CHEMICAL ENGINEERING TRANSACTIONS
VOL. 29, 2012
A publication of AIDIC
The Italian Association of Chemical Engineering
Online at: www.aidic.it/cet
Guest Editors: Petar Sabev Varbanov, Hon Loong Lam, Jiří Jaromír Klemeš

ISBN 978-88-95608-20-4; **ISSN** 1974-9791
DOI: 10.3303/CET1229041

Performance Assessment of Existing Methodologies for Chemical Process Dynamic Simulation

Guillermo Durand*[a], Flavio Manenti[b], Josè Alberto Bandoni[a], Maria Soledad Diaz[a], Guido Buzzi-Ferraris[b]

[a]Planta Piloto de Ingeniería Química, PLAPIQUI (UNS-CONICET), Camino La Carrindanga, Km. 7, 8000 Bahía Blanca, Argentina
[b]Politecnico di Milano, Dipartimento di Chimica, Materiali e Ingegneria Chimica "Giulio Natta", Piazza Leonardo da Vinci 32, 20133 Milano, Italy
gdurand@plapiqui.edu.ar

The chemical engineer has nowadays a wide choice of tools, numerical libraries, and programming languages to perform computations. Actually, it is possible to use several well established commercial packages, implement dedicated solvers into specific programming languages, or use existing numerical libraries. Also, it is possible to combine these possibilities to get either superior performances or more robustness, according to the problem features through the so-called mixed-language approach, which is increasingly spreading in the scientific communities. Since there is no full clarity on their benefits in handling numerical problems and their performances have not been yet compared in the literature, this paper is aimed at analyzing efficiency and robustness of some of the most widespread methodologies adopted for numerical computations: the conventional methods, the implementation of numerical libraries, the mixed-language, and the commercial tools. Specifically, the common case of differential systems is selected as comparison field.

1. Introduction

For several decades scientists and engineers have developed and implemented algorithms, numerical methods, programs, and computational tools in general to handle a wide range of problems (e.g.: linear algebra, nonlinear systems, ordinary differential and differential-algebraic equations systems, se for example Dongarra et al. (1990) and Boroni et al. (2009)). Since the evolution in numerical analysis is quite slower than the evolution of personal computers and power computing (Buzzi-Ferraris, 2011), it is frequent to see new and more sophisticated programs that use old, but evenly performing, numerical methods and computing procedures. This situation is typical of scientific programmers that use relatively younger languages and/or releases for computations so as to exploit new features (i.e. the object-oriented programming using Visual C++ or Java) and, sometimes, are forced to implement in their program existing and well-known codes and routines previously developed in other programming languages (i.e. the procedural routines of Fortran or Matlab) so as not to completely re-write the overall code and numerical methods. Furthermore, well-established and spread tools and solutions require new methods and more performing algorithms to solve increasingly larger systems. In this specific case, a usually older programming language should accept libraries and code samples developed in newer programming languages. This is the case of commercial packages for process simulation, which have a consolidated structure and graphical user-friendly interface originally developed using certain programming languages, that may need of external customizations for extending their application

Please cite this article as: Durand G., Manenti F., Bandoni J. A., Diaz M. S. and Buzzi-Ferraris G., (2012), Performance assessment of existing methodologies for chemical process dynamic simulation, Chemical Engineering Transactions, 29, 241-246

domain by the use of more detailed models and improve their performances by new algorithms as well (Manenti et al. 2011). What is in common for these two opposite situations is the need to merge different programming languages and implement numerical libraries as well to achieve new objectives. From this point of view, there are no papers in the literature, to our knowledge, that compare advantages and disadvantages of mixed-language and other methodologies used to handle chemical engineering issues. This paper aims at investigating and comparing the numerical performance of different methodologies looking forward to their off-line and on-line use. The numerical topic selected for this study is the integration of differential systems, thus the process dynamic simulation.

2. State-of-the-art and organization of the work

It is currently more complex to perform comprehensive numerical comparisons with respect to the previous decades especially in the chemical engineering field where there is a strong penetration of numerical analysis at all the levels and areas. It is rather complicated since the numerical comparisons now should account for differences between object-oriented and procedural solvers, sequential and parallel computing, shared and distributed memory machines and, again, among software architectures, programming languages and other relevant features, whereas years ago there was more or less a single procedural programming language, the sequential calculation procedure only, and a few numerical libraries. Moreover, today there is the coexistence of many methodologies to solve numerical problems, and in turn the availability of many techniques and methods for each methodology, and the existence of multipurpose tools and solvers with specific performances according to the numerical field investigated as well as the selected conditions and input of the single problem. There is the need to clarify what methodologies can be used to address a numerical problem, whatever is the problem, and to highlight their possible pros and cons. There are many papers in the literature that compare numerical libraries for C++ and Fortran languages, algorithms to handle spoiled Jacobians in process control problems and commercial tools (Manenti et al., 2009). Conversely, there are no papers comparing the strategy we can apply, i.e., the philosophy adopted to set up the program and hence through: a) the use of commercial tools; b) the use of programming languages and their available conventional methods; c) the use of programming languages with the implementation of *ad hoc* solvers and d) the use of mixed-languages.

2.1 Commercial tools

The current commercial software available for dynamic simulation has a complex architecture including multipurpose graphical interfaces, thermodynamic libraries, model libraries for unit operations, differential and differential-algebraic solvers, and support tools. Many simulators are field-proven by industrial applications in many areas from the oil & gas to the fine chemical, from the power generation to the petrochemical, and their main advantage is in the intrinsic multipurpose nature and the user-friendly interface, especially in large physical-chemical databases. The use of the main dynamic simulators allow to perform dynamic simulations of complex systems with relatively small effort since the user usually does not see any equation and needs basic tutorial to use them. Nevertheless, their current nature is becoming more and more a problem for chemical engineering issues for different reasons (i.e.: no detailed models within their libraries, slower in performances than programs developed in scientific languages and stiff structures for modifications). Within this context, looking forward to the assessment of performances and robustness, certain process dynamic simulators seem to be preferable than others. gPROMS is an equation-based simulator that gives the possibility to directly enter own code and to solve it by means of the gPROMS's differential solvers. Since this is not possible for certain packages, this is the main reason for the selection of gPROMS as commercial simulator. Thus, in the following, we will refer to this methodology as the gPROMS.

2.2 Programming languages

All of the most used programming languages (Fortran, Pascal, C, Matlab) to solve numerical problems adopt a procedural programming philosophy, which is based on the possibility to write generalized pieces of code to solve problems of different natures. In the last two decades, object-oriented programming dramatically changed the way to think and develop numerical programs and it is slowly having the upper hand over the traditional procedural philosophy (Buzzi-Ferraris, 2011). For these

reasons, we selected two different programming languages for the comparison: a procedural one, the MATLAB, and an object-oriented one, the C++. MATLAB provides the essential basis for comparison since it is a spread programming and educational language with many well-established algorithms (thus, in the following, we will refer to this methodology as the Matlab); on the other hand, C++ offers the opportunity to assess the pros and cons in using external numerical libraries.

2.3 Numerical libraries

As mentioned above, there are many papers reporting library comparisons in order to assess their efficiency and robustness in solving certain numerical problems. The most frequent comparison is proposed between libraries for Fortran and libraries for C++. On the other hand, C++ libraries can be totally or partially based on the object-oriented programming, or no object-oriented at all. Numerical recipes by Press et al. (1988) and Press et al. (1997) are developed for both Fortran and C++ environments. Contrary to the existing papers, this research activity is not aimed at comparing performances of numerical libraries, rather to shows pros and cons of their uses with respect to the other methodologies. Thus, all the aforementioned libraries are evenly good for the scope. We selected the BzzMath (Buzzi-Ferraris, 2012) library, specifically the family of BzzOde solvers, since it is fully object-oriented and hence it should have superior performances and flexibility for target purposes. Lastly, certain general numerical issues that affect all numerical libraries (Buzzi-Ferraris, 2011) have been already fixed in BzzMath. Thus, in the following, we will refer to this methodology as the BzzMath.

2.4 Mixed-language

The mixed-language methodology is characterized by the use of more than a single tool. For example, the use of Fortran subroutines for calculations in C++ is a typical mixed-language approach (Buzzi-Ferraris and Manenti, 2010). The commercial software adopted in this work, gPROMS, has a rather stiff programming structure, and some work-around had to be implemented in order to apply gPROMS simulations to our test cases. The tests consisted in the dynamic simulations of two processes under different conditions, using random-generated control variables' profiles. For gPROMS to automatically read the corresponding input data and run the simulation it uses the options given by the package of running a gPROMS simulation or optimization from the MS-DOS command line, and of reading input data from a MS Excel sheet. A MATLAB code updates the data to be read, then calls the gPROMS executable to run the simulation. When gPROMS run ends, it computes the running time. Less complicated is the mixed-language adopted to couple Matlab and external libraries (BzzMath). The technique is called the Mex-function and is a particular feature of Matlab that allows it to recognize C++ code as its own, this way giving the possibility to create synergies between the power of C++ and the agility of matrix-oriented Matlab. In the following, we will refer to this methodology as the MEX function.

3. Dynamic simulations

3.1 Numerical methods and test models

The numerical methods vary according to the selected methodology. MATLAB uses the Adams-Bashforth and Adams-Moulton methods (Matlab, 2008). BzzMath library use multi-value algorithms (basing on their corresponding multi-step algorithms), it also implements a branched structure to handle the Jacobian matrix sparsity and structure (Manenti, 2011). gPROMS uses the DASOLV, which is based on variable time step/variable order Backward Differentiation Formulae (BDF). This solver uses the MA48 solver for linear algebra; it employs direct LU factorization algorithms designed for large, sparse, asymmetric systems of linear equations (gPROMS, 2004). Most of the ordinary differential equation (ODE) systems resulting in chemical engineering problems have the following characteristics: each function evaluation is highly time-consuming; the Jacobian matrix J is evaluated numerically (finite difference), and the number of equations is quite high. From this point of view, it is clear that function evaluations have the greatest impact on the overall computational effort. As the number of equations is increased, the Jacobian evaluation becomes relevant. Especially for this reason, we selected very simple examples to test and compare the different numerical methodologies. Two test models were selected: a binary conventional distillation column, which leads to an ODE system with 46 equations; and a batch brewery mashing process, which leads to a differential algebraic equations (DAE) system with 18 differential equations and 13 algebraic equations. Since the DAE

system is a stiff one, no distinctions between ODE stiff and nonstiff problems are provided. A binary conventional distillation column was selected as test case for the ODE systems (Luyben, 1990). The dynamic model undergoes the assumption that relative volatility α is constant across the column and theoretical trays. A single inlet feed F with composition z (molar fraction of the light component) enters the column at tray NF at bubble point conditions. The vapour flow exiting the top of the column is condensed by a total condenser and it is sent to a reflux drum with holdup M_D. The flowrate that exits the reflux drum at bubble point conditions has uniform composition x_D. The reflux R is sent through a pump to the top tray, whereas the distillate D exits the unit. The test case for DAE systems is a batch process involving the mashing step in beer manufacturing, which is described in detail in Durand et al. (2009). During the mashing process, solid starch grains undergo a transition into a gelatinized state, which is hydrolyzed by the action of dissolved α-amylase. The hydrolysis products are maltotriose and dextrins. Dextrins are converted into sugars (glucose, maltose) and limit dextrins by the action of dissolved β-amylase. The enzymes, α-amylase and β-amylase, undergo temperature deactivation during the batch cycle. Saccharose and fructose are not included in the model since their concentration in the wort is insignificant. During the mashing β-glucans are extracted from the grist to the liquid phase. Dissolved β-glucans are converted into shorter β-oligosaccharides by β-glucanases, which also suffer temperature deactivation. The arabinoxylans present in the grist dissolve and undergo degradation into oligo-β-xylosides by the action of the endo-xylanase enzyme.

3.2 Simulations

Industrial processes are constantly subject to several kinds of planned or unplanned operating transients. Since our both test cases are simple, it is very easy and fast for any of the methodologies tested here to solve them in their nominal values, thus preventing us to compare computing performance. Because of this fact, the models were tested under transients of variables with noticeable impact on their dynamics. Also, looking forward the use of dynamic models for online predictions and optimizations, many simulations are performed to assess the performances. The dynamic simulations of the distillation column were performed in closed loop and open loop modes using the data provided in Luyben (1990). For the closed loop mode a transient model profile was used for the feed flow. It is comprised of a step increase over the nominal value, starting at time $tstart_1$, keeping that value for $tlength_1$ time units, then a further step change to a decreasing of the same magnitude under the nominal value, continuing for $tlength_2$ time units, and finally returning to the nominal value within an entire time horizon of 600 time units. For a more rigorous comparison, 100 hundred transient profiles were generated, with $tstart_1$ following a distribution of $N(0,1)$ multiplied by 50, within the [0,100] range, while $tlength_1$ and $tlength_2$ followed a $N(100,10)$ distribution within the [50,150] range. Each one of the profiles was tested with amplitude variations of 5/10/15/20/25 %, numbering a total of 500 simulations per each methodology. The time horizon was discretized in time periods of one time unit, in order to apply the inputs from the proportional-integral control. Open loop mode simulations were performed following perturbation profiles comprising only the step change at $tstart_1$, as further perturbations were enough to produce stopping numerical errors for all techniques. Again, 100 profiles were used for each of perturbation amplitudes 5/10/15/20/25%, resulting in 500 simulations per technique. Since the column mode was simulated in open loop mode, simulations were done in two runs, from time 0 to $tstart_1$, then from $tstart_1$ to time 300. Because this division resulted in a discretization in two time periods, open loop simulations required less computational effort. In breweries mashing batches are controlled by temperature profiles, containing 3, 4 or 5 steps where the temperature is kept constant in each one. Since all processes (enzymatic reactions, enzymes and metabolites solubilization and enzymes denaturization) are temperature dependent, the desired final concentrations are reached varying the temperature and length of each step. As in the previous test case, the simulations for the mashing model were carried out using a battery of random-generated temperature profiles. Each profile of 115 minutes used in the simulations involved 5 steps, with each successive step having higher temperature than the previous one. The starting point of each step (variables ts) were generated following a uniform distribution of $U(20,115)$ minutes with checking to assure that a steps started at least 5 minutes after the previous one. The temperature of each step (variables TR) were generated using a $U(42,80)$ Celsius degrees distribution. The temperature profiles were included in the model using sigmoid functions, thus allowing performing simulations without discretizing the time horizon.

4. Numerical discussion

Tables 1-2 show the running simulation times for the test case of the distillation column. Simulations were done on an Intel P8600 at 2.40GHz system with 3.46 GB of RAM. Multicore processing options were disabled for all methodologies to prevent any numerical gaps due to the fact that different parallel strategies could affect the comparison among the methodologies. The BzzMath library tests and the C++ part of the "MEX function" tests were compiled using the MS Visual C++ 6.0 compiler. Each row of the tables shows the aggregated running time for 100 simulations. For the closed loop mode (shown in Table 1) in all methodologies, completing the 100 perturbation profiles for larger perturbation amplitudes was more time consuming, except when using gPROMS, where the tendency was not so obvious. This is a forced situation because larger perturbations cause more complicated dynamics and, thus, more correction actions for the PI control. Conversely, the complex numerical/graphical gPROMS architecture, typical of all commercial packages, hides this aspect. Table 2 shows the computational time required for simulations of the distillation column in open loop mode. As expected, aggregated running times for simulations of the open loop mode of the distillation column took less time than the closed loop mode. The curious point is that an open loop simulation with a perturbation of 5% on the feed flow shows the MEX function (that is the MATLAB using the BzzMath solvers) methodology is more efficient than the BzzMath one (used in C++ environment). This aspect is still to be investigated. In the closed loop simulations, while the amplitude of the perturbation increases, the computational times of BzzMath and MEX function increase too, but their relative gap in computations decreases. Their gap is null when the system becomes unstable and the numerical methods cannot solve it. In all cases the numerical library (BzzMath) was at least an order of magnitude faster than the general programming language (Matlab) and the commercial package (gPROMS), moreover, in the closed loop simulations BzzMath was two orders of magnitude faster than gPROMS and almost two than Matlab. An interesting result is that the mixed-language methodology "MEX function" had a very similar performance to the numerical library's one. the mixed-language methodology required from 41.5 % higher computational time in the closed loop for perturbations on feed flow tests, to only 1.5 % higher in the closed loop for perturbations on feed composition. Table 3 shows the results of the simulation tests carried out for the brewery mashing model, for all four methodologies analyzed, done in the same system and same conditions as the column's test case. Again, the numerical library outperformed the programming language and the commercial package, being one and three orders of magnitude faster, respectively. The mixed-language methodology "MEX function" required 76 % higher computational time, but it still was much closer to the BzzMath performance than the Matlab's one.

Table 1: Aggregated running times for simulations of the binary distillation column, closed loop mode

Perturbation	*Methodology [s]*			
amplitude	*BzzMath*	*MEX function*	*Matlab*	*gPROMS*
5 %	27.093	48.448	753.030	4204.053
10 %	32.219	50.074	759.322	4283.847
15 %	37.672	51.456	770.288	4269.993
20 %	40.734	52.737	785.574	4223.565
25 %	43.907	54.149	798.189	4217.344
Totals	**181.625**	**256.864**	**3866.403**	**21198.802**

Table 2: Aggregated running times for simulations of the binary distillation column, open loop mode

Perturbation	*Methodology [s]*			
amplitude	*BzzMath*	*MEX function*	*Matlab*	*gPROMS*
5 %	18.141	17.487	316.957	540.917
10 %	16.500	16.711	289.093	556.429
15 %	15.422	16.417	270.870	575.063
20 %	14.672	16.290	257.503	597.011
25 %	14.078	16.298	247.117	623.219
Totals	**78.813**	**83.202**	**1381.540**	**2892.639**

Table 3: Aggregated running times for 500 simulations of the brewery mashing test model

	Methodology [s]			
	BzzMath	*MEX function*	*Matlab*	*gPROMS*
Total	2.517	4.435	65.141	1837.135
Average	0.005	0.009	0.130	3.754

5. Conclusions

This work proposed a comparison among numerical methodologies that can be adopted to solve typical chemical engineering problems. Specifically, for relevance and spreading reasons, the dynamic simulation of chemical processes, hence the integration of differential systems, was selected as comparison field. The selected methodologies are the use of commercial packages, the use of conventional numerical methods, the use of dedicated numerical libraries, and the mixed-language approach. An important aspect highlighted by the paper is that there is still a large performance gap between scientific solutions and commercial packages. No one of the commercial packages quoted in the paper can ensure an effective solution for online issues, since the computational times required to solve the simple test cases is too much and their structure is too stiff to accept different numerical kernels to speed-up calculations. In other words, it is not yet possible to use dynamic simulations developed with commercial packages for model predictive control. For example, the 500 simulations of the mashing system, which is a reasonable number of simulations required by a loop of nonlinear model predictive control, are solved in half an hour, a rather high computational time for having a prompt response. On the other hand, the use of dedicated numerical libraries can shrink the computational time (in the case of mashing to 2.5 s). The same concept can be extended to Matlab performance. In addition, the paper emphasizes an interesting possibility: the use of mixed-language approach allows exploiting the potentiality of numerical libraries in other programming environments and software while preserving their performances.

References

Boroni G., Lotito P., Clause A., 2009, A new numerical method for stiff differential equations, Latin American Applied Research, 39, 53-56.

Buzzi-Ferraris G., 2011, New trends in building numerical programs, Computers & Chemical Enginnering, 35, 1215-1225.

Buzzi-Ferraris G., 2012, Politecnico di Milano, <chem.polimi.it/homes/gbuzzi> accesseed 02/08/2012

Buzzi-Ferraris G., Manenti F., 2010, Interpolation and Regression Models for the Chemical Engineer: Solving Numerical Problems Wiley-VCH, Weinheim, Germany.

Dongarra J.J., Duff I., DuCroz J., Hammarling S., 1990, A Set of Level 3 Basic Linear Algebra Subprograms, ACM Transactions on Mathematical Software, 16, 1-28.

Durand G.A., Corazza M.L., Blanco A.M, Corazza F.C., 2009, Dynamic optimization of the mashing process. Food Control, 20, 1127-1140.

gPROMS, 2004, Advanced User Guide. Process Systems Enterprise Ltd.

Luyben W.L., 1990, Process Modeling, simulation and control for chemical engineers, NY, McGraw-Hill.

Manenti F., 2011, Considerations on Nonlinear Model Predictive Control Techniques, Computers & Chemical Engineering, 35, 2491-2509.

Manenti F., Buzzi-Ferraris G., Pierucci S., Rovaglio M., Gulati H., 2011, Process Dynamic Optimization Using ROMeo, Computer Aided Chemical Engineering, 29, 452-456.

Manenti F., Dones I., Buzzi-Ferraris G., Preisig H.A., 2009, Efficient Numerical Solver for Partially Structured Differential and Algebraic Equation Systems. Ind. Eng. Chem. Res., 48(22), 9979-9984.

Matlab (R2008b), Simulink 6.0, Stateflow 6.0, 2008, User's Manual. Natick, MA, The MathWorks, Inc.

Press W.H., Flannery B.P., Teukolsky S.A., Vetterling W.T., 1988, Numerical recipes in C, Cambridge, Cambridge University Press.

Press W.H., Teukolsky S.A., Vetterling W.T., Flannery B.P., 1997, Numerical recipes in Fortran 77 and Fortran 90: the art of scientific and parallel computing, Cambridge University Press, NY.