Backward error analysis in computational geometry

D. Jiang and N. F. Stewart*

Département IRO, Université de Montréal,
CP6128, Succ. CentreVille, H3C 3J7, Canada.
{jiangdi,stewart}@iro.umontreal.ca

Abstract. A recent paper, published in Algorithms—ESA2004, presented examples designed to illustrate that using floating-point arithmetic in algorithms for computational geometry may cause implementations to fail. The stated purpose was to demonstrate, to students and implementors, the inadequacy of floating-point arithmetic for geometric computations. The examples presented were both useful and insightful, but certain of the accompanying remarks were misleading. One such remark is that researchers in numerical analysis may believe that simple approaches are available to overcome the problems of finite-precision arithmetic. Another is the reference, as a general statement, to the inadequacy of floating-point arithmetic for geometric computations.
In this paper it will be shown how the now-classical backward error analysis can be applied in the area of computational geometry. This analysis is relevant in the context of uncertain data, which may well be the practical context for computational-geometry algorithms such as, say, those for computing convex hulls. The exposition will illustrate the fact that the backward error analysis does not pretend to overcome the problem of finite precision: it merely provides a tool to distinguish, in a fairly routine way, those algorithms that overcome the problem *to whatever extent it is possible to do so.*
It will also be shown, by using one of the examples of failure presented in the principal reference, that often the situation in computational geometry is exactly parallel to other areas, such as the numerical solution of linear equations, or the algebraic eigenvalue problem. Indeed, the example mentioned can be viewed simply as an example of an unstable algorithm, for a problem for which computational geometry has already discovered provably stable algorithms.

1 Introduction

This paper presents an exposition of how the combined backward/forward error analysis, from numerical analysis, relates to the study of robustness in computational geometry.

* The research of the second author was supported in part by a grant from the Natural Sciences and Engineering Research Council of Canada.

As stated in our principal reference [1], "... the algorithms of computational geometry are designed for a machine model with exact arithmetic. Substituting floating point arithmetic for the assumed real arithmetic may cause implementations to fail." The authors of [1] go on to say that "due to ... [a] ... lack of examples, instructors of computational geometry have little material for demonstrating the inadequacy of floating point arithmetic for geometric computations, students of computational geometry and implementers of geometric algorithms still underestimate the seriousness of the problem, and researchers in our and neighboring disciplines, e.g., numerical analysis, still believe, that simple approaches are able to overcome the problem." An incremental scan algorithm (which is related to Graham's scan [2] and which we will refer to as *Graham_incremental*), for planar convex hulls, is then studied in some detail. In particular, examples are given which show the algorithm can fail, and an explanation is given for why it fails, when executed with floating-point arithmetic.

The examples given in [1] should indeed be useful to students and teachers of computational geometry, in order to illustrate what can go wrong, and why, when finite-precision arithmetic is used to solve geometric problems. Furthermore, [1] satisfies the criteria of classical experimental science [3, p. 147] in a way that is rare in computer science [4], in that it presents the results of experiments that are repeatable in every detail. In fact, we have implemented the *Graham_incremental* algorithm for example A1 of [1], and we confirm that the algorithm behaves exactly as described in [1] when applied to the data given there[1]. Briefly, for example A1 of [1], *Graham_incremental* produces a completely spurious result.

There are, however, three misleading suggestions in the final sentence quoted above, and it would be unfortunate if they were communicated to students of computational geometry. One of these is the suggestion that the approaches of computational geometry and numerical analysis are somehow adversarial, since in fact they are complementary. Another is the suggestion that numerical analysts believe that they can "overcome" the problem of finite precision. This is not true. What *is* true, however, is that in the case where input data is uncertain and a stability result is available, a backward/forward error analysis, and often a pure backward error analysis, can deal with the problem in a fairly routine way, by showing that a stable algorithm overcomes the problem of finite precision *to whatever extent it is possible to do so*. Indeed, a stable algorithm provides us with a solution that is as good as the data warrants [5]. (Stability will be defined below in the context of a combined backward/forward analysis, but we will usually just refer to a backward error analysis, since this is usually sufficient.)

A third misleading remark in the passage from [1], quoted above, is the reference to the "inadequacy" of floating-point arithmetic for geometric computations, which is incorrect as a general statement. In fact, some algorithms using floating point will provide adequate solutions, while others will not, and a backward error analysis will permit us to recognize which algorithms are satis-

[1] Published confirmation of experimental results is common in fields such as experimental physics, and it is satisfying to participate in this kind of process here.

factory. On the other hand, it *is* true that we must begin by defining precisely what constitutes an *adequate*, or *inadequate*, solution to a geometric problem.

In this paper we will show that numerical robustness for the convex-hull problem is analogous to the case of linear equations, or the algebraic eigenvalue problem, and that when input data is uncertain, the difficulties documented in [1] fit exactly into the paradigm of the backward error analysis. We emphasize that this does not imply that research into other paradigms, including exact arithmetic and others, should not be vigorously pursued. Our only claim is that in the proper context (uncertain input data), the backward-error analysis is a useful approach, and it should not be neglected.

In Section 2 of the paper we will present a brief summary of how the backward error analysis is used in numerical linear algebra, and an illustration will be used to show that breakdowns of methods, of the sort described in [1] for the convex-hull problem, are quite typical in other fields. Then, in Section 3, a description of the combined backward/forward error analysis will be given, and applied to the planar convex-hull problem. These ideas were developed several decades ago, but work such as [5] and [6] is very much relevant today. As already mentioned, the first task is to define exactly what is meant by the "inadequacy" of a solution to the convex-hull problem. We are then in a position to do a *perturbation analysis* [6, Ch. 2] to examine the effects of perturbations of the input data (whether they are caused by original uncertainty or by subsequent application of a stable numerical algorithm). Finally, we discuss Fortune's implementation of the Graham scan, which we will call *Graham_Fortune*. This implementation is numerically stable for the planar convex-hull problem, as proved in [7]. Indeed, a slight modification of the algorithm in [7, Sec. 4] will produce a sequence of points that lie on the topological boundary of their convex hull, and this convex set is the correct convex hull for points that have been relatively perturbed by a small amount. Thus, we can use a pure backward error analysis to affirm that *Graham_Fortune* provides a solution that is as good as we can hope for, given that the data is uncertain.

The situation for planar convex hulls is, therefore, closely analogous to the case of solving linear equations. In both cases there exist unstable algorithms (*Graham_incremental*, and Gaussian elimination without pivoting, respectively), and in both cases there exist stable algorithms (*Graham_Fortune*, and Gaussian elimination with total pivoting, respectively). Also, in both cases there exist examples for which unstable algorithms produce complete nonsense, and this with no warning that anything is amiss. In fact, the only breakdown in the analogy is that in the case of the convex-hull problem, with the error criterion used below as an illustration, the situation is much *better* than for solving linear equations. This is because the perturbation analysis, mentioned above, shows that the problem is *well-conditioned*, which is not always true for linear equations. Thus, whereas even a stable algorithm may produce an unsatisfactory answer for the problem $A\boldsymbol{x} = \boldsymbol{b}$ (if A is the Hilbert matrix, for example), a stable algorithm such as *Graham_Fortune* always produces a satisfactory answer for the convex-hull problem.

2 Backward error analysis for linear-equation solvers

For linear equations, the problem is defined by the pair $[A, \boldsymbol{b}]$, and the solution is defined by $\boldsymbol{x}$ such that $A\boldsymbol{x} = \boldsymbol{b}$. We proceed as follows:

a. *Measuring error in the solution space.* A measure of the inadequacy of an approximate solution $\boldsymbol{y}$, for the problem $[A, \boldsymbol{b}]$, is the relative error $\frac{\|\boldsymbol{x}-\boldsymbol{y}\|}{\|\boldsymbol{x}\|}$, where $\|\cdot\|$ denotes any convenient vector norm.

b. *Perturbation Analysis.* A simple argument shows that if δA is a matrix representing perturbation of the elements of A, and if $\delta\boldsymbol{b}$ is a vector representing perturbations of the elements of $\boldsymbol{b}$, then the solution $\boldsymbol{y}$ of the perturbed problem $[A + \delta A, \boldsymbol{b} + \delta\boldsymbol{b}]$ satisfies (neglecting second-order terms):

$$\frac{\|\boldsymbol{x} - \boldsymbol{y}\|}{\|\boldsymbol{x}\|} \leq \|A\| \cdot \|A^{-1}\| \left\{ \frac{\|\delta A\|}{\|A\|} + \frac{\|\delta\boldsymbol{b}\|}{\|\boldsymbol{b}\|} \right\}, \tag{1}$$

where $\|\cdot\|$ is now used also to denote a matrix norm subordinate [6, p. 56] to the vector norm introduced above. The quantity $\|A\| \cdot \|A^{-1}\|$ is usually referred to as the *condition number* of the problem: for a trivial matrix like the identity it will be equal to 1, while for a Hilbert matrix of even moderate dimension it will be very large. The condition number represents the amount by which a given perturbation of the input data for $A\boldsymbol{x} = \boldsymbol{b}$ will be magnified in the solution. A problem with a low condition number is said to be *well-conditioned*, and a problem with a large condition number is said to be *ill-conditioned*. The two cases are illustrated by the lines linking problems to solutions in Figures 1 and 2, where P denotes the class of problems, and S denotes the class of solutions [8].

c. *Stability proof.* The third step is to seek *stable* algorithms, that is, algorithms that produce a slightly incorrect solution to a slightly perturbed problem [5], as illustrated by the unfilled circles in Figures 1 and 2. (This describes a combined backward/forward error analysis; if the words "a slightly incorrect solution" can be replaced by "the exact solution", so that there is no need for the unfilled circle in S, then we have a pure backward error analysis.) Gaussian elimination with total pivoting is stable for the problem $A\boldsymbol{x} = \boldsymbol{b}$. Such algorithms produce answers that are, for practical purposes, as good as the best answers we can hope for *(even using infinite precision)*, if the "slight perturbation" is smaller than the uncertainty already in the data. Furthermore, by the perturbation analysis of step b, above, the size of the error in the solution can be estimated.

It should be observed that the concept of problem condition, and the corresponding perturbation analysis, are considered prior to any discussion of numerical methods [6, Ch. 2]. This reflects the central idea of the backward error analysis: if the elements of A contain uncertainty that may be as large as $\frac{\|\delta A\|}{\|A\|}$, and the elements of $\boldsymbol{b}$ contain uncertainty that may be as large as $\dfrac{\|\delta\boldsymbol{b}\|}{\|\boldsymbol{b}\|}$, then

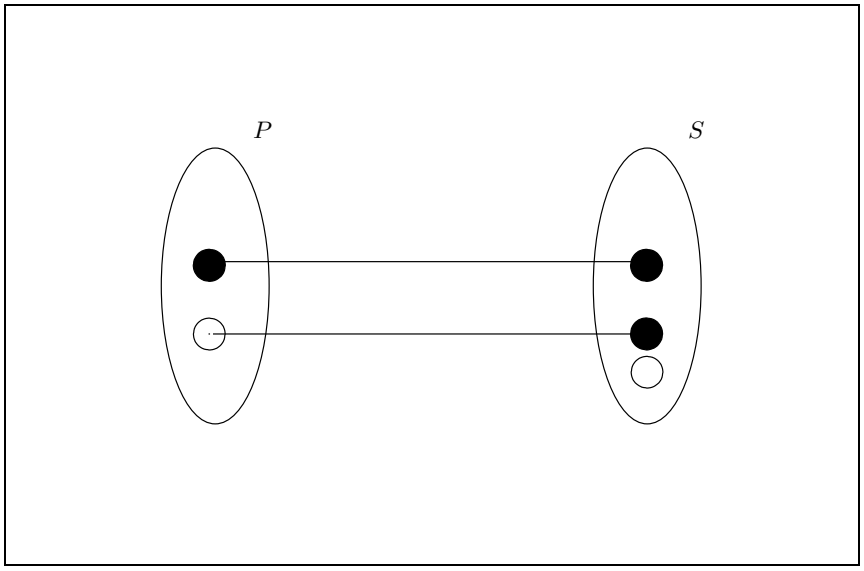

Fig. 1. Well-conditioned problem

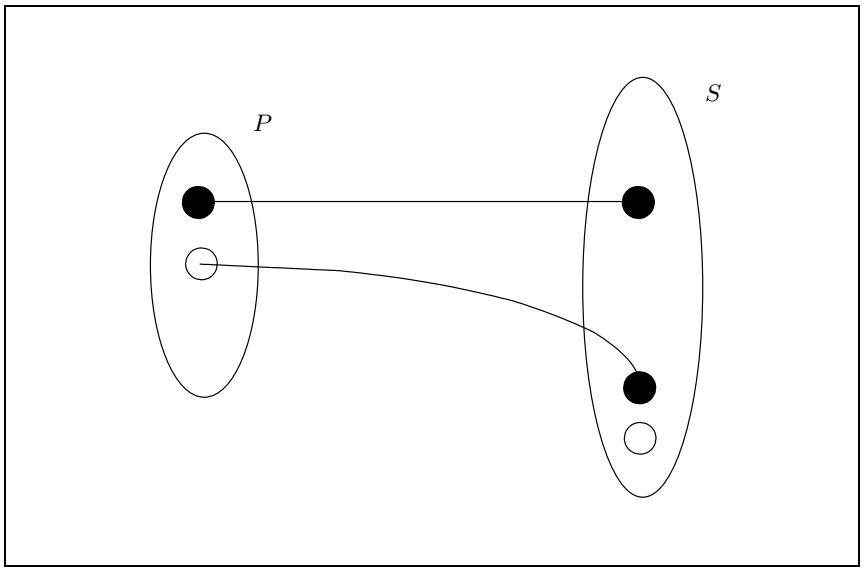

Fig. 2. Ill-conditioned problem

the relative error $\frac{\|\boldsymbol{x}-\boldsymbol{y}\|}{\|\boldsymbol{x}\|}$ may be as large as is indicated in (1). This means that even an exact, infinite-precision algorithm cannot help us avoid a large error in the solution, in the case of an ill-conditioned problem, because of the effects of the inherent uncertainty in the data (see Figure 2). It also means, however, that if we can find an algorithm that produces a solution that is the exact solution, of a problem that differs from the given problem by an amount smaller than the inherent uncertainty in the data, then the algorithm has produced an answer that is as good as the data warrants [5].

We emphasize again that a stable algorithm does not necessarily produce an answer with small error: it only produces an answer with error on the order of that which we must accept in any event, due to data uncertainty (see Figure 2, where the unfilled circle in S indicates that the method has done a good job of solving the perturbed problem, but has nonetheless produced an answer with large error). The backward error analysis does not "overcome" the problem of numerical error: it merely allows us to identify algorithms that produce errors of the same order as those that we must accept anyway.

We conclude this section with the remark that computational geometry is by no means unusual in the fact that there are theoretically exact algorithms that produce nonsense when implemented in floating point arithmetic. For example, in the case of $A\boldsymbol{x} = \boldsymbol{b}$, suppose we attempt to solve the sequence of problems

$$\begin{bmatrix} \phi(\rho) & 1.0 \\ 1.0 & 0.0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1.0 \\ 1.0 \end{bmatrix}, \quad \rho = 0.1, 0.2, \ldots$$

where $\phi(\rho) = \rho^2 - 0.01$. For $\rho = 0.1$, the correct answer is $x_1 = 1.0$, $x_2 = 1.0$, but Gaussian elimination without pivoting, as implemented in the following program, returns the answer $x_1 = 0.0$, $x_2 = 1.0$. There is no division by zero, and no overflow or underflow occurs during the execution of the implemented algorithm. In the evaluation of $A[1][1]$, however, the first term on the righthand side of the assignment statement is shifted off the end of a register and ignored.

```
  double rho = 0.1, phi = rho * rho - 0.01 ;
  double A[2][2] = {{phi, 1.0} , {1.0, 0.0}} , b[2] = {1.0, 1.0} ;
  double x[2] ;
/********* Triangulate A ***********/
  double mult = 1.0 / A[0][0] ;
  A[1][1] = A[1][1] - mult * A[0][1] ;
  b[1] = b[1] - mult * b[0] ;
/********* Back-substitute *********/
  x[1] = b[1] / A[1][1] ;
  x[0] = (b[0] - A[0][1] * x[1]) / A[0][0] ;
  cout<<" The result is: "<<x[0]<<" "<<x[1]<<endl ;
/*********************************/
  The result is: 0 1
```

3 Backward error analysis for planar convex hulls

We will now provide a parallel development for the problem of computing convex hulls of points in the plane. In this case the problem is defined [1] by a finite set of vectors $S = \{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n\}$, where each $\boldsymbol{a}_i$ lies in the plane[2]. An algorithm to compute the extreme points of S will normally select a subset $\{i_1, i_2, \ldots, i_m\}$ of the indices $\{1, \ldots, n\}$ and declare $\{\boldsymbol{a}_{i_1}, \ldots, \boldsymbol{a}_{i_m}\}$ to be the solution, but since we are envisaging the possibility of uncertainty in the problem data, we will permit any non-empty finite set of vectors $Y = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_m\}$ as a solution, where each $\boldsymbol{y}_i$ lies in the plane. It may be noted here that the use of convex-hull algorithms for problems with uncertain data is of real practical interest. For example, in [9] it was shown that Bézier curves and patches do not self-intersect (and adjacent patches do not have extraneous intersections) provided that the origin $\mathbf{0}$ is not an element of the convex hull of certain points depending directly on the Bézier control points. These points will typically contain uncertainty far greater than the perturbations introduced by the implementation of *Graham_Fortune* using even single-precision floating-point arithmetic, and use of this algorithm for this problem would therefore be entirely appropriate.

3.1 Step a: measuring inadequacy

If $\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k\}$ is a finite set of vectors in the plane, define $conv(\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k\}) \subseteq E^2$ to be the convex hull of the set of vectors. We will define the distance d between two distinct solutions Y^1 and Y^2 of the convex-hull problem to be infinite if for $i = 1$ or 2 the vectors in Y^i do not actually lie on the topological boundary of $conv(Y^i)$; otherwise, d is defined to be the Hausdorff distance between $conv(Y^1)$ and $conv(Y^2)$. Defining $d(Y^1, Y^1) = 0$, the distance d is a metric. Let $\{\boldsymbol{a}_{i_1}, \ldots, \boldsymbol{a}_{i_m}\}$ be a set of points lying on the topological boundary of $conv(\{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n\})$, and such that $conv(\{\boldsymbol{a}_{i_1}, \ldots, \boldsymbol{a}_{i_m}\}) = conv(\{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n\})$. The *error* E in a solution Y is defined to be

$$E = d(\{\boldsymbol{a}_{i_1}, \ldots, \boldsymbol{a}_{i_m}\}, Y)/M, \tag{2}$$

where M is a fixed upper bound for the absolute value of any coordinate of any point [7]. (Thus, as in [10], for a solution to be considered accurate, its points are required to actually lie on a convex polygon.) In Figure 3, the solution to the problem defined by $\{\boldsymbol{a}_1, \boldsymbol{a}_2, \boldsymbol{a}_3, \boldsymbol{a}_4, \boldsymbol{a}_5\}$ is $\{\boldsymbol{a}_1, \boldsymbol{a}_2, \boldsymbol{a}_4, \boldsymbol{a}_5\}$. The solution $Y = \{\boldsymbol{a}_1, \boldsymbol{a}_2, \boldsymbol{a}_3, \boldsymbol{a}_4, \boldsymbol{a}_5\}$ has infinite error, since $\boldsymbol{a}_3$ is not in the boundary of $conv(\{\boldsymbol{a}_1, \boldsymbol{a}_2, \boldsymbol{a}_3, \boldsymbol{a}_4, \boldsymbol{a}_5\})$, while $Y = \{\boldsymbol{a}_1, \boldsymbol{a}_2, \boldsymbol{a}_4, \boldsymbol{a}_5, \boldsymbol{y}\}$ has error as indicated by the dashed line.

It is possible to define other measures of distance between solutions of this problem, *e.g.*, we might penalize solutions with redundant points on the boundary of the convex hull.

We will use the simple criterion (2) to illustrate our point, which is that if we wish to prove rigorous theorems about the inadequacy of computed solutions, we must give a careful definition of inadequacy.

[2] We denote the points by $\boldsymbol{a}$ in order to increase the parallelism with Section 2.

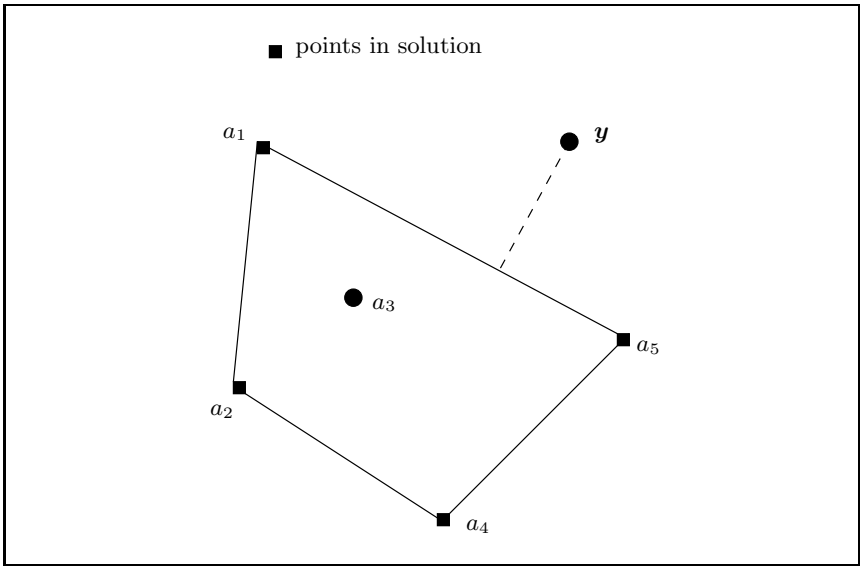

Fig. 3. Example convex-hull problem

3.2 Step b: perturbation analysis

If the input data $\{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n\}$ is uncertain, then the true problem that we wish to solve is defined by $\{\boldsymbol{a}_1 + \delta\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n + \delta\boldsymbol{a}_n\}$, where each $\delta\boldsymbol{a}_i$ is a vector in the plane. Suppose that $\frac{\|\delta\boldsymbol{a}_i\|_2}{\|\boldsymbol{a}_i\|_2} \leq \Delta$, $i = 1, \ldots, n$, where $\|\cdot\|_2$ denotes the Euclidean norm. This means that the relative error in the computed solution could be as large as $\Delta\sqrt{2}$, due to the uncertainty in the input data alone, since the Hausdorff distance between $conv(\{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n\})$ and $conv(\{\boldsymbol{a}_1 + \delta\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n + \delta\boldsymbol{a}_n\})$ has the achievable upper bound of $\Delta\sqrt{2}M$. Thus, if criterion (2) is used, $\sqrt{2}$ can be taken as a condition number for the problem of planar convex hulls.

In comparison with the linear-equations case, this is a very satisfying result: the problem of computing planar convex hulls is always well conditioned. (In this respect, the convex-hull problem is closer to the problem of computing the eigenvalues of a real symmetric matrix than to the problem of solving $A\boldsymbol{x} = \boldsymbol{b}$: the symmetric eigenvalue problem is also always well-conditioned [6, p. 93].)

3.3 Step c: stability of algorithms

Just as in the case of linear equations, there exist both unstable and stable algorithms for the planar convex-hull problem, when criterion (2) is used. In particular, it was shown in [1] that *Graham_incremental* is unstable. This algorithm should therefore not be used (just as we should not use Gaussian elimination without pivoting to solve $A\boldsymbol{x} = \boldsymbol{b}$). On the other hand, a slight modification of the *Graham_Fortune* algorithm of [7] is numerically stable, that is, the computed

answer is such that it is the exact solution for a perturbed problem for which the relative perturbation bound in problem space is at most $O(n\epsilon)$, where ϵ is the relative error of floating-point arithmetic. The algorithm uses a function called *TriangleTest* [7], first to establish lists of candidates for upper and lower chains, and secondly to decide whether or not to retain the *middle* point of possible triplets in these chains. The proof of stability depends on *both* uses of *TriangleTest* to show, for example, that slightly perturbed versions of the candidates for an upper convex chain satisfy the following condition: either they were retained and form part of an actual upper convex chain, or they were not retained but nonetheless lie above the line determined by the two points with minimum and maximum x-coordinate. The slight modification, referred to above, is to use the *a priori* bounds for finite-precision floating-point arithmetic to implement the test of a "left turn" in *TriangleTest* in a fail-safe way, so that an ambiguous point is considered to part of a "left turn", and dropped from the computed convex hull. (This modified test is described in detail in [10], and a similar test was used for another purpose in [11].)

3.4 Consequence

The consequence of this well-conditioning and stability is this: not only is it true that a stable algorithm such as *Graham_Fortune* will always produce an answer that is scarcely more in error than we should expect because of data uncertainty (this conclusion follows from stability), it is true in addition that the actual error in the computed solution is small (this follows from well-conditioning). We are in the situation illustrated in Figure 1. The overall situation is illustrated in Figure 4, where p_1 is the true problem to be solved, p_2 is the problem presented

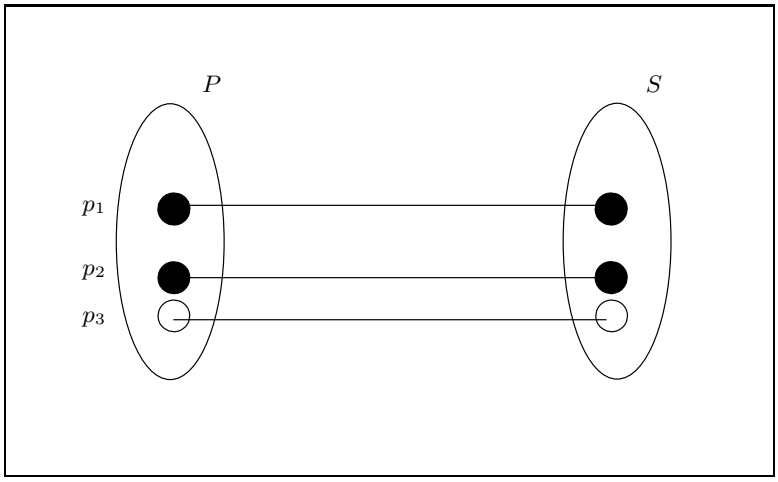

Fig. 4. Overall situation

to the method, and p_3 is the problem for which the method actually finds an exact solution. This is a pure backward error analysis, with a well-conditioned problem. Even if (2) were replaced by a criterion that rendered the problem ill-conditioned, however, it would remain true that the algorithm always produces an answer that is scarcely more in error than we should expect because of data uncertainty.

4 Conclusion

In order to prove theorems about the adequacy of numerical algorithms in computational geometry, we must define how to measure adequacy. Furthermore, in the case where data is uncertain, it is worthwhile to do a perturbation analysis, and seek stable solution methods. Carrying out these steps in the context of the planar convex-hull problem shows that the numerical difficulties described in [1] are unexceptional.

References

1. Kettner, L., Mehlhorn K., Pion, S., Schirra, S. and Yap, C. Classroom examples of robustness problems in geometric computations. *ESA '04: Proceedings of the 12th Annual European Symposium on Algorithms*. 702-713, Bergen, Norway, September 2004. Springer.
2. Graham, R. L. An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters*, 1:132-133, 1972.
3. Cohen, I. B. Revolution in Science. Harvard University Press, 1985.
4. Stewart, N. F. Science and computer science. *ACM Comp. Surveys* (27), No. 1, 39-41, 1995.
5. Kahan, W. M. A survey of error analysis. *IFIP '71*, North Holland, 1214-1239, 1971.
6. Wilkinson, J. H. The Algebraic Eigenvalue Problem. Clarendon Press, Oxford, 1965.
7. Fortune, S. Stable maintenance of point set triangulations in two dimensions. *Proceedings of the 30th annual IEEE Symposium on Foundations of Computer Science* Vol. 30, 494-499, 1989.
8. Hoffmann, C. M. and Stewart, N. F. Accuracy and semantics in shape-interrogation applications. *Graphical Models* 67, No. 5, 373-389, September 2005.
9. Andersson, L.-E., Peters, T. J. and Stewart, N. F. Selfintersection of composite curves and surfaces, *Computer Aided Geometric Design* 15, No. 5, 507-527, 1998.
10. Jaromczyk, J. W. and Wasilkowski, G. W. Computing convex hulls in a floating point arithmetic. *Computational Geometry* 4, 283-292, 1994.
11. Fortune, S. Numerical stability of algorithms for 2D Delaunay triangulations. *International Journal of Computational Geometry and Applications* (5), No. 1, 193-213, 1995.

Reply to "Backward error analysis ..."

Lutz Kettner, Kurt Mehlhorn, Sylvain Pion, Stefan Schirra, and Chee Yap

We highly welcome the opportunity to comment on D. Jiang and N. F. Stewart's paper in these proceedings. The paper refers to our ESA 2004 paper "Classroom Examples of Robustness Problems in Computational Geometry". We quote from the introduction of the full version of our paper.[1]

> The algorithms of computational geometry are designed for a machine model with exact real arithmetic. Substituting floating-point arithmetic for the assumed real arithmetic may cause implementations to fail. Although this is well known, it is not common knowledge. There is no paper that systematically discusses what can go wrong and provides simple examples for the different ways in which floating-point implementations can fail. Due to this lack of examples, instructors of computational geometry have little material for demonstrating the inadequacy of floating-point arithmetic for geometric computations, students of computational geometry and implementers of geometric algorithms still underestimate the seriousness of the problem, and researchers in our and neighboring disciplines still believe that simple approaches are able to overcome the problem. In this paper, we study simple algorithms for two simple geometric problems, namely computing convex hulls and triangulations of point sets, and show how they can fail and explain why they fail when executed with floating-point arithmetic.

We believe that the paper and its companion web page will be useful in teaching computational geometry, and that even experts will find it surprising and instructive in how many ways and how badly even simple algorithms can be made to fail. The companion web page[2] contains all sources and allows the reader to perform our and further experiments. Numerical analysts are well aware of the pitfalls of floating point computation (G.E. Forsythe: Pitfalls in Computation, or Why a Math Book Isn't Enough, Amer. Math. Monthly, 77, 931–956, 1970). Forsythe's paper and many numerical analysis textbooks contain instructive examples of how popular algorithms, e.g., Gaussian elimination, can fail when used with floating point arithmetic. These examples have played a guiding role in the development of robust numerical methods. Our examples are in the same spirit, but concentrate on the geometric consequences of approximate arithmetic. While sophisticated machinery was developed for making numerical computations reliable over the past 50 years, a corresponding machinery for geometric computation does not yet exist to the same extent. However, significant progress was made over the past 15 years. Approaches to reliable geometric computing include the exact computation paradigm, approximate algorithms, and perturbation methods. In our recent courses on geometric computing, we have used the warning negative examples of our paper to raise student awareness for the problem and then discussed these approaches. D. Jiang and N.F. Stewart's suggestion of backward analysis and the papers cited by them would be discussed under the heading approximate algorithms.

[1] http://www.mpi-inf.mpg.de/~mehlhorn/ftp/ClassroomExamples.pdf
[2] http://www.mpi-inf.mpg.de/~kettner/proj/NonRobust/