TOWARDS ROBUST GENERAL PURPOSE SIMULATION SOFTWARE

François E. Cellier Peter J. Moebius

Institute for Automatic Control
The Swiss Federal Institute of Technology Zurich
ETH - Zentrum
CH-8092 Zurich
Switzerland

The aim of this paper is to present methods to improve the robustness of à general purpose run-time system for digital simulation of continuous systems.

This aspect of simulation software robustness is just one of several possible aspects. One could as well discuss the robustness of a language to formulate simulation problems (simulation language) or the robustness of a compiler for such a language (simulation compiler). Such aspects will, however, not be considered in this article.

The run-time system robustness is a very important aspect which has not sufficiently been taken into account in the development of existing simulation software. The aim of this paper is to specify demands rather than to present optimal solutions. Solutions are presented from an engineering point of view. They improve the run-time system robustness to some extent, but better solutions can certainly be found. It is hoped that some of the Numerical Mathematicians attending this Conference may find these problems of interest and may develop better solutions to them in the future. This is one of the main reasons why the authors decided to present this material at a Conference of Numerical Mathematics rather than at a Conference of Simulation Techniques.

I) INTRODUCTION:

A simulation run-time system can be robust in two senses:

a) The user should never be required to provide any kind of information which he does not have at his disposal. He should be able to concentrate on those factors which have to do with the statement of his problem, and should be relieved, as much as possible, of all aspects which have to do with the way his problem is executed on the machine. He should be able to describe his system as easily as possible in terms which are closely related to his common language, but must not be required to provide a step-size for the numerical integration or to specify the integration algorithm to be used.

b) The run-time software itself must be able to check whether the produced time responses are "correct" (within a prescribed tolerance range). The user, normally, has a more or less precise (although often not mathematically formulated) knowledge of the system he is investigating. He has, however, hardly any "insight information" into the tool he is using for that task. He is, usually, very credulous (the obtained results must be correct because the computer displays 14 digits!), and he has no means to judge the correctness of the produced results. For this reason, it is vital that each algorithm in the system has its own "bell" which rings as soon as it is unable to properly proceed. Under no circumstances are incorrect results allowed to be displayed to the user.

II) AUTOMATED SELECTION OF INTEGRATION ALGORITHMS:

A huge step towards robust simulation software has been taken in the development of step-size controlled integration algorithms. Before these algorithms existed, the user of digital simulation software was required to supply information concerning the step-size to be used -- an information item which he clearly did not have at his disposal. Now, the user can simply provide a tolerance range for the accuracy of the results. This is identical to requesting the user to identify the smallest number in his problem which can be distinguished from zero. This question can certainly be answered by any user, independently of whether he is an expert in Numerical Mathematics or not, since it is closely related to the physics of the problem, and not to the numerical behaviour of the algorithm.

Available simulation software, up to now, usually offers a comprehensive selection of different integration algorithms. It does, however, not tell the user which would be the most appropriate one for his particular application. In this way, the user is again confronted with making a decision on something he does not really understand. Experience has shown that the majority of the average users always operate with the default integration method implemented in the package which, in most cases, is a Runge-Kutta algorithm of 4th order. Since he does not know what to specify, he simply ignores that question, and after some time of using the software he has even forgotten that the language provides him with the facility to select among different integration algorithms. So far, no integration algorithm could be found which would be able to handle all kinds of problems equally well, and it is more than doubtful whether such an algorithm could be found at all. The user, who does not make use of the facility to select among different integration rules, will, consequently, often waste a lot of computing power. Although much research has been devoted to the development of different integration methods for the different classes of application problems [3,8], the user has, however, no means to easily determine, from the state space description, the problem class to which his particular application belongs. For this reason, the selection of the appropriate integration algorithm should also be automated.

For this purpose, we try to extract features from an application problem during its execution which are supposed to characterize the numerical behaviour of that particular problem as completely as possible. These features are then combined in a feature space in which we can identify specific clusters for which a particular integration method is optimally suited. The proposed methodology for the solution to this problem originates from pattern recognition.

What features may be used for this purpose? A first feature can be associated with the accuracy requirements for the problem. It can be found that the CPU-cost to execute a particular problem depends on the required relative accuracy. Low order algorithms are appropriate for the treatment of systems from the "gray-" and "black box" area where the available data and models are so vague that a precise numerical integration does not make much sense, whereas higher order algorithms are appropriate for the handling of systems form the "white box" area, e.g. from celestrial mechanics. Since the user is requested to specify the wanted relative accuracy, this feature can be extracted from the input data.

Multi-step methods require more CPU-time during their initial phase, but are more economic than one-step methods if integration goes on over a longer interval of simulated time. This can be explained by the fact that one-step methods are self-starting whereas multi-step methods need to be initialized. This leads to a second feature. Since the integration has to be restarted after "event times" (instants at which some state trajectories are discontinuous), multi-step integration is in favour for purely continuous problems or for problems with few event times, whereas one-step integration is appropriate for combined (continuous/discrete) problems where discrete events occur with a high density.

Each integration algorithm has associated with it a domain of numerical stability. The stiffer a particular problem is (the more the different eigenvalues (λ_i) of the Jacobian are separated), the smaller the step-size (h) must be in order to keep all (λ_i*h) within the stability region of the algorithm.[1] Fortunately, special integration algorithms could be found for which this restriction no longer holds (A-stability, stiff stability). If such an algorithm is used, the step-size need not be reduced due to restrictions imposed by stability demands, but is determined exclusively by the requirements of accuracy. For this reason the eigenvalue distribution of the Jacobian determines a third feature which must influence our decision as to which integration algorithm to use for the execution of a particular problem.

These three features can now be combined to a feature space as depicted in fig.1.

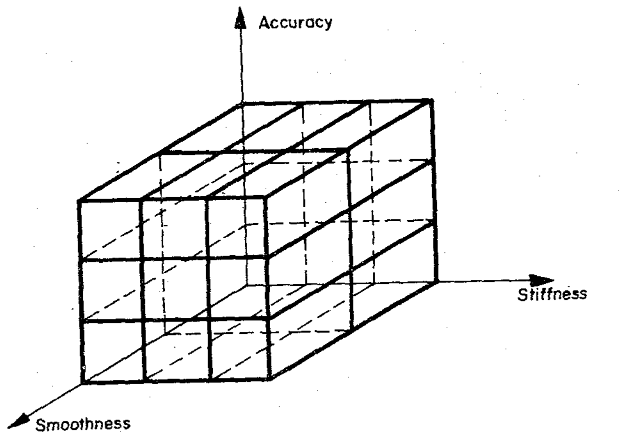

Fig.1: Feature space for selection of integration algorithm

18 clusters have been distinguished in fig.1, and fig.2a and 2b show integration rules which can be associated with them.

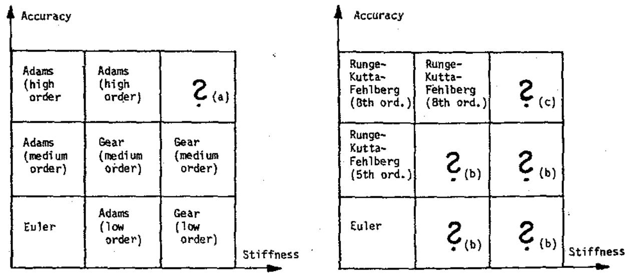

Fig.2: Integration algorithms to be used for smooth (left) and non-smooth (right) problems

As can be seen from fig.2, some of the assignments are still open.

Concerning (a): A Gear algorithm [3] would be appropriate, but it should be of about 8th order (at least for a CDC 6000 series installation -- this depends on the length of the mantissa), whereas, in the Kahaner implementation [6] we use, we have only up to 5th order available.

Concerning (b): For these clusters, a one-step algorithm with a stiff stability behaviour would be most suitable. So far, we have experimented with IMPEX-2 [9], and with DIRK [1], but the programming style of these algorithms, as they are at our disposal at the moment, is not sufficiently elaborate to allow for a fair comparison with the extremely careful and sophisticated Kahaner implementation of the Gear algorithm.

Concerning (c): For this cluster, a high-order stiffly-stable one-step method would be most appropriate. Such an algorithm is, however, unknown to date.

As a matter of fact, the feature space as depicted in fig.1 is still a simplification. To show this, let us consider a system with complex dominant poles close to the imaginary axis. For the treatment of such a system, a stiffly-stable method cannot be properly applied. The system has, however, fast transients, making a Runge-Kutta algorithm not suitable either. Thus, these types of systems, which are called "highly oscillatory" systems, will again require special methods (like stroboscopic methods) for efficient handling (C.W.Gear: private communication). This establishes a fourth feature which is to be used for the determination of the integration method. The reason for the primary simplification lies in the fact that a 3-dim. feature space can be graphed easier than a 4-dim. one (!).

The information provided by these four features is sufficient to determine the best suited integration algorithm for most application problems.

There are even two more features which can be extracted from the eigenvalue distribution of the Jacobian. These are used for other purposes, and will be presented in due course.

So far we have defined features, and we have associated integration methods with them. It remains to determine how these features can be extracted from the state space description of the problem. The first feature (relative accuracy) is user specified on data input. The second feature (smoothness) could also easily be user provided. It is, however, a simple task to detect automatically whether a problem is continuous or combined. If the problem turns out to be combined, one can count the number of event times during a certain period of simulated time, and decide then whether it is a smooth or a non-smooth combined problem. Concerning the third and fourth features (stiffness / highly oscillatory behaviour) one has to compute the Jacobian out of the state space description of the problem. This can either be numerically approximated at run-time, or one can compute it algebraically by means of formulae manipulation at compile-time. This is rarely done by available simulation compilers but it is feasible, and seems to be a promising approach. The wanted features can now be computed by estimating the critical eigenvalues. The eigenvalues with the largest and smallest absolute values can be approximated by appropriate matrix norms, whereas the real part of dominant poles can be found by estimating the "margin of stability" [5,7]. However, since several quantities are needed, we found that it is in most cases faster to compute the whole set of eigenvalues directly by use of the EISPACK software [2]. The required CPU-time turned usually out to be neglectable compared to the time spent for numerical integration.

III) ADAPTIVE SELECTION OF INTEGRATION ALGORITHMS:

In a nonlinear case, the Jacobian will usually be time dependent, and, with it, also its eigenvalues. Since the classification is, in general, defined for linear systems, it may well be that in a nonlinear case it would be best to assign the integration algorithm dynamically to the problem. For this purpose, one has to recompute the eigenvalues from time to time to find out whether the integration method in use is still appropriate. It seems a good idea to recompute the eigenvalues as soon as the step-size, which is controlled by the integration rule, has changed by an order of magnitude, but not before a minimum time span of maybe 0.01 times the run length has elapsed. This can then be used to obtain an adaptive selection of the appropriate integration scheme.

IV) VERIFICATION OF SIMULATION WITH RESPECT TO MODELING:

Let us assume that a valid model has been derived from the physical system under investigation, and let us question what assurance we have that the time responses which we obtain through simulation represent the (valid) model correctly.

For a variable-step integration method being used, we normally trust in the step-size control mechanism which is equivalent to confiding in the error estimation procedure. This will usually be justified as long as the local error which we control can be used as a valid estimate for the global error in which we are interested.

Experience has shown that local errors will not usually accumulate as long as the system is numerically stable. In a nonlinear case, it may, however, happen that some eigenvalues "walk" into the right half-plane for a short period of simulated time. Let us consider, as an example, the well known Van-der-Pol equation. Fig.3 shows how the eigenvalues "walk around" during one limit cycle.

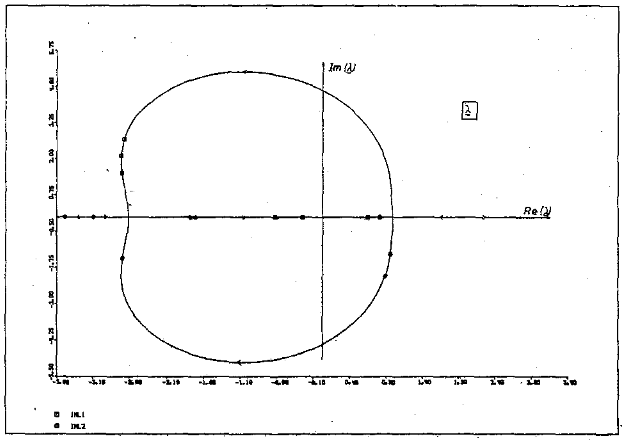

Fig.3: Eigenvalue movement of the Van-der-Pol equation during one limit cycle

As can be seen, the system becomes periodically unstable. During such time intervals, errors will accumulate, and, consequently, we must be careful in the interpretation of the obtained results. This fact should be reported by the software to the user.

For this purpose, we define another feature (stability). A variable STAB is set equal to zero when all eigenvalues lie in the left half-plane and is equal to one as soon as at least one of the eigenvalues moves into the right half-plane.

$$\mathrm{STAB} = \begin{cases} 0.0 : \mathrm{Re}\{\lambda_i\} < 0.0 \; ; \; i=1,\ldots,n \\ 1.0 : \text{otherwise} \end{cases}$$

We now collect statistics on STAB as for time-persistent variables. In this way we obtain the integral of STAB over time divided by the run length:

$$\mathrm{FF5} = \frac{1.0}{|t_{fin} - t_{beg}|} * \int_{t_{beg}}^{t_{fin}} (\mathrm{STAB})dt$$

The fifth feature (FF5) is a real number between 0.0 and 1.0. If it is close to 0.0, the results obtained by simulation have a good chance to be reliable. If it is close to 1.0, the obtained results are most probably nonsense, and they must be cautiously verified.

V) VALIDATION OF THE MODEL WITH RESPECT TO THE SYSTEM UNDER INVESTIGATION:

Another non-trivial question is whether a model, for given experimental conditions, properly represents the system under investigation. There exist several possible means to help the user in this respect (like asking him to supply dimensions for all variables in the system to enable an automated dimensional analysis). In this section we want to show that the eigenvalue distribution can also help to answer this question to some extent.

It has been shown in [4] that only those eigenvalues of a matrix can be properly computed which fulfil the following inequality:

$$|\lambda_i| > \sigma_1 * \varepsilon^{(1/n)}$$

where σ_1 is the largest singular value of the matrix, ε is the machine resolution (e.g. $\sim 10^{-14}$ on a CDC 6000 series installation), and n is equal to the order of the model. For higher order models $\varepsilon^{1/n}$ approaches 1.0 and hardly any eigenvalues will then be properly computable. Smaller eigenvalues can take any value and small modifications of the elements of the matrix can place them almost anywhere within the band of incertainty.

If we now assume that the matrix under investigation is a Jacobian of a state space description for a real physical process, then the elements of the Jacobian are extracted from measurements, and cannot be computed more accurately than $\hat{\varepsilon}$, which is a relative accuracy of measurement. We, therefore, must assume that, within that relative accuracy $\hat{\varepsilon}$, the elements of the matrix can take any values. In this case, we must also assume that eigenvalues of the Jacobian which do not fulfil the more stringent inequality:

$$|\lambda_i| > \sigma_1 * \hat{\varepsilon}^{(1/n)}$$

can take any value within that broader band, although they can be much more accurately computed as soon as any particular values have been assigned to all elements of the Jacobian. This means that as soon as there exist eigenvalues for which the second (more stringent) inequality does not hold, small variations in the systems parameters which lie within the inaccuracy of the measurement can make the model non-stiff or stiff or even unstable. Physically seen, these eigenvalues correspond to merely constant modes which could as well be eliminated from the equation set resulting in a model reduction. Numerically seen, these eigenvalues can lead to accumulation of errors so that these modes can drift away over a longer span of simulated time, again resulting in incorrect simulation trajectories.

Together with the eigenvalues, we compute the following quantity:

$$\mathrm{BORD} = \sigma_1 * \hat{\varepsilon}^{(1/n)}$$

and the number k indicating those eigenvalues whose absolute value is smaller than BORD:

$$k = \sum_{i=1}^{n} (j_i) \; ; \; j_i = \begin{cases} 0: & |\lambda_i| \geq BORD \\ 1: & |\lambda_i| < BORD \end{cases}$$

k represents an integer between 0 and n.

We now collect statistics on the quantity (k/n) as for time-persistent variables, and obtain a sixth feature:

$$FF6 = \frac{1.0}{|t_{fin} - t_{beg}|} * \int_{t_{beg}}^{t_{fin}} (k/n)dt$$

Also the sixth feature (FF6) is a real number between 0.0 and 1.0. If it is close to 0.0, the model has some chance to be valid. If it is close to 1.0, the model is most probably invalid, and it should be further investigated.

Evaluation of features FF5 and FF6 requires computation of the eigenvalues of the Jacobian once per integration step. Since this can be expensive, it should not be done automatically, but the user must have a switch at his disposal to turn computation on and off. In this way he can use these features during the development of a new model, whereas he can turn computation off during production runs.

VI) DETERMINATION OF CRITICAL STATES:

In section V we have discussed the case where single eigenvalues were situated close to the imaginary axis, and we have seen that in such a case it may be possible to reduce the order of the model.

It is, however, as interesting to discuss the oposite case where single eigenvalues are located in the λ-plane far to the left. We call these modes the "critical states" of the system. Very often one is not really interested in these fast transients. In such a case one could eliminate these modes from the equation set. If the fast transients are important one could at least try to utilize special integration techniques (like using singular perturbations) to expedite integration.

One can, of course, again compute the eigenvalue distribution for the solution of this problem. However, it is not always easy to see which state equations are responsible for such an eigenvalue. For this reason we recommend the following procedure.

We reserve an integer array i_n of length n which is initialized to zero. Each time an integration step has to be rejected due to accuracy requirements not being met, we add 1 to each element of the array $i_n(k)$ for which the accuracy is not met. This implies, of course, that the local truncation error is estimated for all state variables independently.

At the end of the simulation run we divide each element of the array by the total number of rejected integration steps and obtain in this way another set of n real numbers between 0.0 and 1.0. Elements with the largest value indicate critical states.

REFERENCES:

[1] R.Alexander: (1977) "Diagonally Implicit Runge-Kutta Methods for Stiff O.D.E.'s". SIAM Journal on Numerical Analysis, vol. 14, no. 6 : December 1977; pp. 1006 - 1021.

[2] B.S.Garbow, Boyle J.M., Dongarra J.J., Moler C.B.: (1977) "Matrix Eigensystems Routines - EISPACK Guide Extension". Springer Verlag, Lecture Notes in Computer Science, vol. 51.

[3] C.W.Gear: (1971) "Numerical Initial Value Problems in Ordinary Differential Equations". Prentice Hall, Series in Automatic Computation.

[4] G.H.Golub, Wilkinson J.H.: (1976) "Ill-Conditioned Eigensystems and the Computation of the Jordan Canonical Form". SIAM Review, vol. 18, no. 4 : October 1976; pp. 578 - 619.

[5] P.Henrici: (1970) "Upper Bounds for the Abscissa of Stability of a Stable Polynomial". SIAM Journal on Numerical Analysis, vol. 7, no. 4 : December 1970; pp. 538 - 544.

[6] D.Kahaner: (1977) "A New Implementation of the Gear Algorithm for Stiff Systems". Unpublished private communication. For further detail contact: Dr. David Kahaner, University of California, LosAlamos Scientific Research Laboratory, Contract W-7405-ENG-36, P.O.Box 1663, LosAlamos NM 87545, U.S.A..

[7] M.Mansour, Jury E.I., Chapparo L.F.: (1978) "Estimation of the Margin of Stability for Linear Continuous and Discrete Systems". Internal Report no. 78-01. To be ordered from: Institute for Automatic Control, The Swiss Federal Institute of Technology Zurich, ETH - Zentrum, CH-8092 Zurich, Switzerland.

[8] J.D.Lambert: (1973) "Computational Methods in Ordinary Differential Equations". John Wiley.

[9] B.Lindberg: (1973) "IMPEX 2 - A Procedure for Solution of Systems of Stiff Ordinary Differential Equations". Report TRITA-NA-7303 . To be ordered from: The Royal Institute of Technology, Stockholm, Sweden.