Grader programs

By R. E. Berry*

This paper examines the possibility of using automatic grading programs for checking some of the practical work of students on a Numerical Analysis course. Two existing programs for checking root-finding techniques were tested to gain experience in using grader programs. A program to check solutions to a system of *n* first order differential equations was written.

1. Introduction

Early in the sixties workers in several American University Computing Laboratories felt that the increasing number of students attending programming courses meant that too much of the Laboratory staffs' time would have to be spent in supervising and marking students' practical work. Since marking is inevitably repetitive it seemed a practical proposition to write a "grader program" which would be able to check other programs. The students' work is written as an ALGOL block and is inserted into the grader program which merely presents the block with different parameter values and checks that the students' algorithm obtains the required solutions. Grader programs have usually been used to check work arising from Numerical Analysis courses though there is no reason why they should not find application in other fields.

The suggestion that programs of this type might prove useful seems first to have been made by Hollingsworth (1960). Subsequent work by Naur (1964), and by Forsythe and Wirth (1965), has shown that grader programs can be most useful, while the more recent work of Perlis and Braden (1965) amply illustrates the wide variety of problems whose solutions may be checked automatically.

The term Grader Program is an American one, and while appropriate is possibly misleading. It does not imply that a mark or grade is given by the program to the work submitted. Some programs do give grades although there is considerable complication if any one of several methods of solution might be acceptable. It seems usually sufficient for the program to provide enough information for a mark to be quickly assigned by the students' supervisor.

In a preliminary study the programs given by Naur, and Forsythe and Wirth were compared. Since these considered the same problem (finding the root of a given function to a given accuracy in a prescribed interval) they provide a good illustration of the methods available for constructing a grader program.

Naur's method requires that the algorithm programmed by the student is included in the grader program as a labelled block, accessed by means of a switch. The grader then assigns values to variables which are global to the block, and in this way presents different problems to the student algorithm. Upon completion of the problem a jump is made from the student block to a given label within the grader program. The solution obtained is then examined and printed out with comments where necessary. Further problems are then presented until the problem list is exhausted, when the grader presents the same problems to the remaining student algorithms.

The alternative, which seems the simpler approach, is the one adopted by Forsythe and Wirth. They ask the student to write a procedure. The only restriction is that the parameters must be in a specific order. To make the work suitable for inclusion in the grader it is necessary to enclose the procedure declaration in a block whose only action is to call the procedure *Test*, which forms the main part of the grader. *Test* has two parameters one of which is the identifier of the students' procedure. Each procedure is tested by supplying it with several sets of parameters and checking the results obtained. The program terminates when all students' procedures have been tested.

These programs proved most interesting, and as a result it was decided to construct a more ambitious grader which would examine procedures for solving *n* simultaneous first order differential equations.

2. The ordinary differential equation grader program

This grader program is designed to test procedures for solving the initial value problem for sets of ordinary differential equations. This is achieved by submitting to the procedure under test various problems each of which is determined by a problem number. A check is made on the accuracy of the procedure by comparing the results obtained with the analytic results. Various other tests are performed, and details of these will be given later.

The following demand would be made of the student:

Write an ALGOL procedure to solve the *n* simultaneous first order equations of the form

$$\frac{dy_i}{dx} = f_i(x, y_1, y_2, \ldots, y_n) \quad i = 1, 2, \ldots, n$$

The procedure will have the following formal parameters in the order given.

n the number of equations;
xa the initial value of x;
xb the final value of x;

* *Computing Laboratory, University of Newcastle upon Tyne,* 1–3 *Kensington Terrace, Newcastle upon Tyne,* 2.

y	an array which will upon entry contain the initial values of the dependent variable, and upon exit must contain the solution of the equations.
fn	a procedure having three parameters *t*, *g*, *s*. The independent variable is *t*, array *g* contains values of the dependent variable. Upon exit from the call *fn* (*t*, *g*, *s*) the element *s* [*i*] of the array *s* will contain the right-hand side of the *i*th equation evaluated at the points *t*, *g*[*i*].
acc	the accuracy required of the solution. (For simplicity this was also used as the accuracy required at each stage of the calculation. The writer is grateful to the referee for his suggestion that $acc \times h/(xb - xa)$ might be a better value to use at each stage, though this might demand a very high accuracy for the last step of an integration by a method using automatic interval changing.)

An interval of integration *h* will be global to the procedure. It is not necessary that the procedures use this value of *h*. Procedures written to determine their own interval of integration should assign the value used for integration to identifier *h*. The identifiers corresponding to *xb*, *acc* may not be called by value. On exit from the procedure the array *y* should contain the required solution. The procedure must contain no output statements, and no jump must be made to any labels outside the procedure. An effort should be made to ensure that no special situations may lead to an infinite loop, division by zero, or any other failure which results in the failure of the whole program. The number of entries to procedure *fn* should be kept as low as possible. A **begin** should precede the procedure declaration, and the declaration should be followed by the call *Test* (*Solver*, [*JONES*]), where *Solver* is the name of the procedure written by student Jones. A final **end** would then make the work suitable for submission to the grader.

As few restrictions as possible are imposed on the student, and the only one which might need explanation concerns identifiers *xb*, *acc*. On exit from the student's procedure a check is made to ensure that the values of *xb*, *acc* have not been changed by the student's procedure. If the procedure calls *xb*, *acc* by value, any change of these values within the procedure could not be detected.

The program consists of three basic procedures, *Test*, *Ex*, *fnp*. Procedure *fnp* is local to *Ex* which is local to *Test*. There are three parameters *t*, *g*, and *s* in a call of *fnp*. *fnp* is the actual parameter which is substituted for the students formal parameter *fn*. Its action depends upon an integer which is the problem number, and different problems are specified by altering this integer.

The student is attempting to solve a system of first order equations, a typical one of which might be $y[i]' = f(x, y[i])$. In a call of $fnp(t, g, s)$ t has the value of *x* the independent variable, array *g* contains the current values of the elements *y*[*i*], and upon exit array *s* will contain the right-hand side of the equations evaluated at these points. Upon each entry the count of the number of calls of *fnp* is increased by one, and this count is examined to ensure that it does not exceed the limit set on the number of calls. If this limit is reached a warning is printed out to this effect and the value of the independent variable at the last entry to *fnp* is printed out. A warning is also printed out if the independent variable is outside the range of integration. In either eventuality the program proceeds to the next problem. Normally the limit on calls of *fnp* will not be reached, and the program will jump to a switch with the problem number as index, and thus evaluate the appropriate right-hand sides.

Procedure *Ex* has eight parameters:

eqns	the number of equations being solved; it can also be the order of the equation being solved.
inx	the initial value of *x*.
outx	the final value of *x*.
reqacc	the truncation error allowed.
interval	the initial value of the interval of integration.
solution	the analytic solution for comparison purposes.
limit	the upper limit on the number of calls of *fnp*.
exno	the problem number.

Four of the above parameters correspond to four of the parameters which the student's procedure must use. The first action of *Ex* is to assign to the parameters used in the call of *Solver* (the student's procedure) the values of the parameters of the current call of *Ex*. In this way each call of *Ex* presents a new problem to the student's procedure. By using a switch list with the problem number as index the initial values of *y* associated with a particular problem are set up. Copies of the initial values in the first and last elements of array *y* are now stored elsewhere, so that by comparison at a later stage it may be established that the solution has or has not been assigned to array *y*. Immediately before the student's procedure *Solver* is called, entry is made to the real procedure *rtime* which stores the time. There are two means of exit from *Solver*; one is the normal exit after which the examiner program continues, and the other is via *fnp* when the limit on function calls is reached. In either eventuality a further entry is made to *rtime* which results in the time elapsed since the initial entry being recorded. By this means the time spent by *Solver* in obtaining a solution is known.

The program continues by making a series of checks. Both the final value of *x* and the required accuracy are tested in case they have been changed, and array *y* is examined to ensure that the solution has been stored there. Should any of these tests fail then the program prints an appropriate warning and continues by considering the next student's work. If the program runs normally then at this stage on the first entry to *Ex* a series of column headings is printed. Upon each succeeding call of *Ex* appropriate entries are made in these columns. The output of results marks the end of *Ex*.

Procedure *Test* has two parameters: the student's procedure, and a text string which is the student's name.

It contains two procedures to simplify output, and it also contains the procedures *rtime*, and *Ex*. When called, *Test* works through all the calls of *Ex*, i.e. the list of problems. Upon completion of the list of problems the total number of calls of procedure *fnp* is printed. The program continues by considering the next student's work.

Nine problems are included in the program, though provision is made in the switch declarations for the inclusion of twenty. While it can never be claimed that a procedure which successfully completes a set of problems can thus produce the correct solution to every problem, it is felt that the examples included give a reasonable indication of the performance of a procedure. One of the practical drawbacks in presenting a difficult set of problems for test purposes is that the time taken by the program to examine several procedures increases considerably. Since large amounts of machine time are not readily available it is prudent to arrange a set of examples whose execution can be accomplished in reasonable time by most procedures.

At the time when this work was undertaken students were not available to produce testing material. It was thus necessary for the writer to prepare several procedures himself, and also to adapt procedures constructed by other members of the laboratory. This had the effect of ensuring that the solution to the question posed was a reasonable one to attempt, and of providing a set of procedures with which students' work could be compared in future. The unfortunate result of the writer having to prepare all the material himself was that no ambiguities in the problem set would be brought to light. He would also be unlikely to make the mistakes which he had designed the grader program to detect.

The methods of Adams–Bashforth, Milne–Simpson, Runge–Kutta, and Kutta–Merson were programmed, and an interesting comparison of methods was made possible by the grader program.

As a by-product it became clear that the grader program was a very useful way of "debugging" procedures, and of testing their actions in a variety of circumstances.

It only remains to give the grader a thorough testing by submitting students' work for grading; it is hoped to do this shortly in an undergraduate class. In addition to the lecturers' comments, the students views of the output are of interest. While the grader is primarily concerned with providing supervisors with information concerning students' work, the program output should also be of some help to the student if this is necessary. The programs of Naur, and Forsythe and Wirth are deficient in this respect. Both programs print out a message to the effect that the student's program has exceeded the limit on the function calls, but neither prints out the value of the argument at termination, the expected solution, nor the student's solution. It is desirable to print sufficient information to help the student in case of failure, and it is hoped that the differential equation grader is satisfactory in this respect.

3. Grading

One of the difficulties which was very well illustrated during the work was that of comparing different methods of solution, so that a fair grade may be given. The case in point was the comparison of multi-step with one-step methods. The combination of the Runge–Kutta starting mechanism and multi-step methods worked satisfactorily for most problems. There was, however, one situation in which it was quite unsatisfactory. If the last step which the multi-step procedure is required to take is less than the interval being used for integration, then the procedure cannot produce an accurate answer. If an interpolation procedure was entered in this eventuality, then this is liable to introduce error, and thus invalidate any comparison. A much better solution would be to have a starting mechanism which considered the whole range of integration and divided it by appropriate factors until a satisfactory interval of integration was obtained. This would produce a multi-step procedure which would cover the situation described. It would also produce a multi-step procedure which did not start with the same interval as the remainder of the procedures being tested. The essence of a grader program is to present to each procedure or labelled block a similar set of conditions and to measure the reaction of the procedures to the given situation. Given this, then a procedure which disregards one of the initial conditions invalidates any comparison with other procedures being tested. This accent on comparison must be made since a grading mechanism such as the one used by Forsythe and Wirth is in fact a strict comparison of procedures.

The foregoing points are made to emphasize the fact that comparison of work is not by any means straightforward. Thus in trying to present a problem to which different methods of solution are available, the possibility of grading successfully the work presented is remote. Grading is only realistic when the methods of solution being tested are the same. The Runge–Kutta method used half as many function calls again as the Adams–Bashforth, and was marginally slower, while the accuracy was comparable. The problem is to give a fair grade to each. The fairest conclusion to be made from this is that while a grading system could prove satisfactory for marking variations of the same method, a program without a grading mechanism is a more realistic prospect for testing different methods of solution. The alternative is to develop a "weighted grading system", but it is doubtful whether this would be worthwhile.

4. Discussion

Grader programs would be extremely simple to use in a software system which did not demand re-compilation of the whole program when further material was added. Students' work might then be added simply. However, in many systems, including the one used, it is necessary to splice procedures for testing into the existing paper tape program and recompile the resulting program

Punched cards would eliminate the splicing but not necessarily re-compilation. A whole series of procedures might be assembled on paper tape, of which just one is required to be run for test purposes.

To cater for this kind of eventuality the following modification is suggested. The procedures could be made into labelled blocks such as appear in Naur's program, but instead of having the switch index derived from a **for** statement (so that the program tests all work submitted), link the switch index with a **read** statement. In this way the procedures to be tested could be specified on a data tape. A similar system could be used for the list of problems, so that only specified problems need be included in any test. This would be beneficial in two ways. In the first instance a student wishing to correct his program which might have failed on one problem only, could state that it was only necessary to check the procedure for this problem. Secondly it would enable the person in charge to present different problems on different runs and thus minimize the chance of the criticism that the students' work would become problem oriented. An alternative to the latter suggestion would be to make the parameters in the procedure call *Ex* variable, so that they may be read in. Changing the range of integration for instance would give one a variable problem. All this, however, is departing further and further from the concept of the grader or examiner as a fixed program, though this might prove a most worthwhile line of development.

These programs are limited in the respect that each can only test one particular problem, though this disadvantage can be offset by preparing suites of programs to cover many different problems. It would not be difficult to select a set of problems which were appropriate to several courses and thus save duplication of work. Apart from this basic set of problems, Master Programs could be written to cover a wide variety of problems. In this respect the section by Perlis and Braden concerned with problems should be consulted since it contains a large selection of problems which might prove appropriate. Perhaps the best example of the situation in which a grader program might prove useful is in testing sorting procedures. The approach which suggests itself is to declare procedures *COMPARE*, and *CHANGE* and others appropriate to magnetic tape, disc, and card handling, global to the sorting routines. These could be called as required, a count could be kept of the number of calls, and in this way an estimate might be obtained of the relative efficiencies of the procedures. By presenting a sample data tape to the sorting procedures the one proving most efficient with this data may be selected. This could in practice prove most useful. It also shows that grader programs have a wider range of application than might initially be apparent.

As has been mentioned before, American workers are both active and enthusiastic in this field. The writer feels, however, that enthusiasm must be tempered by realism. The demand for automatic grading programs or examiner programs is not liable to be considerable in this country, simply because our laboratories and their associated courses are smaller. There is one thing at least which commends their use. They provide an incentive to work. Naur, for instance, claims that "many of the students were positively excited about the problem". The writer found that interest rather than excitement was aroused, and those people who did prepare work for submission found the challenge of writing the best procedure an incentive to work. If this kind of interest can be aroused in a class of students it will be to their own benefit. It is well known that a good teacher is one who can generate and maintain the interest of a class in the work it is doing. If a grader program provides a means of doing this, is this not some recommendation for its use? In the writer's opinion it is.

It is regretted that considerations of space prevent the inclusion of the program with this paper. Interested readers may obtain the full printout of the ALGOL program by application to the author.

A sample of the output produced by the program is given below. The output headings are as follows:

EX	the problem number given as the last parameter in the call of procedure *Ex*.
INTERVAL	the length of the last step taken by the procedure.
F CALLS	the number of entries made to procedure *fnp*.
SOLUTION	the solution obtained by the procedure.
ERROR	the difference between the analytic solution (given as the sixth parameter in the call of *E*x) and the solution obtained by the procedure. When the analytic solution is unknown the dummy $_{10}{+}20$ is inserted instead of the solution in the call of *Ex*. A row of dots is then printed in the error column.
TIME	the time in seconds taken by a procedure to complete a problem.

```
D.E. PROCEDURE TEST

RUNGE–KUTTA MODB

EX   INTERVAL   F CALLS    SOLUTION     ERROR     TIME
 1   0·100000      48      0·600420   0·000004   0  4
 2   0·100000      92     −2·000001   0·000001   0  6
 3   0·050000     148      0·693175   0·000027   0 15
 4   0·050000     196      0·444762   0·000317   0 14
 5   0·392699      40      0·999606   0·000394   0  3
 6   0·025000     352      0·203003   ........   0 27
 7   0·200000      28      0·000000   0·000000   0  2
 8   0·050000     172    221·264956   0·000256   0 10
 9   0·025000     344      2·000001   0·000001   0 21
      TOTAL CALLS 1420
```

ADAMS BASHFORTH 3

EX	INTERVAL	F CALLS	SOLUTION	ERROR	TIME
1	0·100000	44	0·600430	0·000006	0 5
2	0·100000	57	−2·000000	0·000000	0 6
3	0·050000	113	0·693393	0·000246	0 16
4	0·050000	142	0·445388	0·000944	0 15
5	0·392699	44	1·000012	0·000012	0 3
6	0·025000	206	0·203003	....	0 24
7	0·200000	27	−0·000000	0·000000	0 1
8	0·050000	122	221·283361	0·018661	0 9
9	0·025000	197	2·000000	0·000000	0 17
	TOTAL CALLS	952			

SCRATON

EX	INTERVAL	F CALLS	SOLUTION	ERROR	TIME
1	0·100000	50	0·600425	0·000001	0 6
2	0·050000	209	−2·000000	0·000000	0 21
3	0·200000	127	0·693154	0·000006	0 17
4	0·200000	201	0·444445	0·000001	0 20
5	0·392699	42	1·000060	0·000060	0 4
6	0·400000	174	0·202739	....	0 18
7	0·800000	15	0·000000	0·000000	0 1
8	0·050000	113	221·264633	0·000067	0 8
9	0·050000	297	1·999996	0·000004	0 22
	TOTAL CALLS	1228			

MILSIM

EX	INTERVAL	F CALLS	SOLUTION	ERROR	TIME
1	0·100000	36	0·600426	0·000002	0 5
2	0·100000	40	−2·000000	0·000000	0 5
3	0·050000	85	0·693212	0·000065	0 14
4	0·050000	104	0·444776	0·000331	0 14
5	0·392699	42	0·999939	0·000061	0 3
6	0·025000	350	13·911633	....	0 45
7	0·200000	25	−0·000000	0·000000	0 1
8	0·050000	85	221·269402	0·004702	0 8
9	0·025000	200	2·005810	0·005810	0 20
	TOTAL CALLS	967			

K–M ALTERNATIVE v = 0·7

EX	INTERVAL	F CALLS	SOLUTION	ERROR	TIME
1	0·018336	35	0·600419	0·000005	0 3
2	0·162549	40	−1·999999	0·000001	0 3
3	0·036543	157	0·693154	0·000006	0 18
4	0·118173	166	0·444459	0·000014	0 15
5	0·372465	28	0·999944	0·000056	0 2
6	0·007261	100	0·202837	....	0 9
7	0·800000	10	0·000000	0·000000	0 1
8	0·009049	220	221·264705	0·000005	0 15
9	0·034635	164	1·999997	0·000003	0 10
	TOTAL CALLS	920			

KUTTA–MERSON

EX	INTERVAL	F CALLS	SOLUTION	ERROR	TIME
1	0·200000	30	0·600418	0·000006	0 3
2	0·200000	55	−2·000000	0·000000	0 4
3	0·100000	167	0·693154	0·000006	0 19
4	0·200000	186	0·444456	0·000011	0 16
5	0·392699	28	0·999945	0·000055	0 2
6	0·200000	106	0·203246	....	0 9
7	0·800000	15	0·000000	0·000000	0 0
8	0·025000	222	221·264720	0·000020	0 15
9	0·050000	209	1·999999	0·000001	0 14
	TOTAL CALLS	1018			

Acknowledgement

This paper forms the greater part of a dissertation prepared as part of an M.Sc. course. Financial support for the course was obtained from the Science Research Council. My supervisor throughout the work, Mr. L. B. Wilson, deserves special thanks, and I am also grateful to other members of the Laboratory staff for their help and encouragement in this work.

References

Forsythe, G. E., and Wirth, N. (1965). "Automatic Grading Programs", Technical Report: CS17, Stanford University.

Hollingsworth, J. (1960). "Automatic graders for programming classes", *Comm. of the Assoc. Comp. Mach.*, Vol. 3, p. 528.

Naur, P. (1964). "Automatic grading of students Algol programming", *B.I.T.*, Vol. 4, p. 177.

Perlis, A. J., Braden, R. T. (1965). *An introductory course in computer programming*, Monograph No. 7—Discrete System Concepts Project, Carnegie Institute of Technology, p. 81.