criterion which can measure the consistency that is implicitly defined throughout the parameters. To this end, by combining (2) with the equation

$$\gamma_{m+1}(n) = \lambda \frac{\mathcal{F}_m(n-1)}{\mathcal{F}_m(n)} \gamma_m(n-1), \tag{9}$$

we find that

$$\frac{\mathcal{B}_m(n-1)}{\gamma_m(n-1)\mathcal{F}_m(n-1)} = \frac{\mathcal{B}_m(n)}{\gamma_m(n)\mathcal{F}_m(n)}. \tag{10}$$

That is, there exists a time invariant relationship in exact arithmetic deviations from which provide a direct measurement of any destabilizing effects. In contrast to (2), time-deviations detected in (10) can be used in conjunction with adjustable thresholds: The tighter the threshold for acceptable performance, the earlier the detection of numerical error propagation that can ultimately lead to catastrophic divergence. This can be seen as a means to establish if, and to what degree, the state vector is on the consistency manifold, as defined independently in (Regalia, 1992; Slock and Kailath, 1992), respectively.

4. Hybrid FTF Algorithm

In this section we suggest a possible implementation of (10) into the FTF algorithm to produce a hybrid FTF algorithm. Instead of restarting the FTF algorithm when a loss of consistency is detected, we will use the algorithm suggested by Slock (1991). If a loss of consistency is detected early enough it is believed that this algorithm can retain and possibly even improve the consistency. However, this cannot be expected to occur after a single update iteration. This suggests that after the first violation of a consistency threshold, which may be adaptive or static in value, there should be a sufficient period of time for the consistency to be regained, i.e., for a certain predefined number of update cycles, the gate is forced to direct incoming data to Slock's algorithm. After this, (10) will use the FTF algorithm until the threshold is violated again.

If the data are such that the FTF algorithm cannot retain consistency for a significant number of update cycles, then the hybrid FTF algorithm will essentially become Slock's algorithm. Conversely, when the FTF algorithm can be used without a significant loss of consistency for a high number of update cycles, the hybrid algorithm will be essentially the standard FTF algorithm. For cases that describe a mixture of these scenarios, the hybrid algorithm will be a combination of both the algorithms. Nevertheless, (10) makes it possible to determine which algorithm is to be used.

5. Experiments

5.1. Detecting Losses in Consistency

Using Matlab, the FTF algorithm as defined in (Haykin, 1991, p.591) and given in Table 1 was evaluated against the standard RLS algorithm. A fifth-order linear prediction problem was set up using a coloured noise sequence $d(n),\ n = 1, \ldots, 5000$ to which observation noise was added. The noise sequence $d(n)$ was generated using a white noise sequence $u(n),\ n = 1, \ldots, 5000$ and the second-order difference equation

$$d(n) = u(n) + 1.371787d(n-1) - .9409d(n-2), \tag{11}$$

where the coefficients were choosen by experimentation to make the FTF algorithm diverge after approximately 5000 iterations. For 100 trials, each differing by the seed used to generate the random input sequence, the iteration in which the rescue variable $\zeta_m(n)$ first became negative was compared against the following criteria involving (10)

Table 3. Mean difference taken between the first violation of (2) (becoming negative) and (12) for a given tolerence ϵ_i.

i	ϵ_i	Mean difference, 100 trials
1	10^{-3}	153
2	10^{-4}	444
3	10^{-5}	758

to indicate probable near-term divergence: the first iteration t violating the criteria defined by

$$\left[\sum_{k=20}^{t}\left|\frac{\mathcal{B}_p(k)}{\gamma_p(k)\mathcal{F}_p(k)}-\frac{\mathcal{B}_p(k-1)}{\gamma_p(k-1)\mathcal{F}_p(k-1)}\right|\right]/t<\epsilon_i, \tag{12}$$

where $\epsilon_i = 10^{-3},\ 10^{-4}$ or $10^{-5},\ t = 20,\ldots,5000$. Here p was randomly selected for each trial (in which $1 \le p \le 5$). For (12) an allowance of approximately 20 iterations was provided for initialization. Table 3 provides the mean difference between the first violation of $\zeta_m(n) < 1$ and (12).

Table 3 supports our expectations: as ϵ decreases, i.e., the less tolerant one is for inconsistencies, the more time (iterations) one has to avoid catastrophic divergence. This may allow for more flexibility regarding the response while the filter parameters are still usable.

The complete history of (2) and (10) for the last trial, denoted as the 'Rescue Variable' and 'BgF', respectively, is given in Fig. 1. The consistency becomes eroded well before the rescue variable becomes negative. For each trial and for each consistency tolerance ϵ_i, the iteration in which the first violation of the rescue variable and (12) was stored is given in Fig. 2. For trials in which divergence did not occur within the 5000 iteration limit, a large positive value was given to represent the iteration for the rescue parameter (dashed line) while the value zero was given as the iteration representing (12). The motivation for this was to visually depict cases in which the FTF did not diverge in the defined number of iterations while the threshold tolerance may or may not have been exceeded. This scenario occurred twice between the tenth and twentieth trial (see Fig. 2, note the two spikes going to zero). For $\epsilon_3 = 10^{-5}$, the consistency criterion was violated after 4000 iterations for both of these trials, but only for one with $\epsilon_2 = 10^{-4}$ and for neither with $\epsilon_1 = 10^{-3}$.

The difference between the first violation of the criteria defined for the rescue variable and (12) was also computed for each trial. Figure 3 gives the results of these computations for each ϵ_i (dashed line) as well as the mean value (solid line). This graphically depicts the relationship between the consistency tolerance ϵ_i and the filter's proximity to divergence (as measured by the rescue variable first becoming negative, which usually occurs close to the point of divergence, cf. Fig. 1).

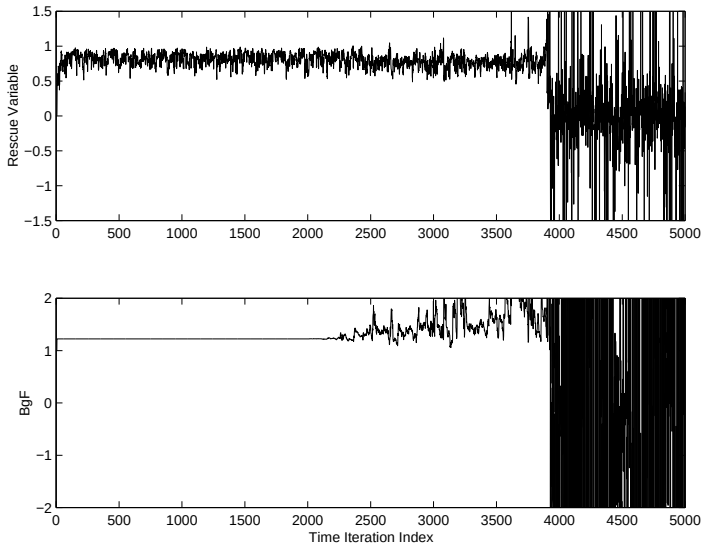

Fig. 1. The complete history for trial 100, in which the rescue variable is compared with (10) as denoted by 'BgF'.

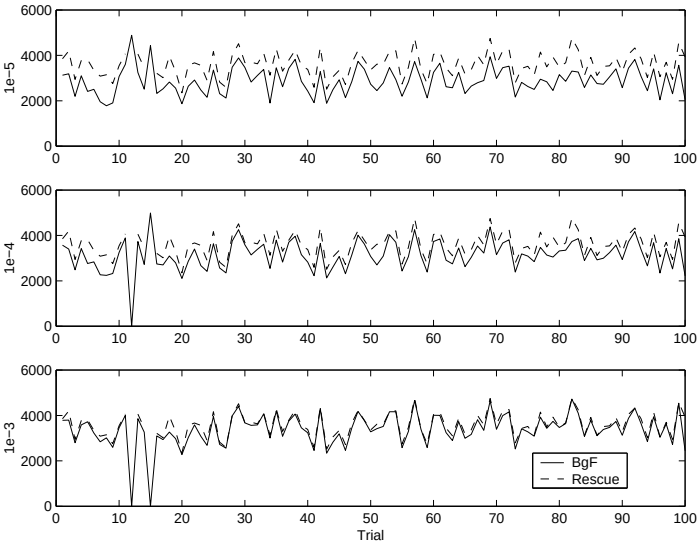

Fig. 2. The iteration in which the first violation occurs for the rescue variable (dashed) and (12), denoted as 'BgF' (solid line). If a violation does not occur for a particular trial, the iteration is set to a large positive value for the rescue variable and to zero for (12).

5.2. Hybrid FTF Algorithm

We now consider whether Slock's algorithm, here to be termed the Stable FTF, can maintain or even improve consistency once it has become degraded. Using the imple-

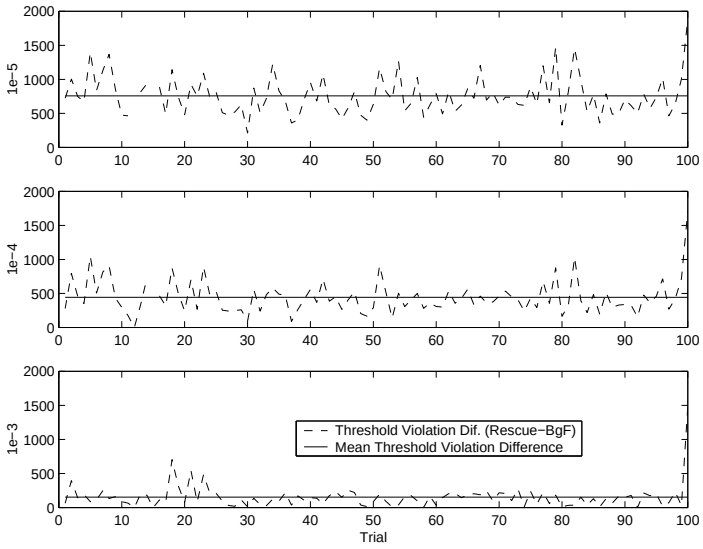

Fig. 3. FTF algorithith update iterations between the first violation occurances for the rescue variable and (12). A positive value n indicates that (12) was satisfied (a consistency violation occurred) n iterations before the rescue variable first became negative. The mean values (solid line) are defined by Table 3.

mentation based on the hybrid FTF algorithm described in Section 4, we used the same linear prediction problem described in Section 5.1 to test three scenarios, each differing through a user-defined input parameter, the *radius*, which determines the coefficents in eqn. (11): (a) When the FTF algorithm is not in danger of diverging, will the hybrid algorithm detect this and allow the FTF algorithm to run its normal course (*radius* = .30)? (b) When the FTF algorithm does not diverge in less than 5000 iterations but loses consistency, will the hybrid algorithm detect the loss in time to bound its decay (*radius*= .50)? (c) When the FTF algorithm loses consistency rapidly, can the hybrid algorithm maintain enough consistency to avoid catastrophic divergence (*radius*= .97)?

For each scenario, one hundred trials were run, each for 5000 iterations and each with a new seed used to generate the input data sequence. For example, Fig. 4 illustrates each of these scenarios for the last (100-th) trial, in which the computed value of the right-hand side of (10) is shown for each iteration. The three scenarios that are defined through the value of the *radius* are denoted by "rad". One can see that for the FTF algorithm the three scenarios are given across the first row. For the first scenario, i.e., rad = .30, there was no apparent loss of consistency in the 5000 iterations that the trial was run. However, for the second scenario, we see that as the trial was completing its last iterations, consistency was beginning to be lost but, nonetheless, the algorithm did not diverge. Finally, for the last scenario, i.e., rad = .97, there was a significant consistency loss and the algorithm even diverged before its last iteration. One also sees that for the other two algorithms, called the

Hybrid FTF and the Stable FTF, the value of the right-hand side of (10) remained constant.

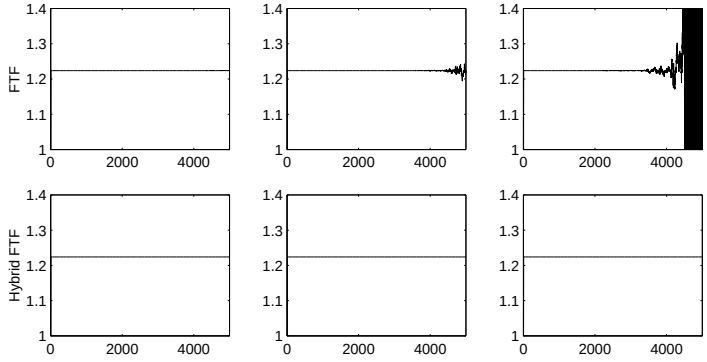

Fig. 4. The value of the right-hand side of (10) for each of 5000 iterations of the 100-th (last) trial. Each column represents one of the three scenarios defined through the parameter 'rad'.

To evaluate the consistency as given by eqn. (10), the relative uncertainty as defined with respect to the RLS algorithm is presented in Fig. 5. Specifically, the right-hand side of (10) was computed for each algorithm from which the consistency for the RLS algorithm was interpreted as the best known value. The relative uncertainty in consistency was then computed as

$$\frac{\|BgF_{RLS} - BgF_i\|}{\|BgF_{RLS}\|}, \tag{13}$$

where i denotes each of the other algorithms, i.e., the FTF, the Hybrid FTF, and the Stable FTF (as given in (Slock and Kailath, 1991)). We see that for each of 100 trials of 5000 iterations, the FTF algorithm shows much worse relative consistency with respect to the RLS algorithm whereas the hybrid and stable algorithms are the same. This demonstrates that the hybrid algorithm, through its use of (12), can sense a degradation in consistency early enough so that by calling (in blocks of 50 iterations) the stable FTF algorithm, a further erosion of consistency, and even divergence, is avoided.

To give a portrait of how consistency is being controlled by the Hybrid FTF algorithm, we have computed the absolute uncertainty with respect to the RLS algorithm for each of the algorithms for the last trial. The result can be found in Fig. 6 for each scenario. If we interpret the desired consistency to be that which is given by the RLS algorithm, then this absolute error (or uncertainty) can be viewed as a measurement of the distance away from the consistency manifold. We see here that for the FTF algorithm and the first scenario (rad = .30), the consistency was indeed beginning to depart from the manifold even if it was not apparent in Fig. 4, where just the value of the right-hand side of (10) was plotted at each iteration of the last trial. Also, in

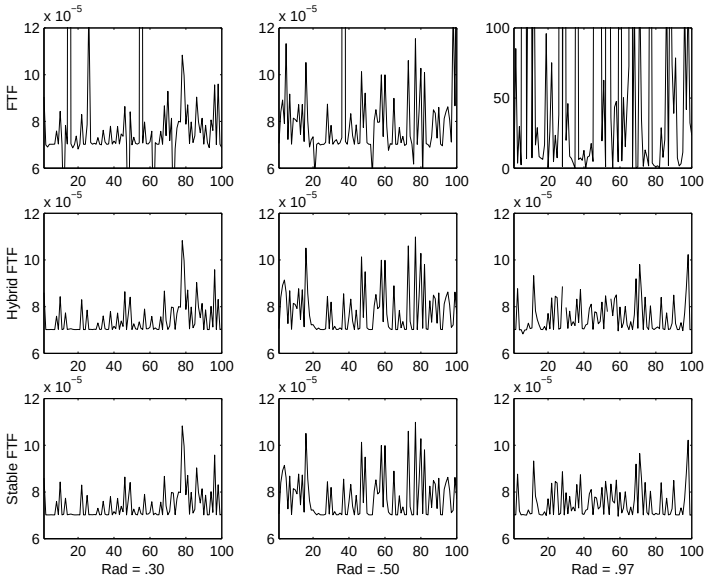

Fig. 5. The relative uncertainty in the consistency as measured with respect to the RLS algorithm. Each of 100 trials is given for the three sencarios that are defined through the value of the parameter 'rad'.

the last scenario (rad = .97) we see that the hybrid FTF algorithm briefly departed from the manifold shortly after the 2000-th iteration but then returned.

Finally, to assess how much work the stable FTF algorithm is to perform, we present in Fig. 7 a plot that reflects the number of calls to the stable FTF required for each trial and for each scenario. The solid horizontal line denotes the mean value over 100 trials for each scenario while the open circles give the numbers of calls for each individual trial. We see that, as anticipated, the fewest number of calls to the stable FTF algorithm occured in the first and second scenario in which the FTF algorithm did not diverge. Between these first two scenarios the mean value was approximately the same. For the last scenario, however, there was indeed a much heavier demand for the stable FTF algorithm. In terms of an operation count, we did not perform an analysis since our implementation was of the most straightforward and possibly naive kind. Hence this is left for ongoing and future work.

6. Conclusion

We have given a new tool to monitor the finite precision effects before the FTF algorithm becomes disconnected from the RLS problem it solves and subsequently suffers from catastrophic divergence. Unlike the existing rescue variable, the proposed device can be given a tolerance for acceptable uncertainty due to finite precision effects. Additionally, this device can signal the eventual divergence of the FTF algorithm sooner

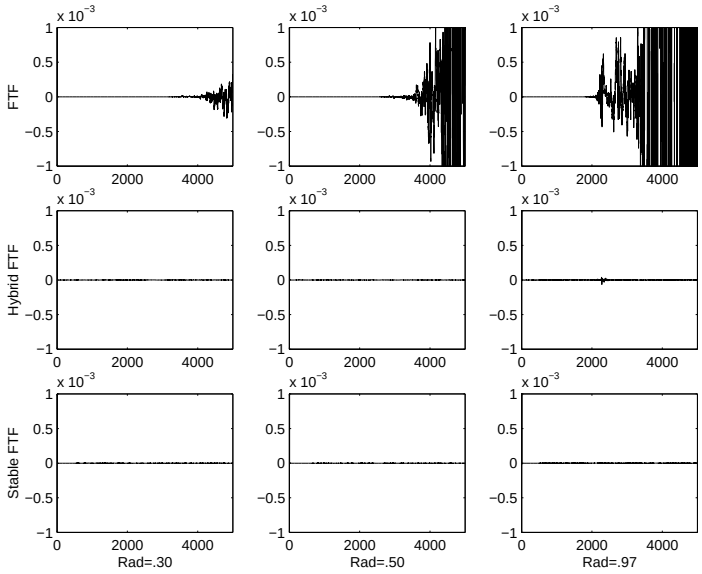

Fig. 6. The absolute uncertainty in consistency as measured with respect to the RLS algorithm. For each scenario this value is plotted for 5000 iterations of the 100-th (last) trial.

than the standard rescue variable, thus allowing a maintenance of consistency before the other parameters become too severely affected.

We have tested this conjecture by introducing a hybrid FTF algorithm that attempts to use the unstable FTF algorithm until consistency has eroded beyond a tolerable level at which time the algorithm essentially turns into the stable FTF algorithm for a predefined number of iterations. After these iterations the hybrid FTF algorithm once again uses the FTF algorithm. Experimentation with the hybrid algorithm revealed that it is indeed possible to switch from the unstable FTF algorithm to the stable one and to regain the lost consistency. We have also seen that the hybrid FTF algorithm performs very much like the stable FTF algorithm but, unlike it, it has a potential to be computationally as efficient as the unstable FTF.

Acknowledgments

Partial support for J.R. Bunch and R.C. Le Borne was provided by NSF grant MIP–9625482.

References

Chaitin-Chatelin F. and Frayssé V. (1996): *Lectures on Finite Precision Computations.* — Philadelphia: SIAM.

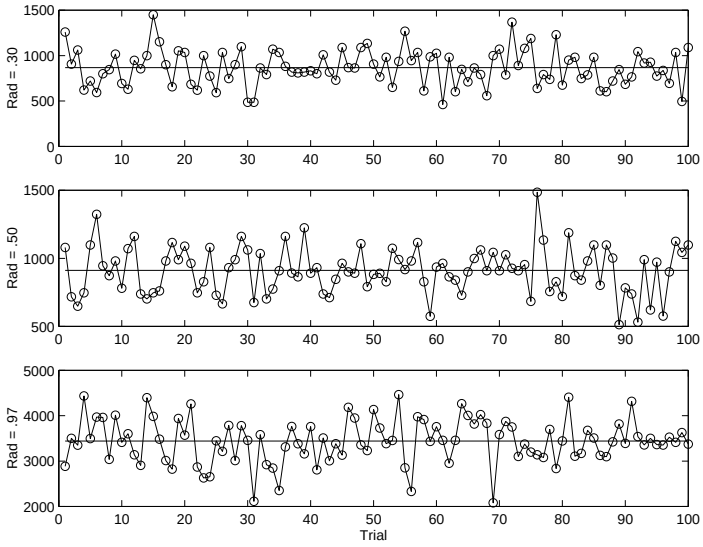

Fig. 7. From the 5000 iterations defining each trial, given is the iteration count in which a call to Slock's stable FTF algorithm was deemed necessary. Otherwise, the regular FTF algorithm was called. The solid horizontal line gives the mean value over the 100 trials for each scenario.

Cioffi J.M. and Kailath T. (1984): *Fast, recursive-least-squares transversal filters for adaptive filtering.* — IEEE Trans. *Acoust. Speech Sign. Process.*, Vol.32, No.2, pp.304–337.

Haykin S. (1991): *Adaptive Filter Theory, 2nd Edn.* — Englewood Cliffs, NJ: Prentice-Hall.

Lin D. (1984): *On digital implementation of the fast Kalman algorithms.* — IEEE Trans. *Acoust. Speech Sign. Process.*, Vol.32, No.5, pp.998–1005.

Regalia P. (1992): *Numerical stability issues in fast least-squares adaptation algorithms.* — Opt. Eng., Vol.31, No.6, pp.1144–1152.

Slock D.T.M. and Kailath T. (1991): *Numerically stable fast transversal filters for recursive least squares adaptive filtering.* — IEEE Trans. *Signal Process.*, Vol.39, No.1, pp.92–114.

Slock D.T.M. and Kailath T. (1992): *Backward consistency concept and round-off error propagation dynamics in recursive least-squares algorithms.* — Opt. Eng., Vol.31, No.6, pp.1153–1169.