Speed Testing of Attitude Estimation Algorithms[1]

Yang Cheng[2] and Malcolm D. Shuster[3]

...My grief shall be with speed redressed.
William Shakespeare
King Henry IV, Part II, Act IV, scene ii

Abstract

The speed of solutions of the Wahba problem, in particular, QUEST, FOAM, ESOQ, ESOQ2, Davenport's original q-method, and Markley's SVD algorithm, are examined. The algorithms are tested for the number of MATLAB® floating-pont operations (flops), MATLAB® execution times, MATLAB® C-mex file execution times, and the execution times of stand-alone compiled C programs. Any of the six algorithms examined here would be excellent candidates (on the basis of speed) for implementation in real missions. A careful account is presented of the problems besetting tests of flop counts and execution times, especially within MATLAB®. The floating-point operation requirements and execution times of all algorithms are given careful scrutiny.

Introduction

The speed of attitude estimation algorithms has been a topic of interest for many years. The need for ever greater speed for attitude computation, even for ground-based computations, became critical in the late 1970s and even the early 1980s and was the reason for the creation of the QUEST algorithm [2–4]. Algorithm speed has remained a topic of interest for on-board attitude determination systems, although it has been becoming less so as the speed and the number of microprocessors onboard continues to increase, and the attitude estimation algorithm becomes a smaller portion of the on-board software. In

[1]This work is a revision and abridgement of an earlier conference article [1].
[2]Research Assistant Professor, Department of Mechanical and Aerospace Engineering, University at Buffalo, State University of New York, Amherst, New York, 14260. email: cheng3@buffalo.edu. website: www.acsu.buffalo.edu/~cheng3.
[3]Director of Research, Acme Spacecraft Company, 13017 Wisteria Drive, Box 328, Germantown, Maryland 20874. email: mdshuster@comcast.net. website: http://home.comcast.net/~mdshuster.

the present work, we examine a wide variety of tests: (1) the number of MATLAB® floating-point operations (flops), more properly called “efficiency” rather than “speed,” (2) the execution times of MATLAB® implementations, (3) the execution times of MATLAB® C-mex files; and finally, (4) the execution times of stand-alone precompiled programs in the C language. We will discuss the characteristics of each test and the difficulties inherent in them.

We will test the fastest attitude estimation algorithms. These are all solutions of the Wahba problem [5], proposed in 1965. All of these algorithms seek the attitude which minimizes the cost function

$$J_A(A) = \frac{1}{2} \sum_{k=1}^{n} a_k \, |\hat{\mathsf{W}}_k - A\hat{\mathsf{V}}_k|^2$$

Here, A is the attitude[3] matrix (direction-cosine matrix), $\hat{\mathsf{W}}_k$, $k = 1, \ldots, N$, are the measured directions as observed in the spacecraft body frame, and $\hat{\mathsf{V}}_k$, $k = 1, \ldots, N$, are the corresponding reference directions in the primary reference frame, assumed to be noise-free.[4] The weights, a_k, $k = 1, \ldots, n$, are inversely proportional to the typical variances of the measurements and make the Wahba problem a maximum-likelihood estimation problem [7].

The algorithms of our study are QUEST, FOAM, ESOQ, ESOQ2, Davenport’s original q-algorithm, and Markley’s SVD algorithm,[5]

The Solutions to the Wahba Problem

Solutions to the Wahba problem fall into two classes. First, there are solutions which solve for the optimal attitude by applying one of the standard algorithms of Numerical Linear Algebra [8] to the attitude profile matrix B or the Davenport matrix K. These algorithms are described in detail in references [9] and [10] and their description need not be repeated here.[6] Examples of such methods are the polar-decomposition method of Farrell and Stuelpnagel [10, 12], the matrix square-root algorithms of Wessner [10, 13] and Brock [10, 14], the original q-method of Davenport [2, 3, 10, 15, 16], and Markley’s SVD algorithm [10, 17], which last is very closely related to Farrell and Stuelpnagel’s polar-decomposition algorithm in concept. We call these the *non-iterative solutions*, a misnomer since, for example, the algorithms of Numerical Linear Algebra used in Davenport’s

[3]Unless otherwise noted, *attitude* in this work will always mean three-axis attitude.

[4]We have used bold sans-serif characters to represent the actual measurement values, since we shall not need to describe random variables in this work. Our notation follows generally that of reference [6].

[5]When Markley’s name does not qualify the name of his algorithm, it will be cited as *M-SVD*, to avoid confusion with the general SVD method of Numerical Linear Algebra [8]. Likewise, Davenport’s original implementation of the q-method using Householder’s method will be cited as *q-Davenport*, since QUEST and the Mortari algorithms (ESOQ, ESOQ2) of the many algorithms based on the Wahba problem, all of which we shall show to be excellent candidates for mission software. We shall compare our results with those of two earlier studies [9, 10].

[6]Reference [10] provides a masterful overview of the Wahba problem. A brief summary of our criticisms of the numerical results of references [9] and [10] can be found in the section “Alternatives to QUEST,” appearing on pages 678–679 of reference [4]. A discussion of the robustness of QUEST can be found in reference [11].

orignal q-method and Markley's SVD algorithms are also iterative, but they are not called iteratively within the attitude determination algorithm.

The other solution methods first determine $\lambda_{\rm max}$, the maximum achievable weighted overlap of the measured directions and the rotated reference directions, which is the maximum characteristic value of the Davenport matrix K [2, 3, 10, 15, 16], by some method especially tailored to the Wahba problem and then, knowing $\lambda_{\rm max}$, solve the simpler problem of constructing the optimal quaternion $\bar{q}^*$ or the optimal direction-cosine matrix A^* from K or B. These are the *fast iterative solutions* to the Wahba problem. The first of these was QUEST [2, 3, 10], followed a decade later by FOAM [10, 18] and finally by Mortari's many Euler and ESOQ algorithms [9, 10, 19–23].[7] Of the Mortari algorithms, the top-of-the-line algorithm is supposedly ESOQ2 [10, 22, 23]. The algorithms of this second group solve for $\lambda_{\rm max}$ either by using a zero-th-order approximation $\lambda_{\rm o}$, easily calculated from the measurement weights, or by applying the Newton-Raphson method to the overlap characteristic polynomial, using $\lambda_{\rm o}$ as an initial value.[8] This procedure was introduced in reference [2] and has been copied by all later fast iterative algorithms. The QUEST algorithm uses the QUEST form of the characteristic polynomial;[9] the other algorithms now use the FOAM form.[10] In infinitely precise arithmetic the two forms are exactly equivalent.

The two classes of algorithms have different numerical qualities. The non-iterative algorithms, which rely on general-purpose numerically stable library routines, are extremely stable, because every conceivable special case has already been treated in the standard library routine. This means, however, that they carry for us a lot of excess baggage which is not needed for the problem at hand (the Wahba problem). As a result they are slower than an algorithm specifically tailored to the needs of the Wahba problem. Davenport's original implementation of the q-method, which employed a library routine for Householder's method for finding the characteristic vectors and characteristic values of a real-symmetric matrix, was burdened with an algorithm of extreme complexity, whose computational burden may vary from library to library. For example, there are 16 different code paths underlying the MATLAB® (LAPACK [24]) *eig* function, depending on the nature of the arguments [25]. A customized implementation that directly calls the appropriate low-level routine beneath the MATLAB® *eig* interface would improve the execution time of Davenport original q-method, because the K matrix is always real-symmetric and 4×4. Likewise, the execution time would be smaller if one could design the program to find only the largest characteristic value and associated characteristic vector. Similar arguments hold for the implementation of a general all purpose SVD algorithm (MATLAB® *svd* [26]) in Markley's SVD

[7]ESOQ1.1 and ESOQ2.1, which were examined in reference [1], are actually the invention of Markley and appear for the first time in references [9] and [10].

[8]As noted frequently since 1978 [2], The zero-th approximation $\lambda_{\rm o}$ of the maximum overlap provides all the useable accuracy needed for the attitude estimate. For attitude data of accuracy 10 arcsec/axis or better, a single Newton-Raphson iteration of the characteristic polynomial for the overlap will yield a value for $\lambda_{\rm max}$ whose precision exhausts that of an IEEE double-precision floating-point number (64 bits or approximately 16 significant digits [2, 3].

[9]In this work, the QUEST characteristic polynomial will always be in the traditional expanded form [11], which does not affect the speed.

[10]At one point, the ESOQ algorithm used essentially the QUEST form of the characteristic equation [9].

method. The iterative algorithms, because they use custom-tailored numerical algorithms, are faster.

The construction of the B or K matrix is the same for all algorithms.[11] Differences in efficiency or in execution times of the various fast algorithms depend on the computational requirements for computing the coefficients of the characteristic polynomial and the construction of the optimal attitude from λ_{max} and B (equivalently, from λ_{max} and K). The computation of the coefficients for the QUEST characteristic polynomial requires more floating-point operations than that of the corresponding coefficients of the FOAM characteristic polynomial. At the same time, the construction of the QUEST quaternion is made more efficient by using intermediate quantities computed for the construction of the QUEST characteristic polynomial. As a result, in terms of flop counts, we anticipate QUEST to have the advantage if the Newton-Raphson sequence of λ_{max} must be computed, but possibly not so if only the zero-th-order λ_{o} is used. This fact has been known for a very long time. This relative disadvantage for QUEST using λ_{o}, as we shall see, appears only for MATLAB® flop counts.

In all of the counts of floating-point operations (flop counts) or timing tests presented here, only minimal versions of the attitude estimation algorithms were tested, as in references [9] and [10]. Missing from the tests, as in references [9] and [10], are the examination of the computational burden due to low-level data checking, data adjustment, data validity tests (in QUEST, for example, the TASTE test [27, 28]), observability tests, the computation of the attitude covariance matrix, or the method of sequential rotations [2, 3, 10], none of which need be the same for each algorithm. The timing tests of references [9] and [10] and this work are, thus, somewhat unrealistic in nature and might never have been studied by the present authors, had references [9] and [10] not established a precedent. In every case, we have insisted that the final output be the quaternion, thus placing a small extra burden on the M-SVD and FOAM algorithms. These algorithms, however, are somewhat slower than the others in any event. Since it is hard to imagine an attitude determination system in which the direction-cosine matrix will not also be needed, and the quaternion will always be needed for archiving results, perhaps, it would have been more democratic to insist that all algorithms output both representations. However, we have not done this.

The Efficiency and Speed Tests

In the present work we have performed four different speed tests:

- MATLAB® flop counts
- MATLAB® execution times
- execution times of MATLAB® C-mex files
- execution times of stand-alone C-language compilations

[11]M-SVD and FOAM do not calculate K, which, however, requires no floating-point multiplications or divisions to be calculated from B.

Of these, only the last is pertinent to execution times of mission software. The first three are presented largely, because references [9] and [10] relied solely on MATLAB® flop counts for the computations. As we shall see, MATLAB® flop counts and even MATLAB® execution times are poor criteria for judging the speed for the Wahba algorithms.

The computer platform in all tests was a Dell Precision PWS 380 desktop personal computer embodying an Intel® Pentium® D CPU with clock frequency 3 GHz and with 2 GB of random access memory. The operating system was Microsoft Windows® XP Professional, version 2002, with service pack 2.

We have generated test data consisting of one thousand randomly generated frames of data, each consisting of 2, 4, and 25 measurements, and tested the algorithms for up to three Newton-Raphson iterations in the computation of $\lambda_{\max}$.[12] The test samples for 2 and 4 measurements were obvious subsets of the 25-measurement samples. We have made certain that in each frame, at least two observation vectors were separated by an angle of at least 0.1 rad and that the angle of rotation of the attitude used to generate the reference vectors from the observation vectors lay between 0.1 rad and $\pi - 0.1$ rad, so that there would be no need to perform the method of sequential rotations [2, 3, 10], In order that the values of the directions measurements be the same for our analysis, measurement noise was applied to the *reference* directions by adding a Gaussian random error of mean zero and standard deviation $\sigma = 6$ arcsec to each component of the reference direction and then renormalizing the vector. These noisy reference vectors were then transformed to the body frame to create the observation vectors. In all tests for n measurements, the inputs were two data matrices (one for the observed directions and one for the reference directions), each of dimension $3 \times 1000\,n$, where $n = 2, 4,$ or 25. When determining execution times, these 1000 samples have been repeated 5,000 times to reduce round-off errors. (One assumes by a factor $1/\sqrt{5000} \approx 1/70$.)

The MATLAB® m-files were written in the MATLAB® script language and executed in MATLAB®. The MATLAB® C-mex files were written in C, compiled into dynamically-linked library files, and executed in MATLAB®. The stand-alone C executables were written in C and executed independently of MATLAB® as a Windows console program. MATLAB® 6.5, which incorporates LAPACK and the JIT-accelerator, is used for most of the MATLAB®-related experiments. MATLAB® 5.3, an earlier version, is used for flop counting. The C source code for the C-mex files and for the stand-alone executables was compiled using Microsoft Visual Studio .Net 2003 (known also as Visual C++ 7.1). More extensive tests were performed in Reference [1].

In our efficiency and speed tests, we have examined the iterative algorithms: QUEST, FOAM, ESOQ, and ESOQ2. The QUEST software has been the same as used in references [9] and [10]. The non-iterative algorithms examined are Davenport's original q-method and Markley's SVD algorithm.[13]

[12]Reference [1] presents also the case of three measurements and up to five iterations in the computation of $\lambda_{\max}$.

[13]Reference [1] examined a wider variety of algorithms, some of them hybrid algorithms developed for those tests. These are not presented here, and the reader should refer to Reference [1] for those results. The additional algorithms included ESOQ1.1 and ESOQ2.1, which are not of equal interest, because these algorithms are only first-order approximations, that is, equivalent to one Newton-Raphson iteration. Reference [1] examined also an alternate version of QUEST (called

MATLAB® flops!

The MATLAB® *flops* function has been obsolete since the incorporation of LAPACK (Linear Algebra PACKage), the modern replacement for LINPACK and EISPACK, in MATLAB® 6, because most of the floating-point operations are now performed in optimized BLAS (Basic Linear Algebra Subprograms) that do not record flop counts [25]. Also, not only may flops not have been counted consistently in previous MATLAB® releases, but non-floating-point operations, such as logical expressions, are ignored obviously in flop counting. More importantly, for modern computer architectures, floating-point operations may not be the dominating factor in execution time. Memory reference and cache usage may be more important [25]. Flop counts for this work have been obtained using MATLAB® version 5.3.

MATLAB® often does not count flops realistically for calls to precompiled utilities (like those for the singular-value decomposition and the spectral decomposition used here). In these cases, not all floating-point operations may be counted. Furthermore, MATLAB® counts additions, subtractions, multiplications and divisions as equal floating-point operations, although for IEEE double-precision numbers a multiplication or division requires more than 50 times as many binary operations as do an addition or subtraction. Hand-counting may be more reliable, but this may be difficult for non-iterative algorithms such as q-Davenport and M-SVD, in which the number of flops will depend on the values of the inputs. Calls to special functions of a single variable generally require eight multiplications, because such calculations are usually performed using an optimized eighth-order polynomial [29].

As an example of the discrepancies in flop counts in MATLAB®, we note that MATLAB® function *det* computes the determinant from the triangular factors of a matrix obtained by Gaussian elimination (the LU decomposition) [30]. This factorization based approach is generally more robust and more accurate than the naive calculation of the determinant using the usual expansion expression. The naive computation of the determinant of a 3×3 matrix requires 12 flops while, according to the MATLAB® *flops* function, an external call to *det* requires 13. The MATLAB® *det* function, however, because it calculates the determinant by performing an LU factorization, requires many more flops than 13. Thus, the discrepancy in the number of flops is greatly underestimated by MATLAB®. In fact, we find that the MATLAB® flop count for the determinant is always equal to that for the LU decomposition of the same matrix. Because the MATLAB®

simply QUEST there) which differed from QUEST here by the fact that it computed a determinant explicitly rather than by a call to a determinant function. This is, in fact, the method of the 1979 NASA code. The differences between the NASA QUEST and the and QUEST of references [9] and [10] are small and do not effect the speed comparisons of the algorithms. OSOQ1 and OSOQ2 in reference [1] were our optimized versions of ESOQ and ESOQ2, which were faster than the ESOQ and ESOQ2 of references [9] and [10], and sometimes faster than QUEST. QUESTOQ1 and QUESTOQ2 are variations of ESOQ and ESOQ2 in which the QUEST characteristic polynomial is used for the computation of the maximum overlap. Their speed tended to be intermediate between the corresponding ESOQ and OSOQ algorithms. We will not report the results here for any of these variant or hybrid algorithms. The reader, once more, is directed to reference [1]. We point out that the speed differences between minor variants in reference [1] do not always appear to be consistent or intuitive.

TABLE 1 MATLAB® Floating-Point Operation Counts (Flops)

Iterations	FOAM	QUEST	ESOQ	ESOQ2	q-Davenport	M-SVD
2 Measurements						
exact	276	179	246	238		
0	257	146	116	114	715	523
1	292	183	257	254		
2	303	194	269	265		
3	314	205	281	276		
4 Measurements						
0	299	188	158	156	771	853
1	334	225	299	296		
2	345	236	311	307		
3	356	247	323	318		
25 Measurements						
0	740	629	599	597	1189	1347
1	775	666	740	737		
2	786	677	752	748		
3	797	688	764	759		

det function has been precompiled, however, the complexity of *det* is hidden somewhat by its high speed in timing tests.

Numerical Results: MATLAB® Flop Counts

Table 1 shows our results for flop counts. "Exact" refers to the computation of $\lambda_{\max}$ from an exact algebraic formula [3], which exists for the two-measurement case. References [9] and [10] found that for zero iterations, the ESOQ and ESOQ2 algorithms were most efficient, which is borne out also by our results here. We note also that in all other cases, it is QUEST which is the most efficient. In missions where star trackers are the principal attitude sensor, one will wish to execute exactly one Newton-Raphson iteration in order to exhaust the resolution of IEEE double precision floating-point numbers. Thus, QUEST will be more efficient with even greater frequency.

Table 1 shows also the results for the non-iterative algorithms.[14] It is interesting to note that the number of flops required for q-Davenport and M-SVD differs less from that for the iterative algorithms as the number of measurements increases, indicating that for a very large number of measurements the flops devoted to computing B and K overwhelm the other operations, as one would expect. However, as we have said, MATLAB® does not count flops realistically for

[14]Because Householder's method and the SVD algorithm, although iterative internally, are not called iteratively within the q-Davenport and M-SVD attitude estimation algorithms, we have placed then on the line of zero iterations in the tables.

TABLE 2 MATLAB® Execution Times (μs) (JIT-Accelerator off)

Iterations	FOAM	QUEST	ESOQ	ESOQ2	q-Davenport	M-SVD
2 Measurements						
exact	241	222	353	343		
0	219	175	230	195	143	208
1	255	230	371	359		
2	264	241	380	368		
3	273	253	389	378		
4 Measurements						
0	221	177	231	196	142	213
1	261	235	376	364		
2	270	248	386	373		
3	278	259	395	382		
25 Measurements						
0	229	187	240	207	151	223
1	271	244	385	373		
2	280	256	394	383		
3	289	269	403	392		

library functions such as *eig* and *svd*, which, naturally, figure prominently in the MATLAB® implementations of q-Davenport and M-SVD.

Numerical Results: MATLAB® Execution Times

Beginning with version 6.5, MATLAB® has incorporated the JIT (just-in-time) accelerator [31] in its system. This greatly lessens the execution time of MATLAB® programs but not necessarily consistently. Therefore, all tests of MATLAB® execution times have been performed with the JIT accelerator turned off. The exact origin of the inconsistencies in timing tests may be very complex.

There are two timing functions in MATLAB®, *cputime* and *tic/toc*. The first records only actual CPU time, while the second measures actual time elapsed ("wall-clock" time) and is, therefore, more sensitive to properties specific to the platform. The results reported here for MATLAB® execution times were all determined using *cputime* (with the JIT accelerator off). This was also the configuration for measuring execution time for C-mex files (see below).

The MATLAB® JIT-accelerator reduces interpreter and data-handling overhead [31] by converting many p-code[15] instructions into native machine instructions. The native machine instructions suffer no interpreter overhead and, therefore, run very quickly. By default, the state of the JIT-accelerator is on. The action of the JIT-accelerator is rather complicated and sometimes seemingly

[15]The p-code was the linear stream of instructions converted from the MATLAB® code and executed by the MATLAB® interpreter in earlier versions.

counter-intuitive. For example, when the JIT-accelerator is on, the function call for the determinant function is slower than the explicit calculation of the determinant, at least in MATLAB® 6.5. When the JIT-accelerator is off, the opposite is true. We have executed our tests with the JIT-accelerator off in order to better simulate the performance of the compiled programs more likely to be used in mission support. Differences in MATLAB® overhead may also play a role.

Table 2 shows the timing results for the execution of the MATLAB® m-files. We note immediately that QUEST is the fastest iterative algorithm in all cases. The differences in speed among the different iterative algorithms were not great, just as the differences in flop counts were not great. Particularly noteworthy is that the speed of the q-Davenport and often M-SVD algorithms exceeds that of QUEST, which shows the unreliability of MATLAB® as a laboratory for speed tests, since these last two algorithms impose a significantly greater computational burden. The greater speed of q-Davenport and M-SVD in this circumstance is due to the fact that much of the code of the MATLAB® *eig* and *svd* m-files has been precompiled from C code.

Numerical Results: MATLAB® C-mex Files

For the C-mex files and stand-alone executable programs have been examined for execution times in the C environment of Microsoft Visual Studio .Net 2003 using the LAPACK routines for characteristic value decomposition and singular-value decomposition taken from the MATLAB® mathematics library (labeled there *eig* and *svd*).

It is obvious from the relatively small difference in the execution times of Table 3 that these execution times are dominated by the MATLAB® overhead. The execution times for the C-mex files are approximately an order of magnitude smaller than those for the m-files. QUEST is seen to be the faster algorithm only for about half of the cases. The difference between the execution times for the iterative and non-iterative algorithms is small, but only because of the excessive overhead of MATLAB® involved in the C-mex files.[16]

Numerical Results: Stand-alone Executable Programs

For stand-alone executables, as shown in Table 4, the timing function (Windows application programming interface, API) *timeGetTime* was used to measure elapsed time under Windows XP. We see that QUEST is fastest in almost every case.[17] As one gets closer to real mission software, the differences in the execution times becomes less important.[18]

[16]Execution times for the C-mex files created using the GNU Compiler Collection are somewhat faster than those created using Microsoft Visual C++.

[17]In reference [1] we found that Microsoft Visual C ++ favored OSOQ1, our optimized ESOQ, while the GNU Compiler Collection favored QUEST.

[18]We have not tested stand-alone compiled C programs for ESOQ1.1 and ESOQ2.1. In terms of MATLAB® execution times, these two algorithms were slower than QUEST. We anticipate that as stand-alone compiled C programs, they will be closely comparable to the other fast algorithms, but, there is no real need for these algorithms, because they are only limited versions of ESOQ and ESOQ2, even if the limitations are not practical limitations.

TABLE 3. Execution Times (μs) for C-mex Files of the Fast Optimal Attitude Estimators within MATLAB®

Iterations	FOAM	QUEST	ESOQ	ESOQ2	q-Davenport	M-SVD
2 Measurements						
exact	32.720	31.500	32.686	31.092		
0	31.186	31.220	31.812	31.406	72.97	69.38
1	33.218	32.374	33.030	33.844		
2	30.156	32.218	32.968	33.156		
3	32.032	32.186	33.218	33.250		
4 Measurements						
0	30.844	31.968	30.250	31.124	71.88	75.47
1	33.438	33.686	32.064	31.406		
2	32.812	33.532	32.688	31.156		
3	32.750	33.342	32.374	31.032		
25 Measurements						
0	35.562	34.438	34.250	34.438	75.00	78.43
1	35.094	34.594	35.688	34.562		
2	34.968	35.188	35.436	34.686		
3	35.406	35.092	35.500	34.688		

Note that in the most realistic situations, ESOQ, the Mortari algorithm which is closest to QUEST, is faster than ESOQ2. It is noteworthy that q-Davenport is only about 30 times slower than QUEST for large amounts of data.[19]

Comparison with the Results of Markley and Mortari

References [9] and [10] are very similar. They differ in that: (1) the ESOQ algorithm of reference [9] uses essentially the QUEST form of the characteristic polynomial, while that of reference [10] uses the FOAM form; and (2) reference [10] reports the data differently from reference [9]. Reference [9] executed the same three test scenarios as reference [10] and contained three tables and five figures analogous to those in reference [10]. Figures 4 and 5 of references [9] and [10] are concerned with speed and are examined here.[20]

Figures 4 and 5 of references [9] and [10] display test results for the algorithm for the case where there are three measurements with equal accuracies of 1 arcsec and are modeled correctly. The geometry of the measurements is similar to that of scenario 2 of references [9] and [10], in which the three direction measurements are coplanar with angular separations of about 175.7 deg between the central and outer vectors.

[19]In other tests in reference [1], but not reported here, the factor was as large as 70.

[20]Again, a brief summary of our criticisms of references [9] and [10] can be found in the section "Alternatives to QUEST," appearing on pages 678–679 of reference [4].

TABLE 4. Execution Times (μs) for Stand-Alone Executable Versions

Iterations	FOAM	QUEST	ESOQ	ESOQ2	q-Davenport	M-SVD
2 Measurements						
exact	0.447	0.350	0.414	0.472		
0	0.382	0.263	0.250	0.303	34.64	30.51
1	0.429	0.337	0.426	0.479		
2	0.465	0.369	0.459	0.510		
3	0.500	0.403	0.496	0.552		
4 Measurements						
0	0.419	0.306	0.295	0.349	35.08	37.29
1	0.478	0.373	0.471	0.548		
2	0.514	0.404	0.503	0.561		
3	0.549	0.437	0.548	0.593		
25 Measurements						
0	1.144	1.021	1.003	1.072	35.68	39.10
1	1.209	1.099	1.168	1.266		
2	1.246	1.131	1.204	1.280		
3	1.285	1.158	1.246	1.315		

Figure 4 of reference [9] presents the number of flops required for each algorithm to exhaust double-precision numerical accuracy of the attitude estimate as a function of the number of measurements.[21] For these data, this means universally one Newton-Raphson iteration for FOAM, ESOQ, ESOQ2, and QUEST. In Figure 4 of reference [9], we see that the QUEST requires the smallest number of flops, as found in the present work. The identical numerical results are depicted in reference [10], but the QUEST results are on a different graph. As a result, it is less obvious in reference [10] that the QUEST algorithm is more efficient than the other fast algorithms.

Figure 5 of reference [9] shows the flop counts for each algorithm as a function of the number of measurements for zero and one iteration of the Newton-Raphson method. These are shown for FOAM, ESOQ, ESOQ2 and QUEST. The number in parentheses in Figure 5 is the number of Newton-Raphson iterations performed. As expected from Table 1 of this work, the flop counts for QUEST(0) are larger than those for ESOQ(0) and ESOQ2(0), but the flop counts for QUEST(1) are smaller than those for all the other iterated algorithms. The curves for FOAM(0) and FOAM(1) are higher than any other curves, also in agreement with Table 1 of the present work. In reference [10], however, the curves for FOAM(0), FOAM(1), ESOQ(1), and ESOQ2(1) have been omitted, with the result that

[21]On the effect of the different choices of the characteristic polynomial on the timing tests of the ESOQ algorithm, see our comments for the QUEST algorithm in the penultimate paragraph of the section “The Solutions of the Wahba Problem.”

QUEST(1) appears from Figure 5 of reference [10] to be the slowest algorithm, although generally, as we have seen in our Table 1 and from reference [9], QUEST is the most efficient algorithm of the six algorithms examined in this article for one or more iterations. The separation in reference [10] of the QUEST results from those of the other fast algorithms makes this greater efficiency of QUEST difficult to see.

Discussion and Conclusions

In efficiency and speed, the subject of the present work, we have seen that MATLAB® greatly exaggerates the differences between the algorithms. QUEST, as we have seen, is most frequently the most efficient algorithm and always the fastest algorithm in MATLAB®. The large overhead of the C-mex files, in general, make the speed differences between the iterative algorithms unimportant. For stand-alone implementations in C on an IBM-compatible PC running Windows XP, the closest we can come to testing the various algorithms in a realistic mission environment, we see that the differences in execution times become very small.[22] The difference is dependent also on the choice of platform, the operating system, and what other programs are being executed simultaneously. From the standpoint of implementation in a real mission, there is no significant advantage in speed of one fast implementation of a solution to the Wahba problem over another. Whatever the "speed" differences in the algorithm, and whatever petty advantage one algorithm may have over another in a given context, the differences are small to the point of unimportance. As stated in reference [4] the best thing to say is that QUEST, FOAM, ESOQ, and ESOQ2 are all in the same speed class.

The reason for the separation of the speed results for QUEST in reference [10] from those of the other fast algorithms in this study is the claim in reference [10] that QUEST is not robust or accurate unlike the fast algorithms developed by the authors of references [9] and [10]. Reference [11] has showed that, by a simple rearrangement of terms in the QUEST characteristic polynomial, the claim of diminished robustness for QUEST is not true, not even for the extreme test scenario 2 of references [9] and [10]. At the same time it should be noted that the scenario used for this robustness test leads to estimation errors so large *for any attitude estimation algorithm* that the attitude solutions might create a danger for attitude control and be useless for the annotation of scientific data. It is hard to imagine a mission in which one would require arc-second attitude estimation accuracy for the direction of one axis but tolerate three-sigma attitude estimation errors about that axis of ± 30 deg. A well-designed attitude determination system would have tests to detect such extreme cases and suppress operations based on them.[23] Furthermore, there is no requirement for QUEST to perform any Newton-Raphson iterations to calculate the maximum overlap eigenvalue $\lambda_{\max}$ [2, 3, 11], and, in fact, the computation of $\lambda_{\max}$ will be sufficiently accurate in scenario 2 of references [9] and [10] if no Newton-Raphson iterations are performed at all, as shown even by those references. The unstated

[22]The greatest dependence, as shown in reference [1], was on the choice of compiler and mathematics library.

[23]The original QUEST software, in fact, had an internal test to detect pernicious cases like that of the extreme test scenario of references [9] and [10]. That test was not documented in reference [2] or [3]. The QUEST validation tests will be presented in an article currently in preparation [32].

purpose of including the Newton-Raphson iterations in QUEST originally was not to calculate $\lambda_{\max}$ but a test parameter TASTE. Last but not least, the poor performance reported in references [9] and [10] is only for a version of the QUEST algorithms especially created for the tests of those references.

In 1978 using the EISPACK [26] library function *EIGRS* for Householder's method written in FORTRAN IV and executed using the G compiler on an IBM-360 Model-75 mainframe computer running under IBM OS 360, Davenport's q-Method was 1000 times longer in execution time than QUEST.[24] Today, for stand-alone executable C compilations of q-Davenport and QUEST running under the Microsoft Windows XP operating system on a Pentium D processor, that ratio has dropped to only about 30.

While a factor of 30 in speed between QUEST and q-Davenport may seem large, it must be recalled that the q-Davenport algorithm is much more stable and robust than QUEST or any of the other algorithms (except for M-SVD) because of the degree of development of library software for computing the spectral decomposition of a real-symmetric matrix. Also, for the q-Davenport algorithm there is no need to exercise the method of sequential rotations [2–4, 10], which can increase overall execution time. Thus, for mission-ready implementations of QUEST and Davenport's original q-algorithm, the relative speed may differ by a factor much less than 30, as we shall show in a later publication in progress. The computation of the attitude-error covariance matrix or the TASTE test [4, 10, 27, 28] is no more burdensome in conjunction with the q-Davenport algorithm than with QUEST. The number of lines of code in on-board mission software devoted to the attitude computation is probably less than one percent except for the most primitive spacecraft. For implementation in the dedicated microprocessor of a star tracker, the speed of attitude computation may be more important. Still, one expects even in that case that data conversion, data adjustment, and conversion for telemetry will be much more time-consuming. In the great scheme of things, the burden of attitude computation is unimportant. Now may even be the time to abandon QUEST in favor of standardizing on the much slower q-Davenport algorithm in its original form.

An interesting result of our speed comparisons is that it is ESOQ, the Mortari algorithm least different from QUEST,[25] not the presumptive top-of-the-line Mortari algorithm ESOQ2, which is the better performer in realistic software implementations. ESOQ2 is the better performer only in the MATLAB® environment.[26]

In any event, this work has shown that for speed, any of the six algorithms examined here (in chronological order of creation: q-Davenport, QUEST, M-SVD, FOAM, ESOQ and ESOQ2) is an excellent candidate for mission support. QUEST has the longest record (nearly three decades) of reliable service, but that need not be the deciding criterion.

[24]Myron Shear, Computer Sciences Corporation, System Sciences Division, Silver Spring, Maryland, 1978 (private communication).

[25]It differs chiefly now in the use of the FOAM characteristic polynomial and the specific *formal* manner by which it solves the equation $(K - \lambda_{\max} I_{4\times4})\bar{q}^* = \mathbf{0}$. The final solution, however, is virtually identical, especially computationally.

[26]Neither ESOQ or ESOQ2, however, is superior to OSOQ1, our slightly "souped-up" ESOQ, in reference [1] in any computer environment. Occasionally, OSOQ1 is even minutely faster than QUEST.

Acknowledgment

The authors are very grateful to F. Landis Markley, creator of the M-SVD and FOAM algorithms, for generously supplying them with the MATLAB® m-files used in references [9] and [10] and for helpful comments. The authors are especially grateful to Grant Martin of The Mathworks, Inc., for much kind help and good advice concerning the execution-time tests, and to F. Landis Markley, Douglas P. Niebur, Sergei Tanygin, Jozef van der Ha, Franklin G. VanLandingham, and Gary Welter for helpful comments. The first author wishes to thank John L. Crassidis and Tarunraj Singh of the University at Buffalo for their continued support.

References

[1] CHENG, Y. and SHUSTER, M. D., "The Speed of Attitude Estimation," presented as paper AAS 07-105, *17th AAS/AIAA Space Flight Mechanics Meeting*, Sedona, Arizona, January 28–February 2, 2007; Proceedings: *Advances in the Astronautical Sciences*, Vol. 127, 2007, pp. 101–116.

[2] SHUSTER, M. D. "Approximate Algorithms for Fast Optimal Attitude Computation" presented as paper AIAA 78–1249, *Proceedings, AIAA Guidance and Control Conference*, Palo Alto, California, August 1978, pp. 88–95.

[3] SHUSTER, M. D. and OH, S. D. 'Three-Axis Attitude Determination from Vector Observations," *Journal of Guidance and Control*, Vol. 4, No. 1, January–February 1981, pp. 70–77.

[4] SHUSTER, M. D. "The Quest for Better Attitudes," *The Journal of the Astronautical Sciences*, Vol. 54, Nos. 3 and 4, July–September 2006, pp. 657–683.

[5] WAHBA, G. "Problem 65–1: A Least Squares Estimate of Spacecraft Attitude," *SIAM Review*, Vol. 7, No. 3, July 1965, p. 409.

[6] SHUSTER, M. D. "A Survey of Attitude Representations," *The Journal of the Astronautical Sciences*, Vol. 41, No. 4, October–December 1993, pp. 439–517.

[7] SHUSTER, M. D. "Maximum Likelihood Estimation of Spacecraft Attitude," *The Journal of the Astronautical Sciences*, Vol. 37, No. 1, January–March, 1989, pp. 79–88.

[8] GOLUB, G. H. and VAN LOAM, C. F. *Matrix Computations*, The Johns Hopkins University Press, Baltimore, 1983.

[9] MARKLEY, F. L. and MORTARI, D. "How to Estimate Attitude from Vector Observations," presented as paper AAS 99-427, *AAS/AIAA Astrodynamics Conference*, Girdwood, Alaska, August 16–19, 1999; Proceedings: *Advances in the Astronautical Sciences*, Vol. 103, pp. 1979–1996, 1999.

[10] MARKLEY, F. L. and MORTARI, M. "Quaternion Attitude Estimation Using Vector Measurements," *The Journal of the Astronautical Sciences*, Vol. 48, Nos. 2 and 3, April–September 2000, pp. 359–380.

[11] CHENG, Y. and SHUSTER, M. D. "An Improvement to the QUEST Algorthm," submitted to *The Journal of the Astronautical Sciences*.

[12] FARRELL, J. L. and STUELPNAGEL, J. C. "Solution to Problem 65–1," *SIAM Review*, Vol. 8, No. 3, July 1966, pp. 384–386.

[13] WESSNER, R. H. "Solution to Problem 65–1," *SIAM Review*, Vol. 8, No. 3, July 1966, p. 386.

[14] BROCK, J. E. "Solution to Problem 65–1," *ibid.*

[15] KEAT, J. *Analysis of Least Squares Attitude Determination Routine DOAOP*, Computer Sciences Corporation, CSC/TM–77/6034, February 1977.

[16] LERNER, G. M. "Three-Axis Attitude Determination," in WERTZ, J. R. (ed.), *Spacecraft Attitude Determination and Control*, Springer Scientific + Business Media, Berlin and New York, 1978, pp. 420–428, especially the footnote on p. 425.

[17] MARKLEY, F. L. "Attitude Determination Using Vector Observations and the Singular Value Decomposition," *The Journal of the Astronautical Sciences*, Vol. 36, No. 3, July-September 1988, pp. 245–258.

[18] MARKLEY, F. L. "Attitude Determination Using Vector Observations: a Fast Optimal Matrix Algorithm," *The Journal of the Astronautical Sciences,* Vol. 41, No. 2, April-June 1993, pp. 261–280.

[19] MORTARI, D. "Euler-2 and Euler-n Algorithms for Attitude Determination from Vector Observations," *Space Technology*, Vol. 16, Nos. 5–6, 1996, pp. 317–321.

[20] MORTARI, D. "Euler-q Algorithm for Attitude Determination from Vector Observations," *Journal of Guidance, Control and Dynamics*, Vol. 12, No. 2, March-April 1998, pp. 328–334.

[21] MORTARI, D. "ESOQ: A Closed-Form Solution to the Wahba Problem," *The Journal of the Astronautical Sciences*, Vol. 45, No. 2, April-June 1997, pp. 195–204.

[22] MORTARI, D. "ESOQ2 Single-Point Algorithm for Fast Optimal Attitude Determination," Paper AAS-97-167, *Advances in the Astronautical Sciences*, Vol. 97, Part II, 1997, pp. 803–816.

[23] MORTARI, D. "Second Estimator for the Optimal Quaternion," *Journal of Guidance, Control and Dynamics*, Vol. 23, No. 4, September–October 2000, pp. 885–888.

[24] BAI, Z., BISCHOF, C. and BLACKFORD, L. S. *LAPACK User's Guide*, The Society for Industrial and Applied Mathematics, 2000.

[25] The MATHWORKS, "MATLAB incorporates LAPACK increasing the speed and capabilities of matrix computation," *Cleve's Corner* in *MATLAB News and Notes* (a quarterly Internet newsletter), Winter 2002.

[26] GRAHAM, B. S. *Matrix Eigensystem Routines—EISPACK Guide*, Springer, New York and Berlin, 1977.

[27] SHUSTER, M. D. "The TASTE Test," *The F. Landis Markley Astronautics Symposium*, Cambridge, Maryland, June 29–July 2, 2008; Proceedings: *Advances in the Astronautical Sciences*, Vol. 132, 2008, pp. 71–81.

[28] SHUSTER, M. D. "The TASTE Test," *The Journal of the Astronautical Sciences*. (to appear)

[29] CODY, W. J. Jr. and WAITE W. *Software Manual for the Elementary Functions*, Prentice-Hall, Englewood Cliffs, New Jersey, 1980.

[30] The MATHWORKS, "Function Reference 1: (A–E)," p. 2-897.

[31] The MATHWORKS, "The MATLAB JIT Accelerator," *Technology Backgrounder* (an Internet newsletter), September 2, 2000.

[32] CHENG, Y. and SHUSTR, M. D. "Validation Tests for Spacecraft Attitude Determination" (in preparation).