Module-oriented automatic differentiation in nonlinear control

Jin Li, Yuejin Tan, and Liangcai Liao

Abstract—**In this paper, a module-oriented automatic differentiation (MAD) approach is presented based on traditional automatic differentiation algorithms. This approach can well exploit the sparsity of the model by partitioning it into a series of sequential modules and choosing the best differentiation algorithm for each module accordingly. Numerical results show that for nonlinear system, module-oriented automatic differentiation can calculate the Lie derivatives and Jacobians efficiently.**

I. INTRODUCTION

CHEMICAL processes present special control problems and offer special opportunities due to their inherent nonlinearity. The use of a linear controller often limits the range of operation to restricted feed-stocks (e.g. fluidized catalytic cracking) and limits the speed of recovery from process disturbances. However, nonlinear controllers are difficult to tune and require excessive computational time and effort to perform the optimization.

The control system technique is based on state and input transformations as well as the input/output relationship for the system. The idea is to globally linearize and stabilize the nonlinear system where the questions of system inversion and disturbance decoupling are precisely defined. There exist two important procedures to globally linearize a nonlinear system, state space linearization and input/output linearization.

Using techniques of nonlinear control, the synthesis of feedback control laws has been successful for relatively simple systems. However, the computations required for complex highly nonlinear systems have been out of reach of previous technology like finite difference approximation (FD) and symbolic differentiation (SD). FD may be the most widely used Jacobian evaluation approach and SD is often used in computer algebra packages [1]. Nevertheless, Automatic differentiation (AD) [2], which is developed rapidly during the recent 20 years, has been well recognized as the most promising one among the differentiation algorithms. Quetzalcoatl [3] adapted Automatic Differentiation in Fortran (ADIFOR) to compute the Lie derivatives and Jacobians necessary for the synthesis of the control laws. The numerical results show that automatic differentiation is efficient. In fact, the above three differentiation algorithms have their own advantages and disadvantages. We can select the best derivative evaluation approach based on the problem's structure. The advantages and disadvantages of the three differentiation algorithms are compared in Section 2. Then, in Section 3, a new approach for Lie derivatives and Jacobian evaluation is presented [4]. In Section 4, the advantages of MAD are demonstrated by numerical experiments. Some concluding remarks are presented in Section 5.

Manuscript received July 15, 2005. This work was supported by the Changling refinery of Sinopec.

Jin Li is with the Department of Management, National University of Defense Technology, Changsha, Hunan 410073 China (phone: 86-731-4575857; fax: 86-731-4573591; e-mail: lj_nudt@ hotmail.com).

Yuejin Tan, was with National University of Defense Technology, Changsha, Hunan, China. He is the president of Information System & Management College, National University of Defense Technology, Changsha, Hunan 410073 China (e-mail: yjtan@nudt.edu.cn).

Liangcai Liao is with the Department of Management, National University of Defense Technology, Changsha, Hunan 410073 China (e-mail: llc_nudt@163.com).

II. COMPARISON OF THE THREE DIFFERENTIATION APPROACHES

Although researchers have done many works on the application of AD, AD cannot completely replace FD and SD. Tolsma and Barton pointed out SD outperforms AD when applied to linear equations [1]. Bischof et al. validated that applying AD in an indirect way to partially separable function is highly efficient [5]. Suppose there is a function $y = f(x)$, $f: R^3 \to R$, where

$$y_1 = f_1(x) = 2(3x_1 + x_2) \tag{1}$$

$$y_2 = f_2(x) = \frac{x_3}{x_3 + 1} \tag{2}$$

$$y_3 = f_3(x) = \frac{x_2}{x_1 - (x_2 / x_3)} \tag{3}$$

$$y = f(x) = f_4(x, y_1, y_2, y_3) = x_2 x_3 y_1 y_2 y_3 \tag{4}$$

f_1, f_2, and f_3 are the modules called by f. f_4 is a typical module for which AD outperforms FD or SD [6]. But as for the other modules, AD is not necessarily the best choice. Since our discussion is based on scalar architectures, the number of scalar computations (represented as C) during differentiation can be used to reflect the real differentiation time, so that the efficiency of the differentiation algorithms can be analyzed quantitatively. Take as an example.

Forward AD:

$$\frac{dy_1}{dx} = 2 \cdot (3 \cdot [1 \quad 0 \quad 0] + [0 \quad 1 \quad 0]), \quad C = 9$$

If the sparsity of (1) is exploited by using a Jacobian-compressing technique, then

$$\frac{dy_1}{dx} = [2 \cdot (3 \cdot [1 \quad 0] + [0 \quad 1]) 0], \quad C = 6$$

Reverse AD:

First, evaluating f_1 in forward mode,

$x_4 = 3x_1$, $x_5 = x_4 + x_2$, $y_1 = 2x_5$

Second, accumulating the elementary partial derivatives in reverse mode,

$\frac{\partial y_1}{\partial x_5} = 2 \cdot 1$, $\frac{\partial y_1}{\partial x_4} = \frac{\partial y_1}{\partial x_5}$, $\frac{\partial y_1}{\partial x_2} = \frac{\partial y_1}{\partial x_5}$, $\frac{\partial y_1}{\partial x_1} = \frac{\partial y_1}{\partial x_4} \cdot 3$, $\frac{\partial y_1}{\partial x_3} = 0$, $C = 2$

SD:

$\frac{\partial y_1}{\partial x_1} = 6$, $\frac{\partial y_1}{\partial x_2} = 2$, $\frac{\partial y_1}{\partial x_3} = 0$, $C = 0$

FD:

$\frac{\partial y_1}{\partial x_i} = \frac{f_1(x_i + \Delta x_i) - f_1(x_i)}{\Delta x_i}$, $i \in \{1,2,3\}$, $C = 18$

Obviously, SD is the best choice for f_1, as no computation is required to evaluate the symbolic Jacobian. It is because the expressions have been simplified during the generation of the symbolic derivatives. Similar computations include multiplications between 1 and any quantity, 0 and any quantity, additions between 0 and any quantity, etc. These computations, called redundant computations, are carried out during each of the iterations in optimization with AD, while they are performed only once before optimization with SD. That is why people tend to use SD for the differentiation of linear equations [1]. Also it can be seen that Jacobian-compressing technique exploits the sparsity of the Jacobian of f_1 and improves the efficiency of AD (forward mode). But if f has not been partitioned into the four modules, this approach will be useless because the Jacobian of f is dense. Indirect-addressing technique is an alternative approach that exploits the sparsity of Jacobian matrix. It uses sparse data structures for matrices and vectors during the computation, so that the computation related to 0 can be omitted. This approach is often not a good choice due to the additional overhead it incurs [1].

TABLE 1 NUMBER OF COMPUTATIONS DURING JACOBIAN EVALUATION

	FORWARD AD WITHOUT COMPRESSION	FORWARD AD WITH COMPRESSION	REVERSE AD	SD	FD
f_1	9	6	2	0	18
f_2	11	5	5	3	15
f_3	25	25	12	17	18

The efficiency of different algorithms for f_2, f_3 can be analyzed in the same way. Table 1 shows the numbers of scalar computations of different algorithms for f_1, f_2 and f_3 during Jacobian evaluations. The combination of similar terms occurs during the simplification of the symbolic derivatives of f_2, eliminating some computations. The eliminated computations, which are also called redundant computations in this paper, have to be carried out in AD algorithm. As for f_3, both SD and FD cost fewer computations than forward AD, but do not excel reverse AD. As is widely known, the ratio of time for Jacobian evaluation by forward AD to time for function evaluation is bounded from the above by $3n$ (n is the number of input variables). For reverse AD, the ratio is bounded by $3m$ (m is the number of output variables). In contrast, the ratio is about n for SD and FD [1]. In f_3, m=n/3, the reverse AD is undoubtedly the best choice. But in the situation where $m \geqslant n$, SD and FD may surpass AD no matter if it is in reverse mode or forward mode. For example, chemical process models usually have few degrees of freedom, where $m \approx n$. So AD is not necessarily the most efficient algorithm.

Besides the above disadvantages, AD has some other limitations. AD can't deal with the modules without source codes, such as proprietary model libraries and compiled legacy modules. Furthermore, the control statements and most assign statements in differentiation codes of AD are not a must for the Jacobian evaluation. For operator overloading AD tools, the overloading during the computation takes a considerable amount of time.

In summary, AD has the following drawbacks:

1. It cannot avoid redundant computations.
2. It cannot fully exploit the inner sparsity of the model.
3. For the model where $m \geq n$ (such as most chemical process models), SD and FD may outperform AD.
4. It cannot differentiate modules without source codes.
5. Additional operations performed by AD may be prohibitively expensive.

In contrast, SD or FD performs well on some of the above aspects. SD can eliminate all the redundant computations during the generation of symbolic derivatives, and can naturally exploit the sparsity of the model. FD does not require source code and is easy to implement. Both SD and FD introduce almost no additional operation and can obtain higher efficiency than AD in many cases.

However, either SD or FD has intrinsic limitations. SD has a problem of expression swell and often entails the repeated evaluation of common expressions. A discontinuous model, usually caused by non-smooth intrinsic functions or IF statements, requires several symbolic Jacobians for Jacobian evaluation, because the derivatives of the parts on either side of a discontinuous point obey different formulae. FD is not able to obtain exact derivatives, and has a problem deciding between truncation error and roundoff error. In addition, FD cannot exploit the sparsity of the model. Table 2 compares the three algorithms.

TABLE 2 THE COMPARE OF THE DIFFERENTIATION ALGORITHMS

	AD	SD	FD
Accuracy	High	High	Low
Memory requirement	Moderate	High	Low
Flexibility	Moderate	Low	High
Efficiency	$3m/3n$	n	n
Exploitation of sparsity	Partial	Full	None
Redundant computations	Many	None	Several

III. MODULE-ORIENTED AUTOMATIC DIFFERENTIATION

According to Table 1, differentiating f_1, f_2, f_3 by SD and f_4 by AD will achieve higher efficiency than differentiating f by AD. Even when AD is the only algorithm available, differentiating f_1, f_2, f_3 and f_4 separately is better because the inner sparsity of f can be exploited. The Jacobians of f_1, f_2, f_3 and f_4 can be accumulated to the overall Jacobian with the chain rule. Obviously, this approach follows the principle of AD; only the elementary function is replaced by modules. If traditional AD is regarded as a computation-oriented one, the new differentiation approach could correspondingly be called module-oriented automatic differentiation (MAD). In MAD, each module can be differentiated by AD, SD or FD according to its structure. Moreover, diverse differentiation codes offered by users, such as legacy codes or specific codes developed by the user for particular modules, can be easily incorporated into MAD.

In summary, MAD has at least three advantages:

1. It can exploit the inner sparsity of the model.
2. It can choose the best differentiation algorithm for each module.
3. It can conveniently utilize the external codes for derivative evaluation.

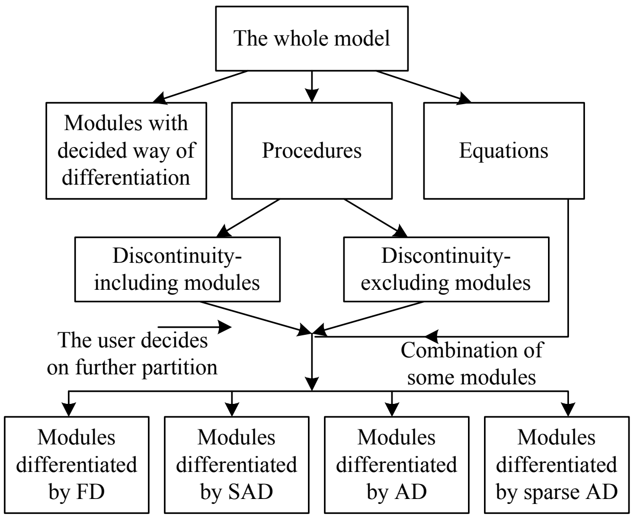

The whole model
Modules with decided way of differentiation
Procedures
Equations
Discontinuity-including modules
Discontinuity-excluding modules
The user decides on further partition
Combination of some modules
Modules differentiated by FD
Modules differentiated by SAD
Modules differentiated by AD
Modules differentiated by sparse AD

Fig.1 The approach to partition the model

The module-oriented automatic differentiation was first proposed by Xiang Li in 2004. The detailed concept, accumulating method of the derivatives of the modules, partition of the model, and choosing the appropriate differentiation approaches in MAD is similar with [4].

There are two approaches for traditional AD to accumulate derivatives: forward mode and reverse mode. In MAD, forward accumulation is used. The reverse accumulation is not recommended here. Because the additional overhead for storing and reading the record of every computation performed may be very costly indeed although for chemical process models $m \approx n$, the reverse AD can achieve similar efficiency as forward AD does. Thus the following discussion focuses on forward MAD.

A. Partitioning the model

The model can naturally be divided into procedures and equations. But properly partitioning the procedures themselves to exploit their inner variables is a tough job for computer to carry out automatically, because the interior structure of the procedures may be very complicated. So an approach which is implemented by the user is required for partitioning a procedure. If the Jacobians of the inner variables in a procedure are not sparse enough, the partitioning could be skipped.

The model can be partitioned into modules of interest in four steps. First, the modules that have a specific differentiation approach, such as legacy codes or specific codes for particular modules developed by the user, are removed from the model. Then the procedures and equations are set apart. Second, the procedures including discontinuities are taken out because they cannot be differentiated by SAD. The third step, which is discussed in the next section, chooses a differentiation algorithms for each module. In this step, the user can decide whether and how to partition a procedure according to the sparsity patterns of the Jacobians of its inner variables. Finally, the adjacent modules that adopt the same differentiation algorithm are combined. Fig. 1 shows the process of the partition.

B. Choosing differentiation approaches

In Section 2, the number of the scalar computations required for a differentiation algorithm is used to measure the efficiency of this algorithm. Similar criterion is used in this section for choosing differentiation approaches for the modules.

C. Summary and discussion

The MAD strategy can be used for Jacobian evaluations during the process systems optimization as follows: First, the process model is partitioned into a series of modules and the best differentiation algorithm for each of the modules is selected. Then the symbolic Jacobians of the modules differentiated by SAD is generated with the aid of a SAD library. For the other modules, the subroutines calling AD or

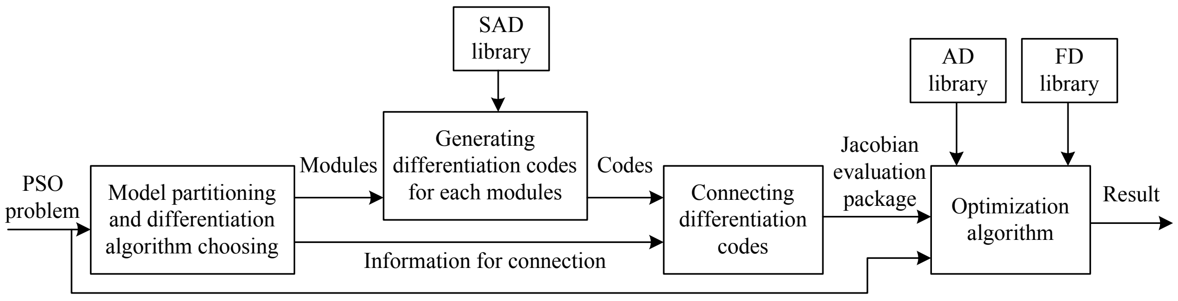
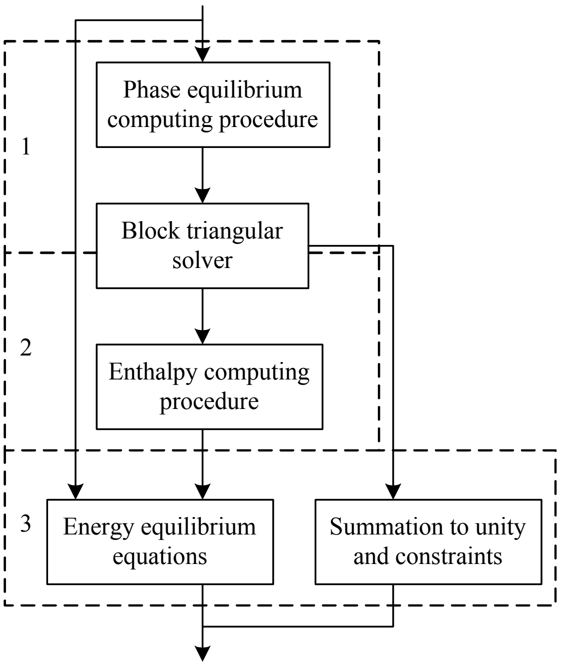

SAD library
AD library
FD library
PSO problem
Model partitioning and differentiation algorithm choosing
Modules
Generating differentiation codes for each modules
Codes
Connecting differentiation codes
Jacobian evaluation package
Optimization algorithm
Result
Information for connection

Fig. 2 MAD strategy in Jacobian evaluation

FD library is produced by the computer. In the third step, the Jacobian evaluation codes for the modules are accumulated into an overall Jacobian evaluation package, which is called by the optimization algorithm to compute the Jacobian of the model with the aid of AD and FD libraries. Fig. 2 illustrates the whole procedure.

Forward AD behaves like a differentiation approach that accumulates the Jacobians of the modules in forward mode and chooses forward AD to differentiate the modules. So forward MAD will never be outperformed by forward AD. In another words, the ratio of time for Jacobian evaluation by forward MAD to the time for function evaluation has a maximum upper bound of *3m*. The memory required by MAD is mainly used to store the symbolic derivatives. Setting an upper bound for the number of scalar computation by SAD as mentioned above can help to control the amount of memory required. If FD is employed for the differentiation of some modules, the accuracy of the overall Jacobian is difficult to analyze and control. In fact, controlling the accuracy of the FD itself is very difficult. When applied to the problems that require precise Jacobians, MAD should avoid using FD.



1
Phase equilibrium computing procedure
Block triangular solver
2
Enthalpy computing procedure
3
Energy equilibrium equations
Summation to unity and constraints

Fig. 3 Structure of the model of the distillation column

IV. NUMERICAL RESULTS

A. Lie derivatives

Consider a CSTR where the isothermal series/parallel Van de Vusse takes place [8]:

Reaction:

A→B→C $k_1 = 50h^{-1}$, $k_2 = 100h^{-1}$

2A→D $k_3 = 10h^{-1}$

Mass Balance:

$$V\frac{dCa}{dt} = F(Ca_0 - Ca) - k_1 CaV - k_3 Ca^2 V$$

$$V\frac{dCb}{dt} = F(-Cb) + k_1 CaV - k_2 CbV$$

steady states:

$Cas = 3.0 mol/l$

$Cbs = 1.117 mol/l$

Objective:

The control objective is to maintain *Cb* constant by manipulating the dilution rate *F/V*. By defining $x = (Ca, Cb)$, and $u = F/V$ (Control effort) we obtain.

$x_1' = -k_1 x_1 - k_3 x_1^2 + (ca0 - x_i)$

$x_2' = k_1 x_1 - k_2 x_2 - x_2 u$

$h(x) = alpha * [ca0/(ca0 - x_1)]$

$alpha = x_2(ca0 - x_1)/ca0$

1. The system has a relative degree *r=2* because $L_g L_f h \neq 0$.

2. For this example, we did the computation of the Lie derivatives for *r=1,2* and the hypothetical cases *r=3,4,5* to determine the CPU time required to produce the Lie derivatives, the size of the code in bytes, the size of the code in number of lines, and the simulation run time.

TABLE 3 THE PERFORMANCE OF AD, SD, AND MAD

RELATIVE ORDER	AD (CPU TIME, S)	SD (CPU TIME, S)	MAD (CPU TIME, S)
1	1.86	0.71	0.72
2	2.00	2.31	2.01
3	2.25	6.59	2.26
4	2.74	19.83	2.75
5	4.67	55.96	4.68

B. Jacobians evaluation

The example is an optimal control problem of an industrial distillation column first discussed by Zhang [9]. The objective of the optimization is to obtain the maximum production (ethylbenzene) value deducting the cost of the materials and the operation. The manipulated variables are reflux rate of the condenser and heat duty of the reboiler. There are 40 theoretical trays including the condenser and the reboiler. The feed flow, which contains 11 components, enters the column at the 25th tray. A composite model was constructed for the problem via Shao and Bailey et al.'s approaches described above.

Fig. 3 illustrates the structure of the model. Forward MAD was applied. The block triangular solver, which was used to replace the mass balance equations, was partitioned into two parts because of its sparsity. Then the equations and the procedures were grouped into three modules, represented by dashed frames in the figure. SAD was chosen for module 1, 3, and FD for module 2. Indirect approach was adopted to accumulate the derivatives for module 2, 3, because the input variables (n_2, n_3) of the modules are many more than the input variables (n) of the model.

The estimated numbers of scalar computations reflect the actual time reasonably well. So the MAD strategy employed SAD for module 1, 3 and FD for module 2. FD, AD and MAD were adopted respectively in the SQP algorithm during the

TABLE 4 OPTIMIZATION RESULTS OF DISTILLATION COLUMN.

DIFFERENTIATION ALGORITHM	ITERATIONS	TIME FOR ONE DIFFERENTIATION (S)	DIFFERENTIATION TIME (S)	OPTIMIZATION TIME (S)	RATIO*(%)
FD	17	18.69	295.38	304.81	96.91
AD	17	101.72	1618.78	1630.35	99.29
MAD	16	9.98	149.17	188.29	78.17

* denotes the ratio of differentiation time to optimization time

optimization. The running results are shown in Table 4. Obviously, MAD achieved the highest efficiency, and, because of the high ratio of differentiation time to optimization time, dramatically reduced the time for optimization.

V. CONCLUSIONS

A new differentiation strategy called MAD, employing the three differentiation algorithms flexibly and accumulating derivatives in a quasi-AD approach, can fully exploit the structure of the problem to improve the efficiency of Lie derivatives and Jacobian evaluations. Estimating the efficiency of a differentiation algorithm by counting the number of needed scalar computations can help to choose the best differentiation algorithm for a module. MAD can reduce the time used for evaluating the Lie derivatives and Jacobians. MAD is more like a methodology than specific algorithm. On the one hand, any differentiation algorithm can be taken as a tool for MAD. On the other hand, the user is required and encouraged to step in the execution of MAD to exploit more of hidden structure information inside module.

REFERENCES

[1] Tolsma, J. E., Barton, P. I. On computational differentiation. *Computers and Chemical Engineering, 22*, 475–490 (1998)

[2] Griewank, A. On automatic differentiation. In M. Iri & K. Tanabe (Eds.), *Mathematical Programming: Recent developments and applications,* 83–108. Kluwer Academic Publishers (1989)

[3] Quetzalcoatl M. J., Nonlinear control via automatic differentiation. Ph. D. Thesis, Case Western Reserve University (2002)

[4] Xiang Li, Zhijiang Shao, Jixin Qian. Module-oriented automatic differentiation in chemical process systems optimization. *Computers and Chemical Engineering, 28*, 1551–1561 (2004)

[5] Bischof C., Bouaricha A., Khademi P. More J., Computing gradients in large-scale optimization using automatic differentiation. *INFORMS Journal on computing*. 9, 185-194 (1997)

[6] Bischof C., Carle, A., Corliss G., Griewank A., Hovland P. ADIFOR-generating derivative codes from Fortran programs. *Scientific Programming*, 1, 1-29, (1992)

[7] Tolsma, J. E., Barton, P. I. Efficient calculation of sparse Jacobians. *SIAM Journal on Scientific Computing*. 20, 2282-2296 (1999)

[8] Berber Ridvan, Brosilow B. Coleman., Insight into the Relationships between Linear and Non-linear Model Based Control and Issues for Further Research, *Proceeding of the NATO Advanced Study Institute on Nonlinear Model Based Control*. Antalya, Turkey (1997)

[9] Zhang, Y., Chemical process simulation and real-time (on-line) optimization, Ph.D. Thesis, Zhejiang University (1998)