Memory conserving efficient methods for solving large sets of stiff differential equations

by CRUIA-CATALIN ROMAN, DAVID GARFINKEL and CARL B. MARRACH
University of Pennsylvania
Philadelphia, Pennsylvania

ABSTRACT

Solution of large systems of stiff differential equations by the most widely used method, that of Gear, is limited by its requirement for the presence in memory of a Jacobian matrix, which may become intolerably large. Methods of alleviating this situation, believed to be broadly applicable, have been worked out with large metabolic models. One general tactic is to remove from the set of differential equations elements which can be represented instead by algebraic equations: rate equations for enzymes and equilibrium relations for very rapid reactions. Another general tactic is to store efficiently only a part of the sparse Jacobian matrix containing the non-zero elements, which requires the presence of a preprocessor, using algorithms which are given in the paper. The Jacobian matrix may sometimes be represented by a diagonal approximation; this works better with the algebraic methods. Examples of the savings are given. These methods have been incorporated into BIOSSIM, a machine-independent simulation language designed for large biological systems, and in our hands have considerably reduced the cost and difficulty of solving large systems of stiff differential equations.

Methods of solving “stiff” differential equations have received considerable attention within the last few years, and important improvements have recently been made. The most important of these is the predictor-corrector method of Gear. This appears to be the most widely used “stiff” differential equation solver, and is widely considered to be the best one. This implicit method requires that the Jacobian matrix of the differential equation variables be present in core memory. Unfortunately the size of this matrix can become intolerably large when modeling complex systems. We describe here an economical method of computing the behavior of large systems based largely on modifications of Gear’s method, which we have thus far applied to biochemical systems. The most important such modification requires the presence in memory only of the non-zero elements of the Jacobian matrix, usually a small part of the total for the systems we have worked with.

Biological systems are often inherently quite complicated. Accurate representation of such systems in terms of differential equations ordinarily results in “stiffness”; this is widely believed to hold true for natural systems generally. A common problem in simulating them is that simplification for the sake of mathematical tractability often leads to biologically unrealistic results. Representing a system in terms of n differential equations (where often n>100) may lead to the following serious dilemma:

(1) n must be large enough for biological realism;
(2) n^2 (the size of the Jacobian matrix) must be small enough for available core memory.

The methods we have developed reduce core memory requirements to a practical range and also save computer time. While they are based on the structure of the biological systems which we have investigated, the methods can be extended in varying degree to other complex systems of stiff differential equations.

We have used two main approaches:

(1) Replace some of the differential equations with algebraic equivalents when this is possible.
(2) Reduce the number of elements of the Jacobian matrix required to be stored in memory for the Gear method to solve the remaining differential equations.

While these two approaches can be combined for a given system, as exemplified later, they are described separately in the two following sections.

SUBSTITUTION OF ALGEBRAIC RELATIONSHIPS

The general approach employed is to separate out from the total system those components whose behavior can be calculated by algebraic equations rather than differential equations, thus greatly reducing the number and possibly the “stiffness” of the latter.
In metabolic models enzyme mechanisms and reactions at equilibrium can be separated out and dealt with by specialized algebraic means:

1. Individual enzyme behavior in a complex model can be represented by an algebraic rate law which can be determined by standard systems analysis procedures. The differential equations for these enzyme forms are converted to explicit algebraic equations. The concentrations of the enzyme forms and the rate at which the enzyme reactions go are calculated by matrix inversion. Only the small molecules involved need be represented by differential equations. The saving of CPU time and memory is indicated by the last two columns of Table 1, although a little space is required for the algebraic equations.

2. Some chemical reactions (e.g., inorganic chelations) are very rapid compared to others in the model and can be represented as always being at equilibrium. Techniques for calculating these equilibria, which have existed for some time have been adapted for joint use with differential equations solving and incorporated into the program described here. As these reactions (here referred to as “fast reactions”) are the ones with the fastest time constants, removing them from the system of differential equations reduces the “stiffness”.

The differential equations which remain after the above elements have been separated out are then solved by the Gear method. The efficiency of Gear's differential equation solver is independent of stiffness over a wide range of stiffness, although we have been able to slow it down by making the equations very stiff. However, if a diagonalized approximation is used for the Jacobian matrix in Gear's method, the sensitivity to stiffness appears to increase considerably. Separating out the fast reactions alleviates this situation by reducing the stiffness. The diagonalized approximation may also be aided by the fact that the elements removed from the Jacobian matrix in this process are commonly off-diagonal elements as well as being the (absolutely) largest ones.

MODIFICATION OF THE GEAR METHOD

In the course of considerable experimentation with various versions of the Gear method we found it to be more than satisfactory. We made efforts to improve its performance by reducing its execution time as well as the space requirements. These changes can be considered “tuning” of the Gear method, which speeded things by 10-15 percent. Analysis of the program's behavior revealed that a large percentage of the differential equation solving time is spent in solving a linear system of equations of the form:

\[ X \cdot J = B \]  

where \( J \) is the Jacobian or partial derivative matrix mentioned above. Furthermore the size of the \( J \) is the factor determining the time spent in finding the solution for \( X \), and reducing the size of \( J \) will reduce execution time as well as space requirements.

It is a fortunate situation that the Jacobian matrix for biological models representing metabolic systems is normally quite sparse, so that sparse-matrix techniques are applicable. These were applied by Curtis and Chance in the CHECK and CHECKMAT program. It is possible to go further because the sparseness is structured so that most of the non-zero elements of the matrix are near the main diagonal, as shown in Figure 1. Furthermore, the non-zero elements which are not near the main diagonal may also have a structure which can be exploited; this is in fact done by the removal of fast reactions described in the preceding section. It would be desirable to eliminate the zero elements of the sparse Jacobian matrix from storage in the most efficient way. However, this could not be achieved initially due to the method used by Gear to solve the linear system—backward decomposition with partial pivoting. The pivoting is dependent upon the value of the elements of \( J \) at each given point. This makes it impossible to predict how the sparseness of \( J \) is changed by the decomposition algorithm. On the other hand, we discovered that the partial pivoting made little difference in the behavior of the Gear's method and could be replaced by a simpler backward decomposition procedure. Once pivoting is eliminated, predicting the effect of the decomposition upon the sparseness of \( J \) becomes possible.

Figure 1—Typical sparseness pattern of the Jacobian matrix representing a metabolic system (non-zero elements are indicated in black)
PROCEDURE FOR SPARSE MATRICES

The sparseness prediction algorithm is:

1. \( i = 1 \) (the sparseness of the first row will never be altered by the decomposition process).
2. \( i = i + 1 \) and \( j = 1 \) (consider the next row).
3. if \( i > N \) go to step 9 (if all \( N \) rows of the Jacobian \( J \) have been analyzed, exit the loop).
4. increment \( j \) until the next non-zero element is encountered (all zero elements skipped so far on the row \( i \) will always remain zero).
5. if \( i = j \) go to step 2, (the sparseness of row \( i \) was determined).
6. if \( j > i \) stop—the matrix \( J \) is not a Jacobian.
7. all elements \( J(i, t) \) for which \( t > j \) and \( J(j, t) \neq 0 \) will be marked as non-zero, unless they are already marked as such.
8. go to step 4.
9. the algorithm stops:

   all elements of \( J \) which have not been marked as non-zero will in no circumstances become non-zero as a result of the decomposition process.

The decomposition algorithm will be described below.

Example: application of the sparseness prediction algorithm to a sample Jacobian. In the following, '*' will indicate original non-zero elements, while a '+' will signify that the given element was marked as non-zero during the application of the algorithm. All elements left unmarked by the sparseness prediction algorithm need not be present in core during the integration process.

<table>
<thead>
<tr>
<th>row</th>
<th>row 1</th>
<th>row 2</th>
<th>row 3</th>
<th>row 4</th>
<th>row 5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0<em>0 0</em></td>
<td>0<em>0 0</em></td>
<td>0<em>0 0</em></td>
<td>0<em>0 0</em></td>
<td>0<em>0 0</em></td>
</tr>
<tr>
<td>0*0</td>
<td>0<em>0 0</em></td>
<td>0<em>0 0</em></td>
<td>0<em>0 0</em></td>
<td>0<em>0 0</em></td>
<td>0<em>0 0</em></td>
</tr>
<tr>
<td>0*0</td>
<td>0<em>0 0</em></td>
<td>0<em>0 0</em></td>
<td>0<em>0 0</em></td>
<td>0<em>0 0</em></td>
<td>0<em>0 0</em></td>
</tr>
<tr>
<td>0*0</td>
<td>0<em>0 0</em></td>
<td>0<em>0 0</em></td>
<td>0<em>0 0</em></td>
<td>0<em>0 0</em></td>
<td>0<em>0 0</em></td>
</tr>
<tr>
<td>0*0</td>
<td>0<em>0 0</em></td>
<td>0<em>0 0</em></td>
<td>0<em>0 0</em></td>
<td>0<em>0 0</em></td>
<td>0<em>0 0</em></td>
</tr>
</tbody>
</table>

Subsequently, the sparseness prediction algorithm is used to create two arrays \( WI \) and \( WJ \). For each non-zero element in \( J \) an entry (indicating the column number of the non-zero element) is made in \( WJ \). All entries in \( WJ \) are ordered by rows and the elements in each row ordered in increasing order (by column numbers). Furthermore, for each row a pointer in array \( WI \) is set to indicate its beginning (e.g., the column of the first non-zero element of row \( i \) is given by \( WJ(WI(i)) \)). An additional pointer binds the last row. In the case of the Jacobian used in the previous example, the arrays \( WI \) and \( WJ \) take the following form:

\[ WI: 1 4 6 9 12 15 \]
\[ WJ: 1 3 5 2 4 1 3 5 3 4 5 2 4 5 \]

The preprocessor will pass to the simulator not only the dimensions of \( WI, WJ, \) and \( WPW \) (the Jacobian dimensioned to the same size as \( WJ \)), but also the content of \( WI \) and \( WJ \).

The saving of memory space resulting from this process becomes more significant in the case of our implementation, described below, which is composed of two programs: a preprocessor (called the generator), and the simulator program which actually solves the differential equations. The generator receives as input the set of chemical reactions to be represented by a set of differential equations and, on analyzing it, dimensions the simulator's arrays to the exact length required by the specific system. It then derives the set of differential equations themselves, following which the task of determining the initial sparseness of the Jacobian becomes trivial.

Whenever required, the simulator will compute the values of the Jacobian elements and store them in the appropriate locations of \( WPW \). The following decomposition routine will replace \( WPW \) by a triangular matrix (stored in the upper corner of the Jacobian) and a triangular multiplication factor matrix (stored in the lower side of the Jacobian). The decomposition algorithm is quite similar to the sparseness prediction algorithm since the latter mimics the actions of the former (\( N \) is the number of variables involved; \( J \) is the conceptual Jacobian, not to be identified with its program implementation \( WPW \)):

1. \( i = 1 \) (the first row is unchanged).
2. \( i = i + 1 \) and \( p = 0 \) (consider the next row).
3. if \( i > N \) go to step 9 (the triangularization is complete).
4. find next non-zero element on row \( i \): \( J(i, j) \neq 0 \), i.e., repeat \( j = WJ(WI(i+p)) \) and \( p = p+1 \) until \( WPW(WI(i+p)) \neq 0 \) (note that \( j \) steps over the elements which are known to be zero).
5. if \( i = j \) go to step 2.
6. if \( j > i \) stop—programming error.
7. (a) set \( J(i, j) = K \) (i.e., \( WPW(WI(i+p)) = K \)) where \( K = -J(i, j)/J(j, j) \).
   (b) add row \( j \) to row \( i \) starting with the column \( j+1 \).
8. go to step 4.
9. stop—algorithm is completed.

By using the result of this decomposition routine, the solution routine will compute the value of the unknowns of the equation (1) for each given \( B \).

In accordance with our expectation, the predictive sparseness method resulted not only in substantial savings in core but also in a significant improvement in the speed of Gear's method: the time required to solve the differential equation system decreased considerably. The advantages resulting from this method become more important with increasing system size. To our knowledge, the only other successful research of this type resulted in a (DEC-10) machine-dependent...
program (M. Pring, to be published) which is not yet available for consultation.

The predictive sparseness method is the last of a large series of experimental methods which unfortunately could not pass our acceptance criteria—they had to prove themselves faster than the regular Gear method. In our first attempt, we tried to use a variation of Gear’s method where the Jacobian is replaced by a diagonal approximation, and this was excessively slow for large stiff systems and suffered a loss in accuracy when the same relative error parameter was used. We also tried a new version of Gear’s package, GEARR\(^{10}\) which replaced the Jacobian by a diagonal band of chosen width. Since most of our systems involve Jacobians which are concentrated around the diagonal (Figure 1), the chances of success seemed good. Furthermore, our experiments showed improvements in the execution speed when the Jacobian indeed had all the non-zero elements inside such a band. However, the method was rejected on further testing, which revealed that as soon as elements outside the band became slightly significant the behavior of the program is adversely affected.

A second group of experiments followed in an attempt to use a Gauss-Seidel iteration procedure to solve the set of linear equations (1). In the belief that the slowdown encountered in the previous cases was due to a poor approximation of the Jacobian, we attempted to use the solution given by the diagonal band as a starting point for the iteration process. However, the convergence was much too slow, especially for large systems.

### POSSIBLE ALTERNATIVE METHODS

As mentioned before, the predictive sparseness method gives good results and the larger the number of differential equations, the greater the improvements appear to be. However, alternative differential equation solving methods are available to the user of our program: Euler, regular Gear, or Gear with the Jacobian replaced by a diagonal approximation. All of these methods may also be combined with the fast reaction method mentioned above.

The Euler method is rarely used to integrate stiff differential equations, but past experience has shown it to work as well—or as badly—as the more sophisticated non-stiff methods when faced with a stiff system. Since decreasing stiffness improves the behavior of Euler, it may sometimes be used in conjunction with the fast reaction method; if the latter greatly reduces the degree of stiffness, this combined method will give an accurate solution in a reasonable time for systems where Euler alone would fail. Since the Gear method is “slow-starting” the Euler method is at a relative advantage for short-time calculations, as seen in the first two columns of Table 1 (the second column includes editing time, the first does not).

### Table 1

<table>
<thead>
<tr>
<th>Size</th>
<th>Description</th>
<th>Method of Solution</th>
<th>Test Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>N=9</td>
<td>No fast reactions, no enzymes</td>
<td>stiff</td>
<td>N=65</td>
</tr>
<tr>
<td></td>
<td>not very stiff, not very</td>
<td>time=1500</td>
<td>N=66</td>
</tr>
<tr>
<td></td>
<td>reactions, enzymes represented</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>by differential equations</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Euler with fast reactions:

- J=0: 29.3 sec
- J=9: 15.9 sec
- J=0: 67.9 sec
- J=9: 15.9 sec

Gear diagonal approximation with fast reactions:

- J=9: 8.8 sec
- J=81: 24.5 sec
- J=0: 12 min 25.9 sec
- J=0: 22.2 sec
- J=81: 41.4 sec
- J=400: 41.4 sec
- J=400: 41.4 sec

Regular Gear with fast reactions:

- J=34: 6.6 sec
- J=34: 20.8 sec
- J=34: 6 min 43.3 sec
- J=34: 17.0 sec
- J=34: 40.8 sec
- J=34: 40.8 sec

Gear-predictive sparseness with fast reactions:

- J=34: 6.6 sec
- J=34: 20.8 sec
- J=34: 6 min 43.3 sec
- J=34: 17.0 sec
- J=34: 40.8 sec
- J=34: 40.8 sec

The diagonal approximation of the Jacobian suffers from many of the same weaknesses as Euler, but to a lesser degree. However, when properly combined with the fast reaction method, surprisingly enough, it is faster than the regular Gear method, for some medium-sized systems tested. This is very significant when we think that only \(N\) locations are needed for the Jacobian in place of \(N^2\). This method is still under investigation, but we do not hope for equally good results with larger systems. Moreover, the fast reaction method has applicability restrictions (e.g., it requires a clean distinction between fast and non-fast reactions) which make it unsuitable in many cases.

The predictive sparseness method will not work where the user interferes with the derivative evaluation routine (written by the preprocessor) by inserting additional (FORTRAN) coding and artificially creating partial derivatives of significant size, which cannot be predicted since they are not derived from the original differential equations. The unmodified Gear method does work in this situation.

A short comparative study of the various methods available in the system is summarized in Table 1 (CPU...
time for integration and the size of core used for the Jacobian).

DEPENDENCE ON WORDLENGTH

During our investigations\(^2\) we were able to run our program on three different machines, the Control Data 6400, the IBM 370/165, and the Digital Equipment PDP-10 (with which the bulk of the work was done). We were investigating the effect of wordlength of the accuracy and number of calculations required to solve a test problem. As expected, the 32 bit processor (IBM) gave slightly less precision and performed more calculations in reaching a solution than did the PDP-10. Surprisingly the 6400 (60 bits) performed even more calculations than did the 32 bit IBM 370. Further investigation showed that a minimum number of calculations occurred at wordlengths of 36 to 40 bits.

Since the Jacobian is formed by numerical differencing within the Gear program, increased wordlength will lead to some elements being very small but finite where shorter wordlength would result in such elements being set to zero. If too many are set to zero, accuracy can be lost in forming the Jacobian, thus requiring more evaluations for convergence. If too few are set to zero, there should be no problem. The pivoting strategy used in the matrix decomposition routine in the original Gear program employed no scaling, so that these small elements affected the pivot with a similar loss in accuracy causing additional matrix evaluations. An earlier version with scaling did not demonstrate this phenomenon. Since our latest method does not employ pivoting, the problem has been eliminated. This is mentioned here because this result is so unexpected: increased wordlength is generally believed to improve the efficiency of matrix computations.

IMPLEMENTATION IN A SIMULATION LANGUAGE: BIOSSIM

Application of these methods of handling the Gear matrix necessarily requires a preprocessor of some kind to determine what non-zero derivatives are possible in the Jacobian matrix and to construct pointers for them. We have combined the Gear method into a machine-independent two-pass language biochemical simulation language which we have been using for some time.\(^3\) This language which writes and solves differential equations can perform the necessary operations in its first pass. Our research into methods for solving stiff differential equations grew out of the need to make this language more efficient. It has otherwise been updated considerably, and a report of this will be submitted elsewhere. A most important aspect of this updating is that the program implementing the language have been “structured” to permit modification by the user.\(^4\) Given the capability of such modification and the ability to write as well as solve ordinary differential equations, this language can probably be applied to systems of equations far removed from those for which it was designed. It has already been used for problems which are more nearly physiological than chemical in their definition and for ecological problems. As communication about such usages would be considerably facilitated by having an acronym for the language, we have named it BIOSSIM (for BIOlogical Structured SIMulator).

The ability to solve large numbers of stiff differential equations is expected to considerably assist the study of biological systems by mathematical means, because the equations can be sufficiently complex to meet the need for realism. It has been difficult to develop such computer models in the past because of the cost of computer time, and the unfavorable running restrictions (e.g., nighttime access only, because of the large core requirement). There is reason to hope that the biological models that are feasible to compute with the methods here described will be realistic enough to be of value for clinical and industrial applications.

The BIOSSIM program will be available, as its predecessors for some time have been, through the SHARE Program Library Agency (Library No. 360D-032.008), and possibly in other ways as well.

SUMMARY

A method of decreasing both the running time and the core occupancy of the Gear stiff differential equation solver by “compacting” its large Jacobian matrix with a preprocessor and auxiliary programs is described. This may be further assisted by separating out appropriate subelements of a large set of stiff differential equations and treating them in a separate computer model. In our hands these methods have had a very large impact on the difficulty and cost of solving complex systems of stiff differential equations.

This project was supported by NIH grants GM 16501, HL 15622, and RR 15.

REFERENCES

978 National Computer Conference, 1976


