17th European Symposium on Computer Aided Process Engineesing – ESCAPE17
V. Plesu and P.S. Agachi (Editors)

The solution of DAE systems by a numerically robust and efficient solver

Davide Manca, Guido Buzzi-Ferraris

CMIC Department, Politecnico di Milano, P.zza Leonardo da Vinci, 32, 20133 MILANO – ITALY, davide.manca@polimi.it

Abstract

The paper describes a modern approach to the solution of Differential and Algebraic Equation (DAE) Systems through a C++ routine: BzzDae belonging to the BzzMath freeware numerical library. After an introduction to BzzMath and the object oriented approach to numerical problems the manuscript focuses the attention on the robustness and efficiency features that characterize BzzDae with respect to the Fortran counterparts. A number of clarifications are given to describe the capabilities of BzzDae over the other solvers.

Keywords

Numerical calculus, DAE solver, robustness, efficiency, algorithm, routine

1. Introduction

The solution of applied numerical problems dealing with the integration of Differential and Algebraic Equation (DAE) Systems calls for the adoption of a suitable solver. The complexity of the problem resides mainly in the intrinsic stiffness of the DAE system as reported by Shampine [1]. This means that specific attention has to be paid to both numerical methods and solver routines. In fact, the DAE problem is quite challenging, not only for the precision required, but also in terms of robustness and efficiency. For example, when combustion processes are involved, the robustness feature can be referred to sudden ignitions, steep profiles, high gradients, cool flames, and highly

discontinuous temperature, pressure, and concentration profiles. On the other hand, the efficiency feature is mainly concerned with numerical simulations that can take a large amount of CPU time. The relevant number of equations, the structure of the Jacobian matrix (banded, block, sparse) as well as its evaluation and update are of paramount importance when trying to address the efficiency feature. This manuscript presents the BzzDae solver, which is an object oriented class written in C++, belonging to the BzzMath library [2, 3].
BzzMath is available on the Internet at www.chem.polimi.it/homes/gbuzzi/ and is downloadable as freeware software for non commercial use.

2. Numerical methods and routines for DAE systems

Before addressing the DAE subject, it is worth spending some words on the numerical algorithms for the solution of ordinary differential equation (ODE) systems. As the ancestors of DAE packages, ODE solvers share several peculiarities and a common theoretical background with them. Starting from DIFSUB [4] and passing to GEAR [5], LSODE [6], VODE [7, 8], and BzzOde [9] the improvement in terms of the features and capabilities of ODE solvers has been continuous, mainly in terms of robustness and efficiency. As far as the DAE systems are concerned, besides the well known LSODI routine [6], the most recent decades were characterized by the evolution of the DASSL routine [10] into DASPK [11] and by the introduction of BzzDae [12]. Other integration routines such as: LIMEX [13], DAESOL [14] and SPRINT [15] are also mentioned in the literature but are not so widely diffused as the previous ones. The aim of this paper is to present and discuss a few modifications and numerical expedients introduced into BzzDae with respect to the well known Fortran routines to increase the computational efficiency of the methods while overcoming their numerical instability.
Usually speaking, when chemical engineering problems are involved, the dynamic simulation of a process/system/equipment calls for the solution of a differential-algebraic equation (DAE) system. The differential equations, which describe the time or space evolution of the investigated system are often first order with initial value conditions. The algebraic portion usually comes from equations of state, chemical and physical equilibria, stoichiometric consistency and boundary conditions. As before said, a quite important feature of DAE systems is their intrinsic stiffness [1]. As a consequence, a dedicated numerical routine must be adopted when solving these stiff problems. Often the simulation of such processes is quite challenging in terms of: precision (unbalance in the dynamics of reactants/products and radicals) efficiency (a simulation can take several hours and even days of CPU time) and robustness (short-lived species, sudden ignition, steep profiles, and high gradients).
BzzDae integrates DAE systems of index 1 and 2 in the form: $\begin{cases} \mathbf{y}' = \mathbf{f}_1(\mathbf{y}, t) \\ \mathbf{f}_2(\mathbf{y}, t) = 0 \end{cases}$.

A rather concise description of the main features of BzzDae is here reported.

On the robustness side BzzDae is characterized by the following features:

1. If the real modeled problem is based on variables that are physically bounded, the only Fortran routine able to deal with constraints is DASPK. DASPK apparently allows assigning a non-negative constraint to the solution vector $\mathbf{y}$ throughout the integration path. By setting an internal index, when calling DASPK, the user can specify that $\mathbf{y}$ should not be negative. This option is the first step towards the extended feature of BzzDae that allows specifying both minimum and maximum constraints. With reference to DASPK, if the model subroutine (FEX) comprises functions, which cannot accept negative arguments, then the integration program will abort during the execution, with a floating point exception, as soon as $\mathbf{y}$ becomes negative. This could seem to be nonsensical since DASPK has been informed that negative $\mathbf{y}$ elements should be avoided. The problem is that DASPK checks for non-negative values, setting the negative variables equal to zero, only after the corrector procedure has reached the convergence within the required precision limit. This means that if, during the predictor step or the corrector iterations, any value of $\mathbf{y}$ becomes negative, the solver will not take any action to overcome the negative value problem. This is the reason for the possible math error here described. The DASPK solver also relies on another method to address this issue: the user can assess the consistency of input vector $\mathbf{y}$ through a status index and immediately return the control to the solver. In this case, DASPK reduces the step size and checks again the convergence within the corrector, but if the integration routine, at the last successful n^{th} step , has found $y_i^n = 0$ and $y_i'^{(n)} < 0$ then any predictor action on the following $(n+1)^{th}$ step will be negative: $y_{i,predictor}^{n+1} < 0$ inducing DASPK to reduce the step size indefinitely, until it finally aborts the integration procedure due to a step size $h \to 0$. The correct approach to the constrained problem should come from the DAE solver in terms of decisions to be made at each step of the integration procedure. This is the only way by which the DAE system routine is safe and math errors are avoided *a priori*. Actually, the BzzDae user (if required by the problem's nature) simply assigns the maximum and/or minimum constraint vectors. The solver automatically handles the constraints, taking care not to violate the assigned bounds. The control is performed before passing any illegal values to the DAE system routine. The correction vector, $\mathbf{b}$, is accepted only when the nonlinear system, resulting from the DAE problem, is accurately solved within the assigned precision and, at the same time $\mathbf{y}$ complies with the constraints. By doing so, the DAE function is always computed with safe $\mathbf{y}$ values.
2. BzzDae adopts a stabilization technique when repeated convergence failures occur. The integration order is automatically reduced to one and the DAE solver restarts from the last successful convergence point. By doing so, the

numerical problem becomes completely consistent as shown in detail in [9] and [12].
3. Both the order and step size are reduced when there are convergence problems.
4. As suggested by Brenan et al. [16] the order is reduced when the elements of the Nordsieck vector are not decreasing.

On the efficiency side BzzDae is characterized by the following features:
1. As it happens within DASPK, and contrary to LSODI, there is a distinct memory allocation for both the Jacobian matrix, $\mathbf{J}$, and its factorization $\mathbf{G}=(\mathbf{I}-hr_0\mathbf{J})$. A direct consequence of such a feature is an overall efficiency improvement since when a different step size or method order is chosen, there is no need to reevaluate the Jacobian matrix, $\mathbf{J}$, and then superimpose its factorization, $\mathbf{G}$, which is needed by the non-linear system solver based on the Newton method. Most of the DAE chemical problems have the following characteristics: (1) each function evaluation (DAE system) is highly time consuming, (2) the Jacobian matrix, $\mathbf{J}$, is evaluated numerically by finite differences, (3) the number of equations is quite relevant. From this point of view, it is evident that the function evaluations have the greatest impact on the CPU time [17]. The Jacobian evaluation becomes more and more exacting when the equations number increases.
2. BzzDae follows a different criterion with respect to DASPK in determining when to update the Jacobian matrix. The first significant difference is that BzzDae checks whether $\mathbf{J}$ is out of date through the following equation: $\mathbf{y}'_{n+1} \cong \mathbf{y}'_n + \mathbf{J}\cdot(\mathbf{y}_{n+1}-\mathbf{y}_n)+\mathrm{f}_t\cdot(t_{n+1}-t_n)$, where $\mathbf{y}'_{n+1},\mathbf{y}'_n,\mathbf{y}_{n+1},\mathbf{y}_n$ are the variables at the n^{th} and $(n+1)^{th}$ iterations and f_t is the time derivative of the DAE system. When the equations number, n_{eq}, is high, the Jacobian numerical evaluation is rather time consuming so it is advisable to delay the update of $\mathbf{J}$ as far as possible. Conversely, if the system has few equations, it is convenient to evaluate $\mathbf{J}$ more frequently in order to increase the Newton method efficiency with a reduced effort. Due to these considerations, BzzDae evaluates a new Jacobian matrix at most after m steps, with m being a function of n_{eq}. Since the Jacobian consistency is controlled in a deeper and more accurate way, respect to the conventional DAE routines, it is also safer to increase the maximum allowed number of steps performed, by keeping the Jacobian matrix constant.
3. DAE systems characterized by sparse and not necessarily structured Jacobian matrices can be easily solved by exploiting the automatic memory allocation and matrix rearrangement of the C++ classes.

The aforementioned features improve the overall performance of BzzDae when dealing with problems characterized by:

- DAE systems with Jacobian matrices having complex λ eigenvalues with negative real parts $\mathrm{Re}(\lambda) < 0$ and large imaginary parts $|\mathrm{Im}(\lambda)| >> 1$. Physically speaking this means: highly oscillating problems;
- DAE systems with large discontinuities in the derivatives (*i.e.* discontinuous or piecewise physical properties, `IF ... THEN` structures, code branching);
- DAE systems that require constrained integration variables, *i.e.* the dependent variables, $\mathbf{y}$, must belong to a feasible interval defined by lower and/or upper bound vectors.

In contrast with the DASSL/DASPK approach, BzzDae normalizes the algebraic portion of the Jacobian matrix by the step size: h. With reference to the Jacobian matrix, $\mathbf{G}$, of the non-linear system, the BzzDae formulation exploits the following structure: $\mathbf{G} = \begin{bmatrix} \mathbf{I} - hr_0 \dfrac{\partial \mathrm{f}_1}{\partial \mathbf{y}_1} & hr_0 \dfrac{\partial \mathrm{f}_1}{\partial \mathbf{y}_2} \\ \dfrac{\partial \mathrm{f}_2}{\partial \mathbf{y}_1} & \dfrac{\partial \mathrm{f}_2}{\partial \mathbf{y}_2} \end{bmatrix}$

as opposite to the DASSL/DASPK structure: $\mathbf{G} = \begin{bmatrix} \mathbf{I} - hr_0 \dfrac{\partial \mathrm{f}_1}{\partial \mathbf{y}_1} & hr_0 \dfrac{\partial \mathrm{f}_1}{\partial \mathbf{y}_2} \\ h\dfrac{\partial \mathrm{f}_2}{\partial \mathbf{y}_1} & h\dfrac{\partial \mathrm{f}_2}{\partial \mathbf{y}_2} \end{bmatrix}$

where: $\mathbf{y}_1$ are the differential variables, $\mathbf{y}_2$ are the algebraic variables, h is the stepsize and r_0 is the first coefficient of the Backward Differential Formula method. Brenan et al. [16] suggest to divide the algebraic equations for a constant proportional to h. The DASPK guidebook [11] suggests to multiply the algebraic equations by a constant which is proportional to the inverse of the stepsize h. Since the second row elements of the original $\mathbf{G}$ matrix are multiplied by h, it is correct and advisable to automatically simplify such elements through an *a priori* division by h. This simple trick significantly increases the robustness and precision of the solver.

3. Conclusions

When addressing DAE problems, it is very advantageous to make use of dedicated DAE solvers while avoiding any decoupled implementation of the differential and the algebraic portions. The performance losses are evident when solving the algebraic portion inside the ordinary differential system.

A further relevant aspect refers to the boundaries on the integration variables. As a matter of fact, the direct manipulation made by the solver itself of the lower and upper limits within the predictor/corrector steps is a critical feature in terms of both robustness and efficiency. Robustness and efficiency are not two antithetic terms. Conversely, when the numerical solver encompasses a crisis, during the DAE integration, the increment in robustness is tightly coupled to a

performance enhancement. In this sense a robust numerical routine should be able to debottleneck several stagnant situations.

References

1. Shampine L. F., "What is Stiffness? Stiff Computation", R. C. Aiken ed., Oxford University Press, New York, 1-16, (1985).
2. Buzzi-Ferraris G., Scientific C++: Building Numerical Libraries the Object-Oriented Way, Addison Wesley Longman, New York, (1993).
3. Manca D., G. Buzzi-Ferraris, BzzMath: an Object Oriented Numerical Project, Proceedings of ICheaP 7, Taormina Italy, 15-18 May, (2005).
4. Gear C.W., Algorithm 407, DIFSUB for Solution of Ordinary Differential Equations, Comm. ACM, 14(3) (1971) 185-190.
5. Hindmarsh A.C., GEAR: Ordinary Differential Equation System Solver, Report UCID 30001, Rev. 3, Lawrence Livermore Laboratory, Livermore, CA, (1974).
6. Hindmarsh A.C., LSODE and LSODI, Two New Initial Value Ordinary Differential Equation Solvers, ACM SIGNUM Newsletters, 15 (1980) 10-11.
7. Brown P.N., G.D. Byrne, and A.C. Hindmarsh, VODE: A Variable Coefficient ODE Solver, Siam J. Sci. Stat. Comput., 10 (1989) 1038-1051.
8. Byrne G.D., A.M. Dean, The Numerical Solution of Some Kinetics Models with VODE and CHEMKIN II, Comp. Chem., 17 (1993) 297-302.
9. Buzzi Ferraris G., D. Manca, "BzzOde : A New C++ Class for the Solution of Stiff and Non Stiff Ordinary Differential Equation Systems", Computers Chem. Engng., Vol. 22, 11, 1595-1621, (1998).
10. Petzold L.R., A Description of DASSL: a Differential/Algebraic System Solver, Scientific Computing, R. S. Stepleman et al. (Eds.), North-Holland, Amsterdam, (1983).
11. Brown P. N., A. C. Hindmarsh and L. R. Petzold, "A Description of DASPK: A Solver for Large-Scale Differential-Algebraic Systems", Lawrence Livermore National Report UCRL, (1992).
12. D. Manca, G. Buzzi-Ferraris, T. Faravelli and E. Ranzi, "Numerical Problems in the Solution of Oxidation and Combustion Models", Combustion Theory and Modelling, 5, 185-199, (2001).
13. Deuflhard P., U. Nowak:, "Extrapolation Integrators For Quasilinear Implicit Ode's", University of Heidelberg, SFB 123, Tech. Rep. 332, (1985).
14. Bauer I., F. Finocchi, W.J. Duschl, H.-P. Gail and J.P. Schlöder, "Simulation of chemical reactions and dust destruction in protoplanetary accretion disks", Astron. Astrophys., 317, 273-289, (1997).
15. Berzins M., R. M. Furzeland, "A User's Manual for SPRINT - A Versatile Software Package for Solving Systems of Algebraic Ordinary and Partial Differential Equations: Part 1 - Algebraic and Ordinary Differential Equations", Report TNER.85.058, Shell Research Ltd, Thornton Research Centre, Chester, (1985).
16. Brenan K. E., S. L. Campbell and L. R. Petzold, "Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations", North-Holland, New York, (1989).
17. Manca D., T. Faravelli, G. Pennati, G. Buzzi Ferraris and E. Ranzi, "Numerical Integration of Large Kinetic Systems", ICheaP-2 Conf. Ser., ERIS, 115-121, (1995).