Tests and Tolerances for High-Performance Software-Implemented Fault Detection

Michael Turmon, *Member, IEEE* Robert Granat, *Member, IEEE*
Daniel S. Katz, *Senior Member, IEEE* and John Z. Lou

Abstract—We describe and test a software approach to fault detection in common numerical algorithms. Such result checking or algorithm-based fault tolerance (ABFT) methods may be used, for example, to overcome single-event upsets in computational hardware or to detect errors in complex, high-efficiency implementations of the algorithms. Following earlier work, we use checksum methods to validate results returned by a numerical subroutine operating subject to unpredictable errors in data. We consider common matrix and Fourier algorithms which return results satisfying a necessary condition having a linear form; the checksum tests compliance with this condition. We discuss the theory and practice of setting numerical tolerances to separate errors caused by a fault from those inherent in finite-precision floating-point calculations. We concentrate on comprehensively defining and evaluating tests having various accuracy/computational burden tradeoffs, and we emphasize average-case algorithm behavior rather than using worst-case upper bounds on error.

Index Terms—Algorithm-based fault tolerance, result checking, error analysis, aerospace, parallel numerical algorithms.

1 Introduction

THE work in this paper is motived by a specific problem — detecting radiation-induced errors in spaceborne commercial computing hardware — but the results are broadly applicable. Our approach is useful whenever the output of a numerical subroutine is suspect, whether the source of errors is external or internal to the design or implementation of the routine. The increasing sophistication of computing hardware and software makes error detection an important issue. On the hardware side, growing design complexity has made subtle bugs or inaccuracies difficult to detect [1]. Simultaneously, continuing reductions in microprocessor feature size will make hardware more vulnerable to environmentally-induced upsets [2]. On the software side there are similar issues which come from steadily increasing design complexity [3], [4]. There are also new challenges resulting from decomposition and timing issues in parallel code and the advent of high-performance algorithms whose execution paths are determined adaptively at runtime (e.g., 'codelets' [5]).

To substantiate this picture, consider NASA's Remote Exploration and Experimentation (REE) project [6]. The project aims to enable a new type of scientific investigation by moving commercial supercomputing technology into space. Transferring such computational power to space would enable highly autonomous missions with substantial on-board analysis capability, mitigating the control latency that is due to fundamental light-time delays, as well as bandwidth limitations in the link between spacecraft and ground stations. To do this, REE does not desire to develop a single computational platform, but rather to define and demonstrate a process for rapidly transferring commercial high-performance hardware and software into fault-tolerant architectures for space.

Traditionally, spacecraft components have been radiation-hardened to protect against single event upsets (SEUs) caused by galactic cosmic rays and energetic protons. Hardening lowers clock speed and may increase power requirements. Even worse, the time needed to radiation-harden a component guarantees both that it will be outdated when it is ready for use in space, and that it will have a high cost which must be spread over a small number of customers. Use of commodity off-the-shelf (COTS) components in space, on the other hand, implies that faults must be handled in software.

The presence of SEUs requires that applications be self-checking, or tolerant of errors, as the first layer of fault-tolerance. Additional software layers can protect against errors that are not caught by the application [7]. For example, one such layer automatically restarts programs which have crashed or hung. This works in conjunction with self-checking routines: if an error is detected, and the computation does not yield correct results upon retry, an exception is raised and the program may be restarted. Since the goal of the project is to take full advantage of the computing power of the hardware, simple process replication is undesirable.

In this paper, we focus on detecting SEUs in the application layer. An SEU affecting application data is particularly troublesome because it would typically have fewer obvious consequences than an SEU to code, which would be expected to cause an exception. (Memory will be error-detecting and correcting, so faults to memory will be largely screened: most data faults will therefore affect the microprocessor or its cache.) Fortunately, due to the locality of scientific codes, much of their time is spent in certain common numerical subroutines. For example, about 70% of the processing in one of the REE science applications, a Next Generation Space Telescope [8] phase retrieval algorithm [9], is spent in fast Fourier transforms. This characteristic of the applications motivates our emphasis on correctness

• M. Turmon, R. Granat, and J. Z. Lou are with the Data Understanding Systems Group, M/S 126-347; Jet Propulsion Laboratory; Pasadena, CA 91109; USA. email: firstname.lastname@jpl.nasa.gov
• D. S. Katz supervises the Parallel Applications Technologies Group, M/S 126-104; Jet Propulsion Laboratory; Pasadena, CA 91109; USA. email: daniel.s.katz@jpl.nasa.gov

testing of individual numerical subroutines.

Following the COTS philosophy laid out above, our general approach has been to wrap existing parallel numerical libraries (ScaLAPACK [10], PLAPACK [11], FFTW [5]) with fault-detecting middleware. This avoids altering the internals of these highly tuned parallel algorithms. We can treat subroutines that return results satisfying a necessary condition having a linear form; the checksum tests compliance with this necessary condition. Here we discuss the theory and practice of defining tests and setting numerical tolerances to separate errors caused by a fault from those inherent in floating-point calculations.

To separate these two classes of errors, we are guided by well-known upper bounds on error propagation within numerical algorithms. These bounds provide a maximum error that can arise due to register effects — even though the bounds overestimate the true error, they do show how errors scale depending on algorithm inputs. Adapting these bounds to the fault tolerant software setting yields a series of tests having different efficiency and accuracy attributes. To better understand the characteristics of the tests we develop, we perform controlled numerical experiments.

The main contributions of this paper are:

- Comprehensive development of input-independent tests discriminating floating-point roundoff error from fault occurrence.
- Within this context, emphasis on average-case error estimates, rather than upper bounds, to set the fault threshold. This can result in orders of magnitude more room to detect faults, see section 5.
- Validation of the tests by explicitly using standard decision-theoretic tools: the probabilities of false alarm and detection, and their parametric plot via a receiver operating characteristic (ROC) curve.

The sections of this paper continue as follows: we introduce our notation and set out some general considerations that are common to the routines we study. After that, we examine how roundoff errors propagate through numerical routines and develop fault-detection tests based on expected error propagation. Next, we discuss the implementation of our tests and explore their absolute and relative effectiveness via simulation experiments. We close by offering conclusions and discussing possible future research.

1.1 Notation

Matrices and vectors are written in uppercase and lowercase roman letters respectively; A^{T} is the transpose of the matrix A (conjugate transpose for complex matrices). Any identity matrix is always I; context provides its dimension. A real (resp. complex) matrix A is *orthogonal* (resp. *unitary*) if $AA^{\mathsf{T}} = I$. A square matrix is a *permutation* if it can be obtained by reordering the rows of I. The size of a vector v is measured by its *p-norm* $\|v\|_p$, of which the three most useful cases are

$$\|v\|_1 = \sum_i |v_i|, \quad \|v\|_\infty = \max_i |v_i|, \quad \|v\|_2 = \left(\sum_i |v_i|^2\right)^{1/2} \quad .$$

The simplest norm on matrices is the *Frobenius norm*

$$\|A\|_F = \left(\sum_{i,j} |a_{ij}|^2\right)^{1/2} \quad .$$

Other matrix norms may be generated using the vector p-norms, and they are computed as

$$\|A\|_1 = \max_i \sum_j |a_{ij}| \quad , \quad \|A\|_\infty = \max_j \sum_i |a_{ij}|$$
$$\|A\|_2 = \sigma_{\max}(A) \quad .$$

The last is the largest singular value of A and is not trivial to find. All other vector and matrix norms are computed in time linear in the number of matrix elements. All the norms differ only by factors depending only on matrix size and may be regarded as equivalent; in this context write the unadorned $\|\cdot\|$. The *submultiplicative property* holds for each of these norms: $\|Av\| \le \|A\|\|v\|$ and $\|AB\| \le \|A\|\|B\|$. See [12, §2.2, 2.3] or [13, §6.2] for more on all these properties. The numerical precision of the underlying hardware is captured by $\mathbf{u}$, the difference between unity and the next larger floating-point number ($\mathbf{u} = 2.2 \times 10^{-16}$ for 64-bit IEEE floating point).

2 General Considerations

We are concerned with these linear operations:

- Product: find the product $AB = P$, given A and B.
- QR decomposition: factor a square A as $A = QR$, where Q is an orthogonal matrix and R is upper triangular.
- Singular value decomposition: factor a square A as $A = UDV^{\mathsf{T}}$, where D is diagonal and U, V are orthogonal matrices.
- LU decomposition: factor A as $A = PLU$ with P a permutation, L unit lower-triangular, U upper-triangular.
- System solution: solve for x in $Ax = b$ when given A and b.
- Matrix inverse: given A, find B such that $AB = I$.
- Fourier transform: given x, find y such that $y = Wx$, where W is the matrix of Fourier bases.
- Inverse Fourier transform: given y, find x such that $x = n^{-1}W^{\mathsf{T}}y$ where W is the $n \times n$ matrix of Fourier bases.

We note that because standard implementations of multidimensional Fourier transform use one-dimensional transform as a subroutine, a multidimensional transform will inherit a finer-grained robustness by using a fault-tolerant one-dimensional subroutine. In this case, fault tolerance need only be implemented at the base level of the computation. Alternatively, one could use appropriate postconditions at the top level. (E.g., the algorithm `fft2`, given an input matrix X, produces a transform Y such that $Y = WXW$.)

Table I summarizes the operations and the inputs and outputs of the algorithms which compute them. Each of these operations has been written to emphasize that some relation holds among the subroutine inputs and its computed outputs; we call this the *postcondition*. The postcondition for each operation is given in table I. In several cases, indicated in the table, this postcondition is a necessary and sufficient condition (NASC), and thus completely characterizes the subroutine's task. (To see this, consider for example `inv`: if its output B is such that $AB = I$, then $B = A^{-1}$.) For the other operations, the postcondition is only a necessary condition and valid results must enjoy other properties as well — typically a structural constraint like orthogonality.

TABLE I
Operations and Postconditions

Name	Signature	Postcondition	NASC?	Extra conditions if no
Product	$P = \texttt{mult}(A, B)$	$AB = P$	Y	
QR decomposition	$(Q, R) = \texttt{qr}(A)$	$A = QR$	N	Q orthogonal R upper triangular
Singular value decomposition	$(U, D, V) = \texttt{svd}(A)$	$A = UDV^{\mathsf{T}}$	N	D diagonal U, V orthogonal
LU decomposition	$(P, L, U) = \texttt{lu}(A)$	$A = PLU$	N	P a permutation L unit lower-triangular U upper-triangular
System solution	$x = \texttt{solve}(A, b)$	$Ax = b$	Y	
Matrix inverse	$B = \texttt{inv}(A)$	$AB = I$	Y	
Fourier transform	$y = \texttt{fft}(x)$	$y = Wx$	Y	
Inverse Fourier transform	$x = \texttt{ifft}(y)$	$x = n^{-1}W^{\mathsf{T}}y$	Y	

The additional properties needed in this case are given in the table but we do not check them as part of the fault test. (In the case of `lu`, it is customary to store L and U in opposite halves of the same physical matrix, so their structural properties are automatically established.) In either case, identifying and checking the postcondition provides a powerful way to verify the proper functioning of the subroutine.

Before proceeding to examine these operations in detail, we mention two general points involved in designing postcondition-based fault detection schemes for other algorithms. Suppose for definiteness that we plan to check one $m \times n$ matrix. Any reasonable detection scheme must depend on the content of each matrix entry, otherwise some entries would not be checked. This implies that simply performing a check (e.g., computing a checksum of an input or output) requires $O(mn)$ operations. Postcondition-based fault detection schemes thus lose their attractiveness for operations taking $O(mn)$ or fewer operations (e.g. trace, sum, and 1-norm) because it is simpler and more directly informative to achieve fault-tolerance by repeating the computation. (This is an illustration of the "little-oh rule" of [14].) The second general point is that, although the postconditions above are linearly-checkable equalities, they need not be. For example, $\|A\|_2 = \sigma_{\max}(A)$ is bounded by functions of the 1-norm and the ∞-norm, both of which are easily computed but not of linear form. One could test a computation's result by checking postconditions that involve nonlinear functions of the inputs, expressed either as equalities or as inequalities. None of the operations we consider requires this level of generality.

Indeed, the postconditions considered in this paper generically involve comparing two linear maps, which are known in factored form

$$L_1 L_2 \cdots L_p \stackrel{?}{=} R_1 R_2 \cdots R_q \quad . \tag{1}$$

This check can be done exhaustively via n linearly independent probes for an $n \times n$ system — which would typically introduce about as much computation as recomputing the answer from scratch. On the other hand, a typical fault to data fans out across the matrix outputs, and a single probe would be enough to catch most errors:

$$L_1 L_2 \cdots L_p w \stackrel{?}{=} R_1 R_2 \cdots R_q w \tag{2}$$

for some probe vector w.

For certain operations (`solve`, `inv`) such postcondition checks are formalizations of common practice. Freivalds [15] used randomized probes w to check multiplication. This approach, known as result-checking (RC), is analyzed in a general context by Blum and Kannan [16]; for further analysis see [1], [14], [17]. Use of multiple random probes can provide probabilistic performance guarantees almost as good as the exhaustive scheme indicated in eqn. 1, without performing all n probes.

Linear checks on matrix operations are also the basis for the checksum-augmentation approach introduced by Huang and Abraham [18] for systolic arrays, under the name algorithm-based fault tolerance (ABFT). The idea has since both moved outside its original context of systolic array computation, and has also been extended to LU decomposition [19], QR and related decompositions [20], [21], SVD, FFT [22], [23], and other algorithms. The effects of multiple faults, and faults which occur during the test itself, have been explored through experiment [24]. Fault location and correction, and use of multiple probe vectors have also been studied in the ABFT context [25], [26].

An earlier paper [20] uses roundoff error bounds to set the detection threshold for one algorithm, the QZ decomposition. Other work ([27], [28]) uses running error analysis to derive methods for maintaining roundoff error bounds for three algorithms. Running error analysis is a powerful technique [13, §3.3] with the potential to give accurate bounds, but requires modifying the internals of the algorithm: we have preferred to wrap existing algorithms without modification. As we shall see, norm-based bounds provide excellent error-detection performance which reduces the need for running error analysis in many applications. Error bounds have also been used to derive input-independent tests for system solution [29]; the paper cleverly builds in one round of iterative refinement both to enable usable error bounds, and also to allow correction of errors in the first computed solution. In contrast to the above papers, our

work emphasizes average-case bounds and attempts to provide a comprehensive treatment of error detection for many operators.

There are two designer-selectable choices controlling the numerical properties of the fault detection system in (2): the checksum weights w and the comparison method indicated by $\stackrel{?}{=}$. When no assumptions may be made about the factors of (2), the first is relatively straightforward: the elements of w should not vary greatly in magnitude so that results figure essentially equally in the check. At the minimum, w must be everywhere nonzero; better still, each partial product $L_{p'} \cdots L_p w$ and $R_{q'} \cdots L_q w$ of (2) should not vary greatly in magnitude.

For Fourier transforms, this yields a weak condition on w: it should at least be chosen so that neither the real or imaginary parts of w, nor those of its Fourier transform, vary radically in magnitude. This rules out many simple functions (like the vector of all ones) which exhibit symmetry and have sparse transforms. For FFT checksum tests, we generated a fixed test vector of randomly chosen complex entries: independent and identically distributed unit normal variates for both real and imaginary parts. The ratios between the magnitude of the largest and smallest elements of this w and its transform are about 200. We see in sec. 5.3 that this choice is far superior to the vector of all ones and other elementary choices. For the matrix operations, on the other hand, little can be said in advance about the factors (but see [30]). We are content to let w be the vector of all ones. Our implementation allows an arbitrary w to be supplied by those users with more knowledge of expected factors.

3 Error Propagation

After the checksum vector, the second choice is the comparison method. As stated above, we perform comparisons using the corresponding postcondition for each operation. To develop a test that is roughly independent of the matrices at hand, we use the well-known bounds on error propagation in linear operations. In what follows, we develop a test for each operation of interest. For each operation, we first cite a result bounding the numerical error in the computation's output, and then we use this bound to develop a corollary defining a test which is roughly independent of the operands. Those less interested in this machinery might review the first two results and skip to section 4.

It is important to understand that the error bounds given in the results are rigorous, but we use them *qualitatively* to determine the general characteristics of roundoff in an algorithm's implementation. The upper bounds we cite are based on various worst-case assumptions (see below) and they typically predict roundoff error much larger than practically observed. In the fault tolerance context, using these bounds uncritically would mean setting thresholds too high and missing some fault-induced errors. Their value for us, and it is substantial, is to indicate how roundoff error scales with different inputs. (See [12, §2.4.6], [29], and section 5 for more on this outlook.) This allows fault tolerant routines to *factor out the inputs*, yielding performance that is more nearly input-independent. Of course, some problem-specific tuning will likely improve performance. Our goal is to simplify the tuning as much as possible.

The dependence of error bounds on input dimension, as opposed to input values, is subtler. This is determined by the way individual roundoff errors accumulate within a running algorithm. Several sources of estimation error exist:

- Roundoff errors accumulate linearly in worst case, but cancel in average case [13, §2.6], [31, §14].
- Error analysis methods often have inaccuracies or artifacts at this level of detail (e.g., [13, note to thm. 18.9]). In particular, the constants will vary depending on which matrix norm is used.
- Carefully implemented algorithms decompose summations so that roundoff errors accumulate slowly [13, §3.1].
- Some hardware uses extended precision to store intermediate results [13, §3.1].

For these reasons, numerical analysts generally place little emphasis on leading constants that are a function of dimension. One heuristic is to replace the dimension-dependent constants in the error bound by their square root [13, §2.6], based on a central limit theorem argument applied to the sum of individual roundoff errors. (The heuristic is justified on theoretical grounds when a randomized detection algorithm [14, alg.2.1.1] is used under certain conditions.) Because of these uncertainties, we give tests involving dimensional constants only when the bound is known to reflect typical behavior.

Result 1 ([12],§2.4.8) Let $\hat{P} = \texttt{mult}(A, B)$ be computed using a dot-product, outer-product, or gaxpy-based algorithm. The error matrix $E = \hat{P} - AB$ satisfies

$$\|E\|_\infty \leq n\|A\|_\infty \|B\|_\infty \mathbf{u} \tag{3}$$

where n is the dimension shared by A and B.

Corollary 2: An input-independent checksum test for `mult` is

$$d = \hat{P}w - ABw \tag{4}$$

$$\|d\|_\infty / (\|A\|_\infty \|B\|_\infty \|w\|_\infty) \stackrel{>}{<} \tau\mathbf{u} \tag{5}$$

where τ is an input-independent threshold.

The test is expressed as a comparison (indicated by the $\stackrel{>}{<}$ relation) with a threshold; the latter is a scaled version of the floating-point accuracy. If the discrepancy is larger than $\tau\mathbf{u}$, a fault would be declared, otherwise the error is explainable by roundoff.

Proof: The difference $d = Ew$ so, by the submultiplicative property of norms and result 1,

$$\|d\|_\infty \leq \|E\|_\infty \|w\|_\infty \leq n\|A\|_\infty \|B\|_\infty \|w\|_\infty \mathbf{u}$$

and the dependence on A and B is removed by dividing by their norms. The factor of n is unimportant in this context, as noted in the remark beginning the section. ∎

Result 3 ([13],thms.18.4,18.9) Let $(\hat{Q}, \hat{R}) = \texttt{qr}(A)$ be computed using a Givens or Householder QR decomposition algorithm applied to the $m \times n$ matrix A, $m \geq n$. The backward error matrix E defined by $A + E = \hat{Q}\hat{R}$ satisfies

$$\|E\|_F \leq \rho m^2 n \|A\|_F \mathbf{u} \tag{6}$$

where ρ is a small constant for any matrix A.

Corollary 4: An input-independent checksum test for `qr` is

$$d = \hat{Q}\hat{R}w - Aw \tag{7}$$

$$\|d\|_F / (\|A\|_F \|w\|_F) \stackrel{>}{<} \tau\mathbf{u} \tag{8}$$

where τ is an input-independent threshold.

Proof: Let $d = \hat{Q}\hat{R}w - Aw$; then $d = Ew$ where E is the error matrix bounded in result 3. We claim that an input-independent checksum test for `qr` is

$$\|d\|_F / (\|A\|_F \|w\|_F) \lesssim \tau\mathbf{u} \quad . \tag{9}$$

Indeed, by the submultiplicative property and result 3,

$$\|d\|_F \le \|E\|_F \|w\|_F \le \rho m^2 n \|A\|_F \|w\|_F \mathbf{u}$$

and the dependence on A is removed by dividing by its norm. The constant ρ is independent of A, the dimensional constants are discarded, and the claim follows. ∎

To convert this test, which uses Frobenius norm, into one using the ∞-norm, recall that these two norms differ only by constant factors which may be absorbed into τ.

Result 5 ([12],§5.5.8) Let $(\hat{U}, \hat{D}, \hat{V}) = \mathtt{svd}(A)$ be computed using a standard singular value decomposition algorithm. The forward error matrix E defined by $A + E = \hat{U}\hat{D}\hat{V}^\mathsf{T}$ satisfies

$$\|E\|_2 \le \rho \|A\|_2 \mathbf{u} \tag{10}$$

where ρ is a small constant for any matrix A.

Corollary 6: An input-independent checksum test for `svd` as applied to any matrix is

$$d = \hat{U}\hat{D}\hat{V}^\mathsf{T} w - Aw \tag{11}$$

$$\|d\|_\infty / (\|A\|_\infty \|w\|_\infty) \lesssim \tau\mathbf{u} \tag{12}$$

where τ is an input-independent threshold.

Proof: Let $d = \hat{U}\hat{D}\hat{V}^\mathsf{T} w - Aw$; then $d = Ew$ where E is the error matrix bounded in result 5. We claim that an input-independent checksum test for `svd` is

$$\|d\|_2 / (\|A\|_2 \|w\|_2) \lesssim \tau\mathbf{u} \quad . \tag{13}$$

Indeed, by the submultiplicative property and result 5,

$$\|d\|_2 \le \|E\|_2 \|w\|_2 \le \rho \|A\|_2 \|w\|_2 \mathbf{u}$$

and the dependence on A is removed by dividing by its norm. The constant ρ is neglected, and the claim follows. This test, which uses 2-norm, may be converted into one using the ∞-norm as with `qr`. ∎

The test for SVD has the same normalization as for QR decomposition.

For some of the remaining operations, we require the notion of a *numerically realistic* matrix. The reliance of numerical analysts on some algorithms is based on the rarity of certain pathological matrices that cause, for example, pivot elements in decomposition algorithms to grow exponentially. Note that matrices of unfavorable condition number are not less likely to be numerically realistic. (Explaining the pervasiveness of numerically realistic matrices is "one of the major unsolved problems in numerical analysis" [13, p.180] and we do not attempt to summarize research characterizing this class.) In fact, even stable and reliable algorithms can be made to misbehave when given such unlikely inputs. Because the underlying routines will fail under such pathological conditions, we may neglect them in designing an fault tolerant system: such a computation is liable to fail even without faults. Accordingly, certain results below must assume that the inputs are numerically realistic to obtain usable error bounds.

Result 7 ([12],§3.4.6) Let $(\hat{P}, \hat{L}, \hat{U}) = \mathtt{lu}(A)$ be computed using a standard LU decomposition algorithm with partial pivoting. The backward error matrix E defined by $A + E = \hat{P}\hat{L}\hat{U}$ satisfies

$$\|E\|_\infty \le 8n^3 \rho \|A\|_\infty \mathbf{u} \tag{14}$$

where the *growth factor* ρ depends on the size of certain partial results of the calculation.

Trefethen and Bau [31, §22] describe typical behavior of ρ in some detail, finding that it is typically of order $\sqrt{n}$ for numerically realistic matrices. We note in passing that this is close to the best possible bound for the discrepancy, because the error in simply writing down the matrix A must be of order $\|A\|\mathbf{u}$. The success of numerical linear algebra is in finding a way to factor realistic matrices A while incurring only a small penalty ρ beyond this lower bound.

Corollary 8: An input-independent checksum test for `lu` as applied to numerically realistic matrices is

$$d = \hat{P}\hat{L}\hat{U}w - Aw \tag{15}$$

$$\|d\|_\infty / (\|A\|_\infty \|w\|_\infty) \lesssim \tau\mathbf{u} \tag{16}$$

where τ is an input-independent threshold.

Proof: We have $d = Ew$ so, by the submultiplicative property of norms and result 7,

$$\|d\|_\infty \le \|E\|_\infty \|w\|_\infty \le 8n^3 \rho \|A\|_\infty \|w\|_\infty \mathbf{u} \quad .$$

As before, the factor of $8n^3$ is unimportant in this calculation. For numerically realistic matrices, the growth factor ρ is also negligible, and the indicated test is recovered by dividing by the norm of A. ∎

Result 9 ([12],§3.4.6) Let $\hat{x} = \mathtt{solve}(A, b)$ be computed using a standard LU decomposition algorithm with partial pivoting, and back-substitution. The backward error matrix E defined by $(A + E)\hat{x} = b$ satisfies

$$\|E\|_\infty \le 8n^3 \rho \|A\|_\infty \mathbf{u} \tag{17}$$

with ρ as in result 7.

In this case, the result is itself a vector: this vector is evaluated directly within the postcondition $Ax = b$, omitting a checksum operation.

Corollary 10: An input-independent test for `solve` as applied to numerically realistic matrices is

$$d = A\hat{x} - b \tag{18}$$

$$\|d\|_\infty / (\|A\|_\infty \|\hat{x}\|_\infty) \lesssim \tau\mathbf{u} \tag{19}$$

where τ is an input-independent threshold.

Proof: We have $d = -E\hat{x}$ so, by the submultiplicative property of norms and result 9,

$$\|d\|_\infty \le \|E\|_\infty \|\hat{x}\|_\infty \le 8n^3 \rho \|A\|_\infty \|\hat{x}\|_\infty \mathbf{u} \quad .$$

As before, leading factors are dropped and the indicated test is recovered by dividing. ∎

There are several ways to compute an inverse matrix, typically based on an initial factorization $A = PLU$. The LINPACK `xGEDI`, LAPACK `xGETRI`, and Matlab `inv` all use the same variant which next computes U^{-1} and then solves $BPL = U^{-1}$ for B.

Result 11 ([13],§13.3.2) Let $\hat{B} = \texttt{inv}(A)$ be computed as just described. Then the left residual satisfies

$$\|\hat{B}A - I\|_\infty \leq 8n^3\rho\,\|\hat{B}\|_\infty\|A\|_\infty\mathbf{u} \tag{20}$$

with ρ as in result 7.

Proof: The cited proof derives the elementwise bound

$$|\hat{B}A - I| \leq 2n|\hat{B}||L||U|\mathbf{u}$$

where $|M|$ is the matrix of absolute values of the entries of M, and the inequality holds elementwise. Just as for `lu`, one can bound $l_{ij} \leq 1$ and $u_{ij} \leq \rho\|A\|_\infty$, which allows conversion of the elementwise bound into the norm bound at the end of §13.3.2. ∎

Corollary 12: An input-independent checksum test for `inv` as applied to numerically realistic matrices is

$$d = \hat{B}Aw - w \tag{21}$$

$$\|d\|_\infty/(\|A\|_\infty\|\hat{B}\|_\infty\|w\|_\infty) \lesssim \tau\mathbf{u} \tag{22}$$

where τ is an input-independent threshold.

Proof: Using the submultiplicative property,

$$\|d\|_\infty = \|(\hat{B}A - I)w\|_\infty \leq \|\hat{B}A - I\|_\infty\|w\|_\infty$$

and the test follows on substituting result 11. ∎

We remark that this bound on discrepancy, larger than that for `lu`, is the reason matrix inverse is numerically unstable. The factor $\|A\|\|A^{-1}\|$ is the condition number $\kappa(A)$.

We close this section with tests for Fourier transform operations. The $n \times n$ forward transform matrix W contains the Fourier basis functions; recall that $W/\sqrt{n}$ is unitary. Following convention, here the forward transform multiplies by W, while the inverse preserves scale by using $n^{-1}W^{\mathsf{T}}$.

Result 13: Let $\hat{y} = \texttt{fft}(x)$ be computed using a decimation-based fast Fourier transform algorithm; let $y = Wx$ be the infinite-precision Fourier transform. The error vector $e = \hat{y} - y$ satisfies

$$\|e\|_2 \leq 5n\log_2 n\,\|x\|_2\mathbf{u} \tag{23}$$

Proof: See [13], thm. 23.2, remembering that $\|y\|_2 = \sqrt{n}\|x\|_2$. ∎

Corollary 14: An input-independent checksum test for `fft` is

$$d = (\hat{y} - Wx)^{\mathsf{T}}w \tag{24}$$

$$|d|/(n\log_2 n\,\|x\|_2\|w\|_2) \lesssim \tau\mathbf{u} \tag{25}$$

where τ is an input-independent threshold.

Proof: This follows from result 13 after neglecting the leading constant. ∎

In this case we include the dependence on matrix size because it is known to be accurate [13, p. 468]. Using a similar result, we may also obtain bounds for `ifft`.

Corollary 15: An input-independent checksum test for `ifft` is

$$d = (\hat{x} - n^{-1}W^{\mathsf{T}}y)^{\mathsf{T}}w \tag{26}$$

$$|d|/(\log_2 n\,\|y\|_2\|w\|_2) \lesssim \tau\mathbf{u} \tag{27}$$

where τ is an input-independent threshold.

Note that scaling by n^{-1} in `ifft` removes a factor of n from the test's normalization relative to `fft`.

4 Implementing the Tests

It is straightforward to transform these results into algorithms for error detection via checksums. The principal issue is computing the desired norms efficiently from inputs to, or results of, the desired calculation. For example, in the matrix multiply, instead of computing $\|A\|\|B\|$, it is more efficient to compute $\|\hat{P}\|$ which equals $\|AB\|$ under fault-free conditions. By the submultiplicative property of norms, $\|AB\| \leq \|A\|\|B\|$, so this substitution always underestimates the upper bound on roundoff error, leading to false alarms. On the other hand, we must remember that the norm bounds are only general guides anyway. All that is needed is for $\|AB\|$ to scale as does $\|A\|\|B\|$; the unknown scale factor can be absorbed into τ.

Taking this one step farther, we might compute $\|\hat{P}w\|$ as a substitute for $\|A\|\|B\|\|w\|$. In fact, $\hat{P}w$ would often be computed anyway as a means of checking the integrity of $\hat{P}$ later, so the result check would come at only $O(n)$ additional cost. On the other hand, the simple vector norm runs an even greater risk of underestimating the bound, especially if w is nearly orthogonal to the product, so it is wise to use instead $\lambda\|w\| + \|\hat{P}w\|$ for some problem-dependent $\lambda > 0$. Extending this reasoning to the other operations yields the comparisons in table II.

The tests all proceed from the indicated difference matrix Δ by computing the norm

$$\delta = \|\Delta w\| \quad , \tag{28}$$

scaling it by a certain factor σ, and comparing to a threshold. The matrix Δ is of course never explicitly computed because it is more efficient to multiply w into the factors comprising Δ. The naive test is the un-normalized ($\sigma = 1$) comparison

$$T0: \quad \delta/\|w\| \lesssim \tau\mathbf{u} \quad . \tag{29}$$

The other tests have input-sensitive normalization as summarized in table II. First, the *ideal test*

$$T1: \quad \delta/(\sigma_1\|w\|) \lesssim \tau\mathbf{u} \tag{30}$$

is the one recommended by the theoretical error bounds of section 3, but may not be computable using the values on hand (e.g., for `inv`).

The other two tests are based on quantities computed by the algorithm and may also be suggested by the reasoning above. First, the *matrix test*

$$T2: \quad \delta/(\sigma_2\|w\|) \lesssim \tau\mathbf{u} \tag{31}$$

involves a matrix norm (except for `solve`, `fft` and `ifft`) of computed algorithm outputs. When more than one variant of the

TABLE II
Algorithms and Corresponding Checksum Tests

Algorithm	Δ	σ_1	σ_2	σ_3	Note
`mult`	$\hat{P} - AB$	$\|A\|\|B\|$	$\|\hat{P}\|$	$\|\hat{P}w\|$	—
`qr`	$\hat{Q}\hat{R} - A$	$\|A\|$	$\|\hat{Q}\hat{R}\|$	$\|Aw\|$	σ_1 easier than σ_2
`svd`	$\hat{U}\hat{D}\hat{V}^\mathsf{T} - A$	$\|A\|$	$\|\hat{U}\hat{D}\hat{V}^\mathsf{T}\|$	$\|Aw\|$	σ_1 easier than σ_2
`lu`	$\hat{P}\hat{L}\hat{U} - A$	$\|A\|$	$\|\hat{P}\hat{L}\hat{U}\|$	$\|Aw\|$	σ_1 easier than σ_2
`solve`	$(A\hat{x} - b)^\mathsf{T}$	$\|A\|\|x\|$	$\|A\|\|\hat{x}\|$	—	result is a vector
`inv`	$I - \hat{B}A$	$\|A\|\|A^{-1}\|$	$\|A\|\|\hat{B}\|$	$\|\hat{B}\|\|Aw\|$	$\|\hat{B}Aw\|$ useless
`fft`	$(\hat{y} - Wx)^\mathsf{T}$	$n\log_2(n)\|x\|$	—	—	result is a vector
`ifft`	$(\hat{x} - n^{-1}W^\mathsf{T}y)^\mathsf{T}$	$\log_2(n)\|y\|$	—	—	result is a vector

matrix test is available, we have chosen a reasonable one. The *vector test*

$$T3: \quad \delta/(\lambda\|w\| + \sigma_3) \stackrel{\geq}{<} \tau\mathbf{u} \tag{32}$$

involves a vector norm and is therefore more subject to false alarms. A major advantage of $T3$ (see the σ_3 column in table II) is that for the factorizations, it needs only $\|Aw\|$ which is very simple to compute in the typical case when algorithm inputs are already checksummed. We note that the obvious vector test for `inv` uses $\|\hat{B}Aw\|$, but since $\hat{B} = \texttt{inv}(A)$, this test becomes almost equivalent to $T0$: we suggest using the vector/matrix test shown in table II. The ideal tests $T1$ for the Fourier transforms need only the norm of the input, which is readily calculated, so other test versions are omitted.

Clearly the choice of which test to use is based on the interplay of computation time and fault-detection performance for a given population of input matrices. Because of the shortcomings of numerical analysis, we cannot predict that one test will significantly outperform another. The experimental results reported in the next section are one indicator of real performance, and may motivate more detailed analysis of test behavior.

5 Results Under Simulated Faults

We describe the simulation setup, present results for the matrix operations (`mult`, `lu`, `svd`, `qr`, and `inv`) under two input-matrix distributions, and finally show results for the Fourier transform using various probe vectors w. These simulations are intended to test the essential effectiveness of the proposed checksum technique for fault tolerance, as well as to sketch the relative behaviors of the tests described above. Due to the special nature of the populations of test matrices, and the shortcomings of the fault insertion scheme, these results should be taken as a good estimate of relative performance, and a rough estimate of ultimate absolute performance.

5.1 Experimental Setup

In essence a population of random matrices is used as input to a given computation; faults are injected in half these computations, and a checksum test is used afterward to attempt to identify the faulty computations. For the matrix operations, random test matrices A of a given condition number κ are generated by the rule

$$A = 10^\alpha U D_\kappa V^\mathsf{T} \quad . \tag{33}$$

The random matrices U and V are the orthogonal factors in the QR factorization of two square matrices with entries that are independent unit normal random variables. (They are therefore random orthogonal matrices from the Haar distribution [32].) The diagonal matrix D_κ is filled in by choosing independent uniformly distributed random singular values, rescaled such that the largest singular value is one and the smallest is $1/\kappa$. These matrices all have 2-norm equal to unity; the overall scale is set by α which is chosen uniformly at random between -8 and +8. A total of $2N$, 64×64 matrices is generated by independent draws from the rule (33). Equal numbers of matrices for each κ in $\{2^1, ..., 2^{20}\}$ are generated over the course of the $2N$ draws. Vector inputs for `fft` have no concept of condition number and they are generated by

$$v = 10^\alpha (u_1 + \sqrt{-1}u_2) \quad . \tag{34}$$

Here, α is as above, and u_1 and u_2 are vectors of $n = 64$ independent unit normal variables.

Faults are injected in half of these runs (N of $2N$ total) by changing a random bit of the algorithm's state space at a random point of execution. Specifically, the matrix algorithms when given $n \times n$ inputs generally have n stages and operate in place on an $n \times n$ matrix. The Fourier transform has $\log_2 n$ stages which operate in place on the n-length vector. So for testing, the algorithm is interrupted at a random stage, and its workspace perturbed by altering exactly one bit of the 64-bit representation of one of the matrix/vector entries a to produce a new entry, $\text{flip}(a)$. For example, `lu` consists of application of n Gauss transformations. To inject a fault into `lu`, it is interrupted between two applications of the transformation, and one entry of the working matrix is altered via flip. If the entry is used in later stages of the algorithm, then the fault will fan out across the array; if

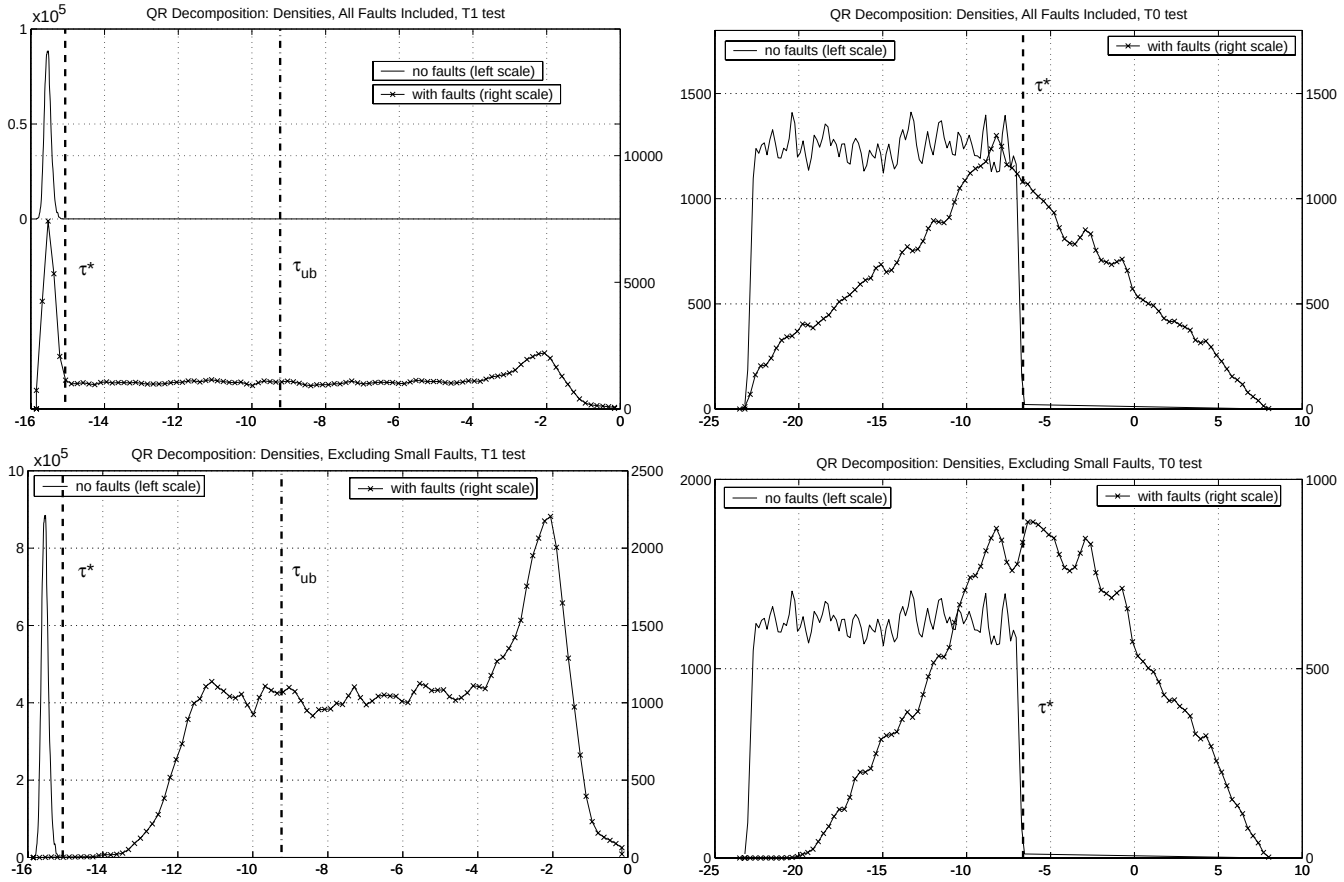

Fig. 1. Densities for `qr` decision criteria $T1$ and $T0$ under fault-free and faulty conditions. The abscissa shows the log (base 10) of the decision criterion; the ordinate is its relative frequency.

not, only one element of L or U will be affected. We modified the well-known algorithms of Press et al. [33] because of their transparency, but similar results were obtained from other implementations. For `mult`, we used the standard inner product or "ijk" algorithm; for `qr`, `inv`, `lu`, and `fft` we used `qrdcmp`, `gaussj`, `ludcmp`, and `four1`, all in double precision versions and running on IEEE-754 compliant hardware.

For each random draw of algorithm inputs, our simulation computes all four detection criteria (the left-hand sides of eqns. 29–32). This *detection criterion* is the function that a fault-detecting algorithm computes and then compares to a fixed threshold $\tau\mathbf{u}$ to decide whether to declare a fault. In our simulations, this yields two populations of detection criteria—under fault-free and faulty conditions—for each combination of test and algorithm. Each population contains N criterion values, one for each random selection of input arguments.

5.2 Results: Matrix Operations

We begin to understand how the detection criterion affects error rates by examining one test in detail. The upper-left panel of figure 1 shows probability densities of the logarithm (base 10) of the $T1$ detection criterion for `qr` under fault-free (straight line) and faulty (crosses) conditions. (In this panel only, the curves overlap and must be shown with different zero levels; in all panels the left scale is for the fault-free histogram while the right scale is the faulty histogram.) The fault-free curve is gaussian in shape, reflecting accumulated roundoff noise, but the faulty curve has criterion values spread over a wide range due to the diverse sizes of injected faults. A test attempts to separate these two populations — for a given τ, both False Alarms (roundoff errors tagged as data faults) and Detections (data faults correctly identified) will be observed. The area under the fault-free probability density curve to the right of a threshold τ equals the probability of false alarm, P_{fa}; area under the faulty curve above τ is P_d, the chance of detecting a fault.

This panel also shows τ^* (dashed line), which is defined to be the smallest tolerance resulting empirically in zero false alarms, and $\tau_{ub} > \tau^*$ (dash-dot line), the "worst-case" theoretical error bound of result 3. (We have conservatively used $\rho = 10$.) These threshold lines are labeled as described above, but appear on the log scale at the value $\log_{10}(\tau\mathbf{u})$. The point is that use of an average-case threshold enables detection of all the events lying between τ^* and τ_{ub}, while still avoiding false alarms.

Different test mechanisms deliver different histograms, and the best tests result in more separated populations. The upper-right panel shows the naive $T0$ test for `qr`. This test exhibits considerable overlap between the populations. Due to incorrect normalization, errors are spread out over a much greater range overall (about 30 orders of magnitude) and the fault-free errors are no longer a concentrated clump near $\log_{10}(\mathbf{u}) \approx -15.7$. The

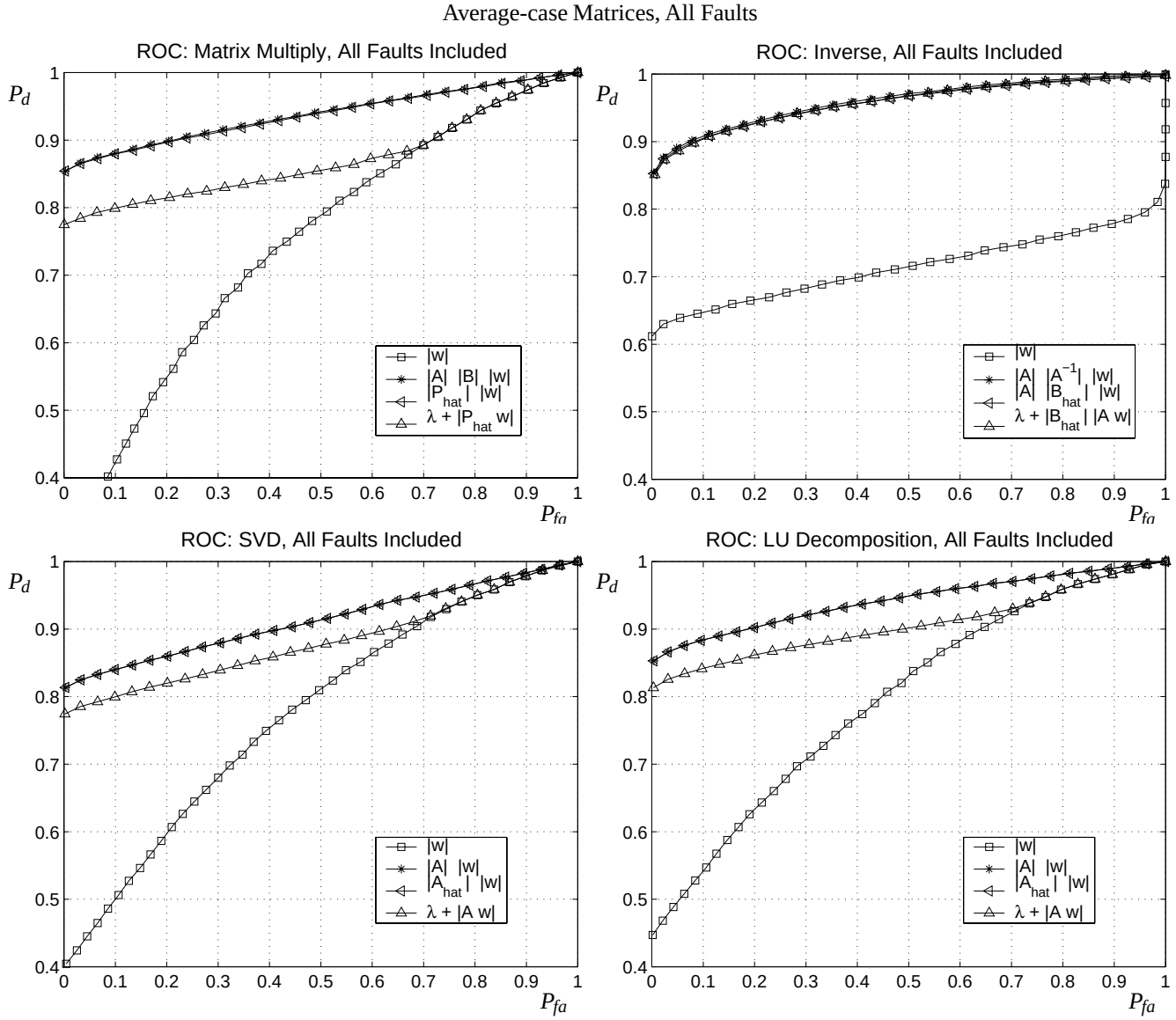

Fig. 2. ROC for random matrices of bounded condition number, including all faults.

lowest threshold which avoids false alarms (τ^*, dashed line) is now so large that it fails to detect a large proportion of faults.

Of course, some missed fault detections are worse than others since many faults occur in the low-order bits of the mantissa and cause very minor changes in the matrix element, of relative size

$$E_{rel} = |\mathrm{flip}(x) - x|/x \quad . \tag{35}$$

Accordingly, a second set of density curves is shown in the lower panels of figure 1. There, faults which cause a perturbation of size less than $E_{rel}^{min} = 10^{-10}$ are screened from the fault-containing curve (the fault-free curve remains the same). Removing these minor faults moves experiments out of the far left of the fault-containing curve: in some cases, these are experiments where the roundoff error actually dominates the error due to the fault. Again, a substantial range of experiments still exists above τ^* but below τ_{ub}. As we shall see, these plots are typical of those observed for other algorithms.

Characteristics of a test are concisely expressed using the standard receiver operating characteristic (ROC) curve. This is just a parametric plot of the pair (P_{fa}, P_d), varying τ to obtain a curve which illustrates the overall performance of the test — summarizing the essential characteristics of the interplay between densities seen in figure 1. Large τ corresponds to the corner where $P_{fa} = P_d = 0$; small τ yields $P_{fa} = P_d = 1$. Intermediate values give tests with various accuracy/coverage attributes. See figures 2 and 3. In these figures, $T0$ is the line with square markers and $T3$ is marked by upward pointing triangles; $T0$ lies below $T3$. $T2$ is shown with left pointing triangles, and $T1$, the optimal test, with asterisks; these two tests nearly coincide. We use $\lambda = 0.001$ in $T3$.

Because N independent runs are used to estimate both P_{fa} and P_d, the standard error for either is $(P(1-P)/N)^{1/2}$: the standard deviation of N averaged 0/1 variables [34, p. 107]. Thus, the standard error of an estimate $\hat{P}$ of P_d, based on N runs, may be estimated as $(\hat{P}(1-\hat{P})/N)^{1/2}$ [34, p. 131]. For figures 2 and 3 we used $N = 20000$ and curves have confidence

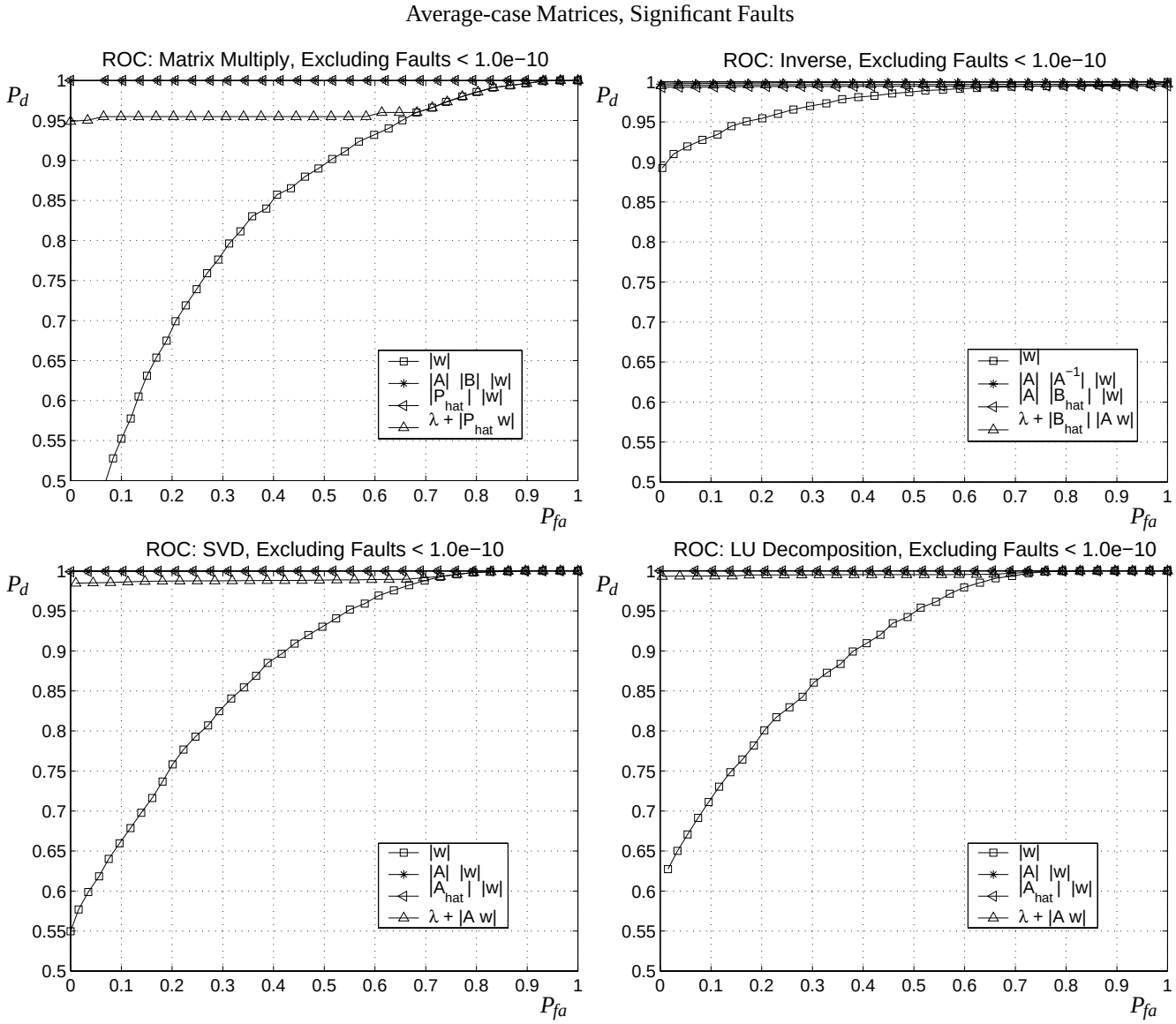

Fig. 3. ROC for random matrices of bounded condition number, excluding faults of relative size less than 10^{-10}.

bounds better than 0.005.

As foreshadowed by `qr` in figure 1, the ROCs of figure 2 show that some faults are so small they cannot be reliably identified after the fact. This is manifested by curves that slowly rise, not attaining $P_d = 1$ until $P_{fa} = 1$ as well. Therefore, we show a second group of ROCs (figure 3). In this set, faults which cause a perturbation less than $E_{rel}^{min} = 10^{-10}$, about 30% of all faults, are screened from the results entirely. This is well above the accuracy of single-precision floating point and is beyond the precision of data typically obtained by scientific experiment, for example. These ROCs are informative about final fault-detection performance in an operating regime where such a loss of precision in one number in the algorithm working state is acceptable.

We may make some general observations about the results. Clearly $T0$, the un-normalized test, fares poorly in all experiments. Indeed, for `inv`, the correct normalization factor is large and $T0$ could only detect some faults by setting an extremely low τ. This illustrates the value of the results on error propagation that form the basis for the normalized tests. Generally speaking,

$$T0 \ll T3 < T2 \approx T1 \quad . \tag{36}$$

This confirms theory, in which $T1$ is the ideal test and the others approximate it. In particular, $T1$ and $T2$ are quite similar because generally only an enormous fault can change the norm of a matrix — these cases are easy to detect. And the vector test $T3$ suffers relative to the two matrix tests, losing about 3–10% in P_d, because the single vector norm is sometimes a bad approximation to the product of matrix and vector norms used in $T1$ and $T2$. However, we found that problem-specific tuning of λ allows the performance of $T3$ to virtually duplicate that of the superior tests.

To further summarize the results, we note that the most relevant part of the ROC curve is when $P_{fa} \approx 0$; we may in fact be interested in the value P^*, defined to be P_d when $P_{fa} = 0$. This value is summarized for these experiments in table III, as the fault screen E_{rel}^{min} is varied. (Table values for $E_{rel}^{min} = 0$ and

TABLE III
P^* for four matrix operations, average-case inputs

E_{rel}^{min}	mult($T1$)	svd	lu	inv($T2$)
0	.847 (.003)	.795 (.003)	.840 (.003)	.824 (.003)
10^{-14}	.943 (.002)	.887 (.002)	.936 (.002)	.919 (.002)
10^{-13}	.987 (.001)	.941 (.002)	.984 (.001)	.968 (.001)
10^{-12}	.998 (.001)	.983 (.001)	.998 (.001)	.992 (.001)
10^{-11}	1 (.001)	.996 (.001)	1 (.001)	.999 (.001)
10^{-10}	1 (.001)	.999 (.001)	1 (.001)	1 (.001)
10^{-9}	1 (.001)	.999 (.001)	1 (.001)	1 (.001)
10^{-8}	1 (.001)	1 (.001)	1 (.001)	1 (.001)
τ^*	2.37	13.5	7.09	0.30

TABLE IV
P^* for four matrix operations, worst-case inputs

E_{rel}^{min}	mult	svd	lu	inv($T2$)	qr
0	0.570	0.630	0.605	0.423	0.621
10^{-14}	0.622	0.703	0.669	0.476	0.702
10^{-13}	0.657	0.734	0.706	0.500	0.730
10^{-12}	0.695	0.771	0.749	0.527	0.779
10^{-11}	0.723	0.809	0.780	0.552	0.811
10^{-10}	0.748	0.839	0.789	0.576	0.828
10^{-9}	0.754	0.852	0.800	0.600	0.842
10^{-8}	0.767	0.879	0.820	0.629	0.853
τ^*	10.41	47. 2	18.0	5.04	24.2

10^{-10} can in fact be read from the ROC curves, figures 2 and 3.) Standard errors, shown in parentheses, are figured based on the experiment sample size as described above. This shows that for this set of inputs, and this fault injection mechanism, 99% coverage appears for $T1$ and $T2$ at approximately $E_{rel}^{min} = 10^{-12}$; this level of performance is surely adequate for most scientific computation. Also tabulated is the threshold value τ^* at which P^* is attained; this is of course multiplied by $\mathbf{u} = 2.2 \times 10^{-16}$ when used in the threshold test. In each case the value is reported for the test *without* normalization by any leading polynomial involving matrix dimension. In each test, the $p = \infty$ norm (the maximum row sum of a matrix) was used, and the checksum vector had all ones and $\|w\|_\infty = 1$. The low τ^* demonstrates that significant cancellation of errors occurs.

Table IV is compiled from similar experiments using a worst-case matrix population of slightly perturbed versions of the Matlab "gallery" matrices. This is a worst-case population because it contains many members that are singular or near-singular (17 of the 40 gallery matrices used have condition number greater than 10^{10}), as well as members designed as counterexamples or hard test cases for various algorithms. Only $N = 800$ tests on this population were run because of the smaller population of gallery matrices: standard errors are larger, about 0.02. Note also that the choice of gallery matrices is arbitrary so the sampling bias probably dominates the standard errors in these results. In worst-case — and no practical application should be in this regime — coverage drops to about 60–90%. This gives an indication of the loss in fault-detection performance incurred by a numerically ill-posed application. Even in this case we see that significant cancellation of errors occurs, and the τ^* values do not change much.

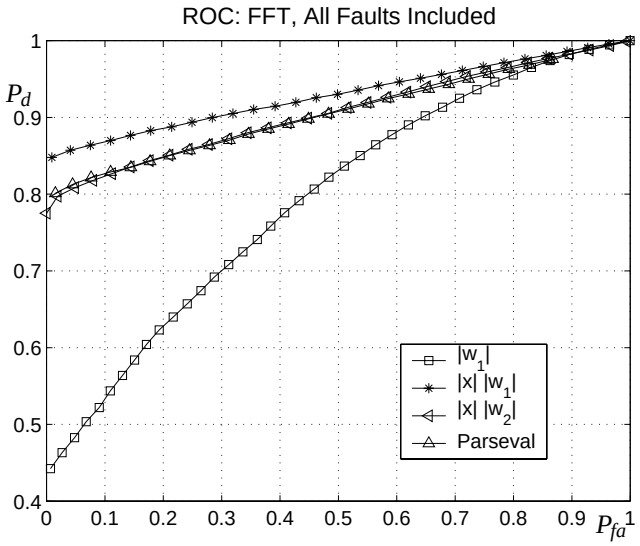

Fig. 4. ROC for fault tolerant FFT, including all faults.

5.3 Results: Fourier Transform

In related simulations we examine the performance of tests for the Fourier transform. In addition to the randomized weight vector w_1 defined at the end of sec. 2, we also used a deterministic vector w_2 with real and imaginary part both equal to

$$\cos(4\pi(k - n/2)/n), \quad k = 0, 1, ..., n-1. \tag{37}$$

This is a half-period of a cosine and it exhibits no particular symmetry in the Fourier transform. However, the ratio between the largest and smallest elements of the transform of w_2 is larger by a factor of ten than w_1. We also use the conservation of energy (Parseval) postcondition

$$y = \texttt{fft}(x) \implies \|y\|_2 = \sqrt{n}\,\|x\|_2 \tag{38}$$

to define a related test for $\hat{y} = \texttt{fft}(x)$

$$(\|x\|_2 - n^{-1/2}\|\hat{y}\|_2)/\|x\|_2 \lessgtr \tau\mathbf{u} \quad . \tag{39}$$

This *Parseval test* has been normalized to reflect scaling of the residual by the input's magnitude.

The ROC curve in figure 4 summarizes performance of these tests ($N = 20000$ samples implies standard errors better than 0.005). The naive $T0$ test has poor performance, as observed earlier. The Parseval test and the w_2 checksum test have about the same performance, but both are clearly bested by the w_1 checksum test. This ranking is repeated in table V, which shows P^* for various error screens. The w_1 checksum test is able to detect all faults larger than 10^{-11}. The large gap between τ^* and the theoretical threshold setting τ_{ub} (from result 13) illustrates again how much can be gained from an average-case outlook. For `fft` the observed scaling of the error is known, both as a function of input magnitude and input dimension, but the multiplicative constant is not.

TABLE V
P^* for three methods of FFT error checking, as fault screen is varied

E_{rel}^{min}	Checksum 1	Checksum 2	Parseval
0	.838 (.003)	.771 (.003)	.783 (.003)
10^{-14}	.931 (.002)	.855 (.003)	.869 (.003)
10^{-13}	.980 (.001)	.904 (.002)	.920 (.002)
10^{-12}	.997 (.001)	.946 (.002)	.956 (.002)
10^{-11}	1 (.001)	.975 (.002)	.970 (.001)
10^{-10}	1 (.001)	.986 (.002)	.973 (.001)
10^{-9}	1 (.001)	.988 (.002)	.974 (.001)
10^{-8}	1 (.001)	.988 (.002)	.972 (.002)
τ^*	0.076	0.053	9.85
τ_{ub}	5	5	—

6 Conclusions and Future Work

Faults within certain common computations — computations which in some cases dominate application run time — can be detected by exploiting properties their outputs must satisfy. Once detected at the subroutine level, the subroutine can be retried, or an exception raised to be caught by a higher level of the fault-tolerance system software. Following earlier work, we define tests for eight Fourier and linear-algebraic floating-point operations by checking that the computed quantities satisfy a necessary condition, implied by the form of the operation, to within a certain tolerance. Theoretical results bounding the expected roundoff error in a given computation provide tests which generally work by comparing the norm of a checksum-difference vector, scaled according to algorithm inputs, to a threshold.

For each operation, a family of readily computable tests is easy to define and implement (see table II and eqns. 29–32). The tests have different time/performance tradeoffs. The key question for a fault tolerance practitioner is to set the threshold to achieve the right tradeoff between correct fault detections and false alarms. Because of the imprecision inherent in the error bounds, theoretical results can only give an indication of how expected error scales with algorithm inputs; the precise constants for best performance must in general be determined empirically for a given algorithm implementation.

In our simulation tests, the observed behavior of these tests is in good agreement with theory. All the linear-algebraic operations tested here (`mult`, `qr`, `svd`, `lu`, and `inv`) admit tests that are effective in detecting faults larger than 10^{-10} at well above the 99% level on a broad range of matrix inputs. For factorizations, the easy-to-compute $T3$ (vector-norm) test gives performance within 1–3% of the more complex tests. The naive un-normalized test fares poorly in all tests. For `fft`, a checksum test with randomly chosen weights also performs very well, detecting all faults larger than 10^{-11} and clearly outperforming the Parseval-based test. Finally, the simulation results illustrate that conventional error bounds, if followed uncritically, can result in tolerances too high by several orders of magnitude for typical matrix inputs. More faults can be detected by using realistic thresholds.

Because the tests may be implemented as wrappers around the overall computation, they may be used with little modification in any high-performance package. For example, our implementation consisted of a set of wrappers around various ScaLAPACK and FFTW subroutines. Our choice of checksum tests was based on computational cost; for most operations we were able to perform the ideal test, but in some cases (such as `mult`) we employed an approximate test. We tested our implementation both by comparing the results with those generated by Matlab computations and, in the case of `fft`, by use of simulated fault injection at the granularity of machine instructions. In these tests we observed general agreement with theory. For details of these results, see [35].

As a final note we observe that other common subroutines, such as those involving sorting, order statistics, and numerical integration, also require more than $O(n)$ time and are candidates for fault-detecting versions. Additionally, a multiple-checksum fault-detection scheme would help to raise coverage if this is deemed necessary for some applications.

Acknowledgments

This work was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration. Thanks are due to the anonymous referees, whose comments helped the presentation and clarity of the paper. The authors also thank John Beahan, Rafi Some, Roger Lee, and Paul Springer of JPL for suggesting and commenting on parts of this work.

References

[1] M. Blum and H. Wasserman, "Reflections on the Pentium division bug," *IEEE Trans. Comput.*, vol. 45, no. 4, pp. 385–393, 1996.

[2] P. E. Dodd et al., "Single-event upset and snapback in silicon-on-insulator devices and integrated circuits," *IEEE Trans. Nuclear Science*, vol. 47, no. 6, pp. 2165–2174, 2000.

[3] M. Sullivan and R. Chillarege, "Software defects and their impact on system availability — A study of field failures in operating systems," in *Proc. FTCS-21*, 1991, pp. 2–9.

[4] S. G. Eick, T. L. Graves, A. F. Karr, J. S. Marron, and A. Mockus, "Does code decay? Assessing the evidence from change management data," *IEEE Trans. Software Engineering*, vol. 27, no. 1, pp. 1–12, 2001.

[5] M. Frigo and S. G. Johnson, "FFTW: An adaptive software architecture for the FFT," in *Proc. ICASSP*, 1998, vol. 3, pp. 1381–1384.

[6] J. Conlon, "Losing limits in space exploration," in *Insights*, pp. 22–25. NASA High Performance Computing and Communications Program, Moffett Field, CA, Nov. 1998, see also http://ree.jpl.nasa.gov.

[7] F. Chen, L. Craymer, J. Deifik, A. J. Fogel, D. S. Katz, A. G. Silliman Jr., R. R. Some, S. A. Upchurch, and K. Whisnant, "Demonstration of the REE fault-tolerant parallel-processing supercomputer for spacecraft onboard scientific data processing," in *Proc. Intl. Conf. Dependable Systems and Networks*, 2000, pp. 367–372.

[8] H.S. Stockman and J. Mather, "NGST: Seeing the first stars and galaxies form," in *Galaxy interactions at low and high redshift*. 1999, pp. 493–499, Kluwer, IAU Symposia 186.

[9] T. Murphy and R. Lyon, "NGST autonomous optical control system," Space Telescope Science Institute, 9 March 1998.

[10] L. S. Blackford et al., *ScaLAPACK Users' Guide*, SIAM, 1997.

[11] R. A. van de Geijn, P. Alpatov, G. Baker, and C. Edwards, *Using PLAPACK: Parallel Linear Algebra Package*, MIT, 1997.

[12] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Johns Hopkins, Baltimore, second edition, 1989.

[13] N. J. Higham, *Analysis and Stability of Numerical Algorithms*, SIAM, 1996.

[14] H. Wasserman and M. Blum, "Software reliability via run-time result-checking," *J. ACM*, vol. 44, no. 6, pp. 826–849, 1997.

[15] R. Freivalds, "Fast probabilistic algorithms," in *Proc. 8th Symp. Mathemat.*

Foundat. Comput. Sci., 1979, pp. 57–69, also in *Lecture Notes in Computer Science*, vol. 74, Springer.
[16] M. Blum and S. Kannan, "Designing programs that check their work," in *Proc. 21st Symp. Theor. Comput.*, 1989, pp. 86–97.
[17] M. Blum, M. Luby, and R. Rubinfeld, "Self-testing/correcting with applications to numerical problems," *J. Computer and System Sciences*, vol. 47, no. 3, pp. 549–595, 1993.
[18] K.-H. Huang and J. A. Abraham, "Algorithm-based fault tolerance for matrix operations," *IEEE Trans. Comput.*, vol. 33, no. 6, pp. 518–528, 1984.
[19] J.-Y. Jou and J. A. Abraham, "Fault-tolerant matrix arithmetic and signal processing on highly concurrent computing structures," *Proc. IEEE*, vol. 74, no. 5, pp. 732–741, 1986.
[20] F. T. Luk and H. Park, "An analysis of algorithm-based fault tolerance techniques," *J. Parallel and Dist. Comput.*, vol. 5, pp. 172–184, 1988.
[21] M. P. Connolly and P. Fitzpatrick, "Fault-tolerent QRD recursive least squares," *IEE Proc. Comput. Digit. Tech.*, vol. 143, no. 2, pp. 137–144, 1996.
[22] Y.-H. Choi and M. Malek, "A fault-tolerant FFT processor," *IEEE Trans. Comput.*, vol. 37, no. 5, pp. 617–621, 1988.
[23] S. J. Wang and N. K. Jha, "Algorithm-based fault tolerance for FFT networks," *IEEE Trans. Comput.*, vol. 43, no. 7, pp. 849–854, 1994.
[24] J. G. Silva, P. Prata, M. Rela, and H. Madeira, "Practical issues in the use of ABFT and a new failure model," in *Proc. FTCS-28*, 1998, pp. 26–35.
[25] D. L. Boley, R. P. Brent, G. H. Golub, and F. T. Luk, "Algorithmic fault tolerance using the Lanczos method," *SIAM J. Matrix Anal. Appl.*, vol. 13, no. 1, pp. 312–332, 1992.
[26] D. L. Boley and F. T. Luk, "A well-conditioned checksum scheme for algorithmic fault tolerance," *Integration, The VLSI Journal*, vol. 12, pp. 21–32, 1991.
[27] A. Roy-Chowdhury and P. Banerjee, "A new error analysis based method for tolerance computation for algorithm-based checks," *IEEE Trans. Comput.*, vol. 45, no. 2, pp. 238–243, 1996.
[28] A. Roy-Chowdhury and P. Banerjee, "Tolerance determination for algorithm-based checks using simplified error analysis techniques," in *Proc. FTCS-23*, 1993, pp. 290–298.
[29] D. Boley, G. H. Golub, S. Makar, N. Saxena, and E. J. McCluskey, "Floating point fault tolerance with backward error assertions," *IEEE Trans. Comput.*, vol. 44, no. 2, pp. 302–311, 1995.
[30] J. A. Gunnels, D. S. Katz, E. S. Quintana-Orti, and R. van de Geijn, "Fault-tolerant high-performance matrix-matrix multiplication: Theory and practice," in *Proc. Intl. Conf. Dependable Systems and Networks*, 2001, pp. 47–56.
[31] L. N. Trefethen and D. Bau, *Numerical Linear Algebra*, SIAM, 1997.
[32] G. W. Stewart, "The efficient generation of random orthogonal matrices with an application to condition estimators," *SIAM Jour. Numer. Anal.*, vol. 17, no. 3, pp. 403–409, 1980.
[33] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C*, Cambridge U., second edition, 1992.
[34] N. L. Johnson, S. Kotz, and A. W. Kemp, *Univariate Discrete Distributions*, Wiley, New York, second edition, 1991.
[35] M. Turmon, R. Granat, and D. S. Katz, "Software-implemented fault detection for high-performance space applications," in *Proc. Intl. Conf. Dependable Systems and Networks*, 2000, pp. 107–116.

Michael Turmon is a principal member of the technical staff at the Jet Propulsion Laboratory, California Institute of Technology. His research is in the theory of generalization in neural networks, applications of model-based and nonparametric statistics to image understanding, software fault tolerance, and solar physics. He received Bachelor's degrees in Computer Science and in Electrical Engineering, and subsequently an M.S. in Electrical Engineering, from Washington University, St. Louis. He obtained his Ph.D in Electrical Engineering from Cornell University in 1995. Michael was a National Science Foundation Graduate Fellow (1987–90), won the NASA Exceptional Achievement Medal for his work on pattern recognition in solar imagery (1999), and received a Presidential Early Career Award for Scientists and Engineers (PECASE) in 2000. He has been co-investigator on the NASA/ESA SOHO spacecraft, and is co-investigator on the CNES Picard mission. Michael is a member of the IEEE (Computer and Information Theory Societies) and the Institute for Mathematical Statistics.

Robert Granat is a Senior Member of the Technical Staff in the Data Understanding Systems Group at the Jet Propulsion Laboratory. He received his B.S. in Engineering and Applied Science from the California Institute of Technology, and his M.S. in Electrical Engineering from the University of California, Los Angeles in 1998. Robert's areas of research are large scale scientific computing, software fault tolerance, statistical pattern recognition, and tomographic imaging. He is a member of the IEEE.

Daniel S. Katz is supervisor of the Parallel Applications Technologies group within the Exploration Systems Autonomy section at the Jet Propulsion Laboratory, which he joined in 1996. He was the Applications Project Element Manager for the Remote Exploration and Experimentation Project while this work was performed. From 1993 to 1996 he was employed by Cray Research (and later by Silicon Graphics) as a Computational Scientist on-site at JPL and Caltech. His research interests include: numerical methods, algorithms, and programming applied to supercomputing, parallel computing, and cluster computing; fault-tolerant computing; and computational methods in both electromagnetic wave propagation and geophysics. He received his B.S., M.S., and Ph.D degrees in Electrical Engineering from Northwestern University, Evanston, Illinois, in 1988, 1990, and 1994, respectively. His work is documented in numerous book chapters, journal and conference publications, and NASA Tech Briefs. He is a senior member of the IEEE, designed and maintained (until 2001) the original website for the IEEE Antenna and Propagation Society, and serves on the IEEE Task Force on Cluster Computing's Advisory Committee.

John Z. Lou received his B.S. in applied mathematics and M.S. in engineering mechanics from Shanghai Jiao-Tong University in 1982 and 1985, respectively. He received his Ph.D in computational mathematics from the University of California at Berkeley in 1991. During 1991–1993, he worked as a staff scientist at the Naval Command, Control and Ocean Surveillance Center in San Diego, California. He is now a senior staff member at the Jet Propulsion Laboratory, California Institute of Technology. His research interests include numerical modeling of space and earth science applications, advanced computing and software technologies. His work includes the development of algorithms and software in those areas.