AAS 07-105

THE SPEED OF ATTITUDE ESTIMATION

Yang Cheng[1] and Malcolm D. Shuster[2]

[Go] wisely and slowly, they stumble that run fast.

William Shakespeare (1564-1616)
Romeo and Juliet, Act II, scene iii

欲速則不達[3]
Confucius (551–479 BCE)

Abstract

Claims that the QUEST algorithm for spacecraft attitude estimation is slower than more recent solutions to the Wahba problem are examined. It is shown that these claims depend on the selection of data reported in their support. A more complete analysis shows that in some cases QUEST indeed requires a greater number of MATLAB® floating-point operations than some other algorithms, but that QUEST is more efficient in the majority of cases. In computational speed, as determined by MATLAB® execution time, QUEST has been found to be fastest in all cases without qualification. For tests of stand-alone executable programs in the C language, QUEST is not faster in all cases than our OSOQ1 algorithm, which is only a mostly faster implementation of the ESOQ1 algorithm, but generally faster than other implementatins of ESOQ algorithms. A careful study is made of the problems besetting tests of flop counts and execution time, especially within MATLAB®. The floating-point operation requirements and execution times of all algorithms are given careful scrutiny.

Introduction

In an earlier work [1], we investigated the accuracy and robustness of the QUEST algorithm [2] and the other fast batch attitude estimators based on the Wahba problem [3]. In that work we showed that assertions [4] that the QUEST algorithm was not accurate

[1]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.buffalo.edu/~cheng3.
[2]Director of Research, Acme Spacecraft Company, 13017 Wisteria Drive, Box 328, Germantown, Maryland 20874. email: mdshuster@comcast.net. website: http://home.comcast.net/~mdshuster.
[3]Yú sù zé bù dá. Wish for haste and there is no success.

and robust compared to other fast batch attitude estimators were true only if QUEST were executed in an unnecessary manner in a very far-fetched scenario. Reference [4] left the reader with the misimpression that QUEST was less capable than the algorithms developed by the authors of Reference [4], and, perhaps, even dangerous to use, which was not true. Reference [1] showed, in fact, that following a very minor rearrangement of terms in the QUEST characteristic polynomial,[4] QUEST performed just as well as any other algorithm even in the unphysical and unreasonable scenario 2 or Reference [4].

In the present work we continue Part I [1] of this article and address our attention to the question of computational speed. Reference [4] made several claims, in particular, as noted in Part I, it claimed that:

- QUEST is less accurate than the algorithms developed by the authors of Reference [7] and does not necessarily converge.
- The zeroth-order approximation of QUEST requires more MATLAB® floating-point operations than the zeroth-order approximations of the ESOQ1 and ESOQ2 algorithms of Mortari.
- The first-order approximation of QUEST requires more MATLAB® floating-point operations than the first-order algorithms ESOQ1.1 and ESOQ2.1.

Part I of this work [1] refuted the first claim at length and that refutation need not be repeated here. Suffice it to say that all solutions to the Wahba problem[5] are equally accurate, and QUEST is as robust as any other solution to the Wahba problem except for Davenport's original implementation of the q-method and Markley's SVD algorithm,[6] which are extremely stable but also very slow.

The second and third claims are absolutely true. However, they are the only cases in which QUEST requires more MATLAB® floating-point operations than the algorithms examined in Reference [4]. In every other case, the QUEST algorithm requires fewer MATLAB® floating-point operations than any of the other fast optimal attitude estimators. In terms of MATLAB® execution time the QUEST algorithm is the fastest algorithm in all cases without qualification. In terms of the execution time of stand-alone executable compilations of C-programs, the only true measure of computational speed, then either QUEST or our OSOQ1 algorithm (an optimized ESOQ1) is the fastest depending on the case. Reference [4] tells the truth, but, obviously, it does not tell the whole truth, because the scope of its numerical investigations were too limited.

The presentation of complete results for the flop requirements and execution times of the fast optimal attitude estimators is the subject of the present Part II of this work. We shall begin with a discussion of the solutions to the Wahba problem from the point of view of computational burden, and following this a discussion of the nature of "timing" tests as performed within MATLAB® and as compiled C-code and the numerical results.

[4]The expression $\lambda^4 - (a+b)\lambda^2 + ab$ is replaced by $(\lambda^2 - a)(\lambda^2 - b)$.

[5]These do not include the ESOQ1.1 and ESOQ2.1 algorithms, because (by design) they are only first-order approximations. We shall, however, perform tests of these two algorithms, because they figure prominently in Reference [4].

[6]As in Reference [1]. 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 [5]. 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 are also implementations of the q-method.

In order to make our language less ambiguous we will say that an algorithm is more *efficient* if it requires fewer floating-point operations and *faster* if is requires less execution time. As we shall see, the number of floating-point operations is a poorer indicator of efficiency than the number of binary operations, and the number of floating-point operations provided by MATLAB® may be the worst indicator of all.

The present work contains no startling theory. Its interest derives from the depth and breadth of the numerical results. A great deal of time has been spent in the past on efficiency and speed tests, typically in MATLAB®. In the present work we discuss carefully the value of such tests, with particular regard for one test which has been the subject of dispute for nearly a decade, namely which of the two algorithms, QUEST or ESOQ2, is the fastest. Our clear answer: neither of the above.

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 to the attitude profile matrix B or the Davenport matrix K. These algorithms are described in detail in [4] and their description need not be repeated here. Examples of such methods are the polar-decomposition method of Farrell and Stuelpnagel [4, 6], the matrix square-root algorithm of Wessner [4, 7] and Brock [4, 8], the q-method of Davenport [2, 4, 9], and Markley's SVD algorithm [4, 10], which is very similar to Farrell and Stuelpnagel's polar-decomposition algorithm in concept and execution.

The second class of solutions first determines λ_{max}, the maximum characteristic value of the Davenport matrix K [1, 2, 4], by some method especially tailored to the Wahba problem and then, knowing λ_{max}, solves the simpler problem of constructing the optimal quaternion $\bar{q}^*$ or the optimal direction-cosine matrix A^* from K or B. These are the faster solutions to the Wahba problem. The first of these was QUEST [2, 4] followed a decade later by FOAM [4, 11] and finally by Mortari's many Euler and ESOQ algorithms [4, 12–16].[7] Of the Mortari algorithms, the top-of-the-line algorithm is ESOQ2 [4, 15, 16]. The algorithms of this second group solve for λ_{max} by applying the Newton-Raphson method to the characteristic polynomial using an initial value λ_{o} easily calculable from the weights of the Wahba cost function. This procedure was introduced in Reference [2] and has been copied by all later algorithms of the second class.[8] The QUEST algorithm uses the QUEST form of the characteristic polynomials, the other algorithms the FOAM form. In infinitely precise arithmetic the two forms are exactly equivalent.

The two classes of algorithms have different numerical qualities. The algorithms of the first class, 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,

[7]ESOQ1.1 and ESOQ2.1 are actually the invention of Markley.

[8]As noted frequently, for attitude data of accuracy 10 arcsec/axis or better, a single Newton-Raphson iteration of the characteristic polynomial will yield a value for λ_{max} whose precision exhausts that of an IEEE double-precision floating-point number (64 bits or approximately 17 significant digits).

whose computational burden may vary from library to library. For example, there are 16 different code paths underlying the MATLAB® (LAPACK [17]) *eig* function, depending on the nature of the arguments [18]. A customized implementation that directly calls the appropriate low-level routine beneath the MATLAB® *eig* interface would improve the execution time of q-Davenport, 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*) in Markley's SVD method. The algorithms of the second class, because they use custom-tailored numerical algorithms are faster.

The computational complexity of using the QUEST or FOAM form of the characteristic polynomial are the same, and the construction of the B or K matrix is the same for all algorithms.[9] Differences in execution 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 $\lambda_{\max}$ and B (equivalently, from $\lambda_{\max}$ and K). A time-saving feature of the QUEST algorithm is that it constructs the optimal quaternion from the same intermediate variables as were used to construct the characteristic polynomial.

In all of the flop-count or timing tests presented here, only minimal versions of the attitude estimation algorithms were tested, as in Reference [4]. Missing from the tests, as in Reference [4], are the examination of the computational burden due to low-level data checking, data adjustment, data validity tests (in QUEST the TASTE test [20]), observability tests, the computation of the attitude covariance matrix, or the method of sequential rotations [2], none of which need be the same for each algorithm. The timing tests of Reference [4] and this work are, thus, somewhat unrealistic in nature and might never have been studied by the present authors, had Reference [4] not established a precedent. In every case we have insisted that the final output be the quaternion, thus placing an additional 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® floating-point operation counts (flop counts)
- MATLAB® execution times
- execution times of MATLAB® C-mex files
- execution times of stand-alone C-language compilations

Of these only the last is pertinent to execution times of mission software. The first three are presented largely because Reference [4] relied solely on MATLAB® flop counts for the computations. As we shall see, MATLAB® flop counts and even MATLAB®

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

execution times are poor criteria for judging algorithm speed. We have used two different C compilers and mathematics libraries to generate the C-mex files and the stand-alone executable programs. It is interesting that these two do not generate compatible results for relative execution times.

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 each consisting of 2, 3, 4, or 25 measurements. The test samples for 2, 3, and 4 measurements were an obvious subset of the 25-measurement samples. We 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 [3, 4]. Measurement noise was applied to the reference directions by adding a Gaussian random error of mean zero and standard deviation $\sigma = 6$ arcsec 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 $n = 2, 3, 4,$ or 25. When determining execution times these 1000 samples have been repeated 5,000 times to reduce truncation 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 with 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) or the GCC (GNU compiler collection) C/C++ compiler.

In our efficiency and speed tests we have examined the iterative algorithms: QUEST, QUEST$^{\mathrm{MM}}$, FOAM, ESOQ1, ESOQ2, OSOQ1, OSOQ2, QUESTOQ1 and QUESTOQ2, and the non-iterative algorithms: q-Davenport, M-SVD, ESOQ1.1 and ESOQ2.1. The last two perform one execution for the solution of the maximum overlap characteristic value $\lambda_{\max}$, a normally iterative procedure. Since these algorithms perform exactly one and only one execution of the computation of $\lambda_{\max}$, we have classed them as non-iterative. QUEST$^{\mathrm{MM}}$ is the version of QUEST implemented by Reference [4] and used an external function call to evaluate determinants. The original QUEST implementation calculated the determinant explicitly without an external function call. Both implementations have been tested for efficiency and speed. The differences in the results for QUEST and QUEST$^{\mathrm{MM}}$ will show the extent to which minute differences in coding effect these results. OSOQ1 and OSOQ2 differ from ESOQ1 and ESOQ2 in the same way that QUEST differs from QUEST$^{\mathrm{MM}}$. QUESTOQ1 and QUESTOQ2 are variations of ESOQ1 and ESOQ2 in which the QUEST characteristic polynomial is used for the computation of the maximum overlap characteristic value. The QUESTOQ1 and QUESTOQ2 results will not be presented explicitly.

Numerical Results: MATLAB® Flop Counts

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 [18]. 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 in flop counting. More importantly, for modern computer architectures floating-point operations may not be the dominating factor in execution speed. Memory reference and cache usage may be more important [18]. 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 characteristic value 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 64 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 [22].

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) [23]. 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, calculates the determinant by performing an LU factorization, and thus 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® *det* function has been precompiled, however, the complexity of *det* is hidden somewhat by its high speed.

Table 1 shows our results for flop counts of the fast optimal algorithms. Reference [4] found that for zero iterations the ESOQ1 and ESOQ2 algorithms were most efficient, which is borne out also by our results here. That work found also that for a single iteration, ESOQ1.1 was the fastest. ESOQ1.1 and ESOQ2.1 are only approximate algorithms incapable of refinement by further iteration. We note also that in all other cases it is QUEST and QUEST$^{\mathrm{MM}}$ which are the fastest.

In Reference [4], because of the supposed convergence problem, the QUEST algorithm (really the QUEST$^{\mathrm{MM}}$ algorithm) was eliminated as a contender in the speed (really the efficiency) contest. However, as we commented in Reference [1], that lack of robustness

TABLE 1 MATLAB® Floating-Point Operation Counts (Flops) for the Fast Optimal Attitude Estimators

Iterations	FOAM	QUEST^{MM}	ESOQ1	ESOQ2	QUEST	OSOQ1	OSOQ2
2 Measurements							
exact	276	179	246	238	187	178	188
0	257	146	116	114	147	112	113
1	292	183	257	254	193	191	201
2	303	194	269	265	207	202	212
3	314	205	281	276	221	210	223
4	325	216	293	287	235	224	234
5	336	227	305	298	249	235	245
3 Measurements							
0	278	167	137	135	168	133	134
1	313	204	278	275	214	212	222
2	324	215	290	286	228	223	233
3	335	226	302	297	242	234	344
4	346	237	314	308	256	245	255
5	357	248	326	319	270	256	266
4 Measurements							
0	299	188	158	156	189	154	155
1	334	225	299	296	235	233	243
2	345	236	311	307	249	244	254
3	356	247	323	318	263	255	265
4	367	258	335	329	277	266	276
5	378	269	347	340	291	277	287
25 Measurements							
0	740	629	599	597	630	595	596
1	775	666	740	737	676	674	684
2	786	677	752	748	690	685	695
3	797	688	764	759	704	696	706
4	808	699	776	770	718	707	717
5	819	710	788	781	732	718	728

TABLE 2. MATLAB® Floating-Point Operation Counts (Flops) for the q-Davenport, M-SVD, ESOQ1.1, and ESOQ2.1 Algorithms

Number of Measurements	q-Davenport	M-SVD	ESOQ1.1	ESOQ2.1
2	715	523	161	163
3	752	797	182	184
4	771	853	203	205
25	1189	1347	644	646

occurred only for a very unphysical and somewhat ridiculous example, and disappeared entirely with a trivial rearrangement of terms in the QUEST characteristic polynomial (see footnote 5 above). At the same time ESOQ1 and ESOQ2 win the speed contest on the basis of their performance for no iterations whatsoever, the case for which QUEST, even for the far-fetched scenario 2 of Reference [4], has no robustness problem. This seems a somewhat contrived result. Clearly, in the vast majority of cases it is QUEST (or QUEST$^{\mathrm{MM}}$) which is the most efficient in MATLAB®.

Table 2 shows the results for the non-iterative algorithms. 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 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 in its system. This greatly lessens the execution time of MATLAB® programs but not necessarily consistently. A program may execute more quickly than a subset of its operations. 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.

The MATLAB® JIT-accelerator reduces interpreter and data-handling overhead [24] by converting many p-code[10] 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 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, therefore, 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.

For OSOQ1 and OSOQ2 the MATLAB® *profile* function was used to optimize the m-files by tracking their execution times. Those parts of the m-files that took unnecessarily long to execute were rewritten to improve efficiency.

Tables 3 and 4 show the timing results for the execution of the MATLAB® m-files. We note immediately that QUEST$^{\mathrm{MM}}$ and QUEST are the fastest algorithms in all cases, and our optimized OSOQ1 and OSOQ2 are faster then the corresponding ESOQ1 and

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

TABLE 3 MATLAB® Execution Times (μs) for the Fast Optimal Attitude Estimators (JIT-Accelerator off)

Iterations	FOAM	QUEST^{MM}	ESOQ1	ESOQ2	QUEST	OSOQ1	OSOQ2
2 Measurements							
exact	241	222	353	343	227	325	291
0	219	175	230	195	181	216	185
1	255	230	371	359	236	342	310
2	264	241	380	368	248	350	319
3	273	253	389	378	260	359	329
4	283	265	398	386	271	368	336
5	291	276	407	395	283	376	345
3 Measurements							
0	220	175	229	195	181	217	185
1	259	234	376	364	240	348	312
2	268	246	386	373	253	357	321
3	277	258	395	383	264	366	330
4	286	269	403	391	276	375	339
5	295	281	412	400	286	383	348
4 Measurements							
0	221	177	231	196	183	218	186
1	261	235	376	364	241	348	314
2	270	248	386	373	253	358	323
3	278	259	395	382	265	366	332
4	288	270	403	391	276	375	340
5	296	282	413	400	288	384	349
25 Measurements							
0	229	187	240	207	192	226	195
1	271	244	385	373	252	357	323
2	280	256	394	383	264	366	332
3	289	269	403	392	275	374	341
4	297	280	412	401	287	384	349
5	306	292	421	409	299	393	358

TABLE 4. MATLAB® Execution Times (μs) for the q-Davenport, M-SVD, ESOQ1.1, and ESOQ2.1 Algorithms (JIT-accelerator off)

Number of Measurements	q-Davenport	M-SVD	ESOQ1.1	ESOQ2.1
2	143	208	287	268
3	141	212	288	268
4	142	213	289	269
25	151	223	296	279

ESOQ2. 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 M-SVD algorithms exceed 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.

Numerical Results: MATLAB® C-mex Files

For the C-mex files and stand-alone executable programs we have examined the execution times for two different C environments and mathematics libraries. The first of these was Microsoft Visual Studio Net 2003 (also known as Visual C++ 7.1) using the LAPACK routines for characteristic value decomposition and singular-value decomposition taken from the MATLAB® mathematics library (labeled there *eig* and *svd*. The second C environment was GCC and the similar routines from the GSL (GNU Scientific Library). For the iterative algorithms, we have presented the results only for C-mex files constructed using Microsoft Visual C++, but for the non-iterative algorithms, we have shown the results for both compilers to show the typical differences in speed.

It is obvious from Table 5 that the execution times are dominated by the MATLAB® overhead. The execution times for the C-mex files are approximately an order of magnitude greater than those for the m-files. 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. Note that the execution times for the C-mex files in Table 6 created using the GCC are somewhat faster than those created using Microsoft Visual C++. This is likely due to the fact that the Microsoft Visual C++ versions have used the LAPACK mathematics library of MATLAB®, whose functions are much more complex than those of the GSL library used by C.

Numerical Results: Stand-alone Executable Programs

For stand-alone executables, the timing function (Windows application programming interface, API) *timeGetTime* was used to measure elapsed time under Windows XP. Here, we have presented explicit results for both Microsoft Visual C++ and for GCC.

The results for the iterative algorithms are shown in Tables 7 and 8. The results of the two tables are very similar. We see that Microsoft Visual C ++ favors OSOQ1, our optimized ESOQ1, while the GCC favors QUEST. Note that the FOAM algorithm requires considerably longer execution times than the other iterative algorithms, which are all roughly comparable. As one gets closer to real mission software, the differences in the execution times becomes less important. For the most part the execution times for the QUESTOQ algorithms were intermediate between those of the respective OSOQ and ESOQ algorithms.

The results for the non-iterative algorithms are shown in Table 9. Here, we have included also the combination of the Microsoft Visual C++ compiler with the GSL mathematics library. The remaining fourth combination of the GCC coupled with the MATLAB® mathematics library presented special difficulties in MATLAB® 6.5 and was not studied. The differences in execution times reflect the better optimization capabilities of the Microsoft Visual C++ compiler and the greater simplicity of the functions in the GSL mathematics library.

TABLE 5. Execution Times (μs) for C-mex Files of the Fast Optimal Attitude Estimators within MATLAB® (using Microsoft Visual C++)

Iterations	FOAM	QUEST^{MM}	ESOQ1	ESOQ2	QUEST	OSOQ1	OSOQ2
2 Measurements							
exact	32.720	31.500	32.686	31.092	31.500	33.030	33.062
0	31.186	31.220	31.812	31.406	31.406	30.874	31.314
1	33.218	32.374	33.030	33.844	31.842	32.030	31.312
2	30.156	32.218	32.968	33.156	32.376	31.500	31.718
3	32.032	32.186	33.218	33.250	32.000	32.062	32.844
4	32.656	32.814	32.624	33.562	31.906	31.718	31.874
5	32.906	31.876	33.156	33.562	32.126	32.782	32.250
3 Measurements							
0	31.156	30.844	31.374	31.562	31.436	31.376	31.718
1	32.936	32.280	32.376	30.812	31.500	32.342	32.906
2	32.188	32.500	32.436	31.342	32.218	31.968	32.156
3	32.186	32.220	32.876	31.282	31.468	32.344	32.094
4	31.718	32.186	32.718	31.436	31.312	31.218	32.530
5	32.408	31.624	32.094	31.032	31.564	32.282	31.844
4 Measurements							
0	30.844	31.968	30.250	31.124	30.000	29.780	31.250
1	33.438	33.686	32.064	31.406	33.812	33.436	33.782
2	32.812	33.532	32.688	31.156	32.782	33.344	33.592
3	32.750	33.342	32.374	31.032	33.186	33.250	33.218
4	33.218	33.814	32.874	31.376	32.626	33.530	33.250
5	33.376	33.000	32.938	31.500	33.562	33.626	32.936
25 Measurements							
0	35.562	34.438	34.250	34.438	35.156	34.344	34.656
1	35.094	34.594	35.688	34.562	35.656	34.562	35.530
2	34.968	35.188	35.436	34.686	35.656	35.874	35.188
3	35.406	35.092	35.500	34.688	35.344	35.376	35.250
4	35.906	35.064	35.342	34.750	35.906	34.780	34.812
5	36.280	34.874	35.814	34.562	35.936	35.938	35.156

TABLE 6. Execution Times (μs) for the q-Davenport and M-SVD Algorithms as C-mex files in MATLAB®

Number of Measurements	q-Davenport (MSVC/MATLAB)	q-Davenport (GCC/GSL)	M-SVD (MSVC/MATLAB)	M-SVD (GCC/GSL)
2	72.97	54.53	69.38	57.81
3	71.87	54.85	74.53	57.03
4	71.88	54.53	75.47	57.03
25	75.00	57.66	78.43	60.63

TABLE 7. Execution Times (μs) for Stand-Alone Executable Versions of the Fast Optimal Attitude Estimators (using Microsoft Visual C++)

Iterations	FOAM	QUEST^{MM}	ESOQ1	ESOQ2	QUEST	OSOQ1	OSOQ2
2 Measurements							
exact	0.447	0.350	0.414	0.472	0.343	0.337	0.405
0	0.382	0.263	0.250	0.303	0.268	0.244	0.304
1	0.429	0.337	0.426	0.479	0.332	0.323	0.401
2	0.465	0.369	0.459	0.510	0.363	0.360	0.436
3	0.500	0.403	0.496	0.552	0.395	0.391	0.469
4	0.530	0.436	0.534	0.581	0.429	0.424	0.501
5	0.573	0.475	0.574	0.630	0.468	0.461	0.540
3 Measurements							
0	0.397	0.281	0.267	0.320	0.290	0.265	0.325
1	0.460	0.351	0.447	0.518	0.343	0.350	0.420
2	0.495	0.382	0.481	0.533	0.376	0.384	0.452
3	0.530	0.414	0.522	0.565	0.409	0.416	0.485
4	0.560	0.447	0.557	0.626	0.442	0.451	0.519
5	0.603	0.486	0.597	0.637	0.480	0.490	0.559
4 Measurements							
0	0.419	0.306	0.295	0.349	0.307	0.292	0.350
1	0.478	0.373	0.471	0.548	0.369	0.366	0.437
2	0.514	0.404	0.503	0.561	0.400	0.400	0.468
3	0.549	0.437	0.548	0.593	0.432	0.431	0.502
4	0.580	0.471	0.584	0.657	0.467	0.466	0.536
5	0.624	0.509	0.624	0.668	0.505	0.506	0.575
25 Measurements							
0	1.144	1.021	1.003	1.072	1.023	1.000	1.069
1	1.209	1.099	1.168	1.266	1.093	1.094	1.157
2	1.246	1.131	1.204	1.280	1.127	1.122	1.190
3	1.285	1.158	1.246	1.315	1.155	1.153	1.223
4	1.313	1.193	1.288	1.374	1.190	1.185	1.260
5	1.353	1.231	1.326	1.386	1.228	1.226	1.298

Note that the execution time for q-Davenport is at best about 60 times longer than that of QUEST.

TABLE 8. Execution Times (μs) for Stand-Alone Executable Versions of the Fast Optimal Attitude Estimators (compiled using GCC)

Iterations	FOAM	QUEST^{MM}	ESOQ1	ESOQ2	QUEST	OSOQ1	OSOQ2
2 Measurements							
exact	0.617	0.412	0.441	0.504	0.397	0.431	0.558
0	0.531	0.328	0.295	0.334	0.332	0.291	0.396
1	0.576	0.434	0.459	0.531	0.415	0.452	0.551
2	0.610	0.468	0.497	0.569	0.446	0.513	0.585
3	0.648	0.504	0.536	0.607	0.481	0.531	0.624
4	0.685	0.538	0.579	0.642	0.517	0.569	0.661
5	0.723	0.575	0.615	0.680	0.552	0.608	0.698
3 Measurements							
0	0.551	0.350	0.315	0.357	0.346	0.322	0.422
1	0.602	0.438	0.480	0.540	0.419	0.462	0.553
2	0.643	0.473	0.519	0.578	0.454	0.499	0.589
3	0.679	0.506	0.557	0.613	0.488	0.536	0.631
4	0.718	0.539	0.599	0.653	0.522	0.579	0.666
5	0.757	0.575	0.636	0.690	0.554	0.614	0.706
4 Measurements							
0	0.589	0.382	0.347	0.384	0.383	0.350	0.455
1	0.644	0.478	0.520	0.577	0.460	0.510	0.593
2	0.684	0.513	0.560	0.615	0.495	0.546	0.630
3	0.720	0.548	0.598	0.653	0.528	0.585	0.668
4	0.757	0.582	0.642	0.691	0.561	0.623	0.705
5	0.796	0.616	0.680	0.728	0.595	0.659	0.742
25 Measurements							
0	1.618	1.397	1.372	1.283	1.403	1.241	1.416
1	1.653	1.468	1.504	1.577	1.455	1.487	1.579
2	1.694	1.505	1.545	1.608	1.485	1.525	1.619
3	1.731	1.539	1.578	1.675	1.522	1.568	1.657
4	1.765	1.578	1.621	1.684	1.560	1.597	1.694
5	1.803	1.603	1.660	1.722	1.587	1.637	1.733

Discussion and Conclusions

We have seen that the performance of the Wahba algorithms, with regard to accuracy, robustness and speed, is much more complex than presented in Reference [4]. The comments made by Reference [4] were correct certainly, but the larger picture is much more interesting.

TABLE 9. Stand-alone Execution Times (μs) for the q-Davenport and M-SVD Algorithms

Number of Measurements	q-Davenport (MSVC/MATLAB®)	q-Davenport (GCC/GSL)	q-Davenport (MSVC/GSL)
2	34.64	23.79	18.11
3	34.95	24.14	18.50
4	35.08	24.22	18.41
25	35.68	24.61	18.61
	M-SVD (MSVC/MATLAB®)	M-SVD (GCC/GSL)	M-SVD (MSVC/GSL)
2	30.51	23.49	18.85
3	36.51	22.40	18.85
4	37.29	22.44	18.93
25	39.10	23.57	19.65

As pointed out in Part I of this work [1], when the QUEST characteristic equation has been put in partially-factored form, that is, in a form similar to that of the FOAM algorithm, QUEST becomes as robust and as accurate for any number of Newton-Raphson iterations as any other algorithm, except (in robustness) for the q-Householder and M-SVD algorithms, of course.

In efficiency and speed, the subject of the present Part II, 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 great inefficiency of the C-mex files in general make the speed difference between the iterative algorithms unimportant. Clearly, barring some special operational need, one would never want to create a C-mex file for a solution to the Wahba problem. 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 smaller still and the relative differences are greatly dependent on the choice of compiler and mathematics library. The difference is dependent also on the choice of platform, the operating system, and what other programs are running simultaneously. From the standpoint of implementation in a real mission, there is no significant advantage of one fast implementation of a solution to the Wahba problem over another.

In 1978 using the EISPACK [19] 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.[11] 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 60.

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

While a factor of 60 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). Also, for the q-Davenport algorithm there is no need to exercise the method of sequential rotations [2, 4, 21], which can increase overall execution time. The computation of the attitude-error covariance matrix or the TASTE test [2, 4, 21] 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. In the great scheme of things makes the burden of attitude computation unimportant. Now may be the time to abandon QUEST in favor of standardizing on the q-Davenport algorithm. QUEST has had a long and useful career in attitude estimation. Perhaps, it is time for QUEST to retire.

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 Reference [8] 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.

References

[1] CHENG, Y. and SHUSTER, M. D., "Fast Optimal Attitude Estimators: I. Factorization, Accuracy and Robustness," submitted to *The Journal of the Astronautical Sciences*.

[2] 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.

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

[4] 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.

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

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

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

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

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

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

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

[18] 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.

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

[20] SHUSTER, M. D. and FREESLAND, D. C. “The Statistics of TASTE and the Inflight Estimation of Sensor Precision,” *Proceedings (CD), NASA Flight Mechanics Symposium*, NASA Goddard Space Flight Center, Greenbelt, Maryland, October 18–20, 2005.

[21] SHUSTER, M. D. “In Quest of Better Attitudes” (Dirk Brouwer lecture), Paper No. AAS-01-250, *Advances in the Astronautical Sciences*, Vol. 108, 2001, pp. 2089–2117.

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

[23] The MATHWORKS, "Function Reference: det," http://www.mathworks.com/access/helpdesk/help/techdoc/ref/index.html?/access/helpdesk/help/techdoc/refdet.html&.

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