A New Algorithm Based on rSQP and AD

Jin Li, Yuejin Tan, Liangcai Liao

Abstract—**An efficient optimization algorithm based on reduced sequential quadratic programming (rSQP) and automatic differentiation (AD) is presented in this paper. With the characteristics of sparseness, relatively low degrees of freedom and equality constraints utilized, the nonlinear programming problem was solved by improved rSQP solver. In the solving process, AD technology was used to obtain accurate gradient information. The numerical results show that the combined algorithm, which is suitable for large-scale process optimization problems, can calculate more efficiently than rSQP itself.**

I. INTRODUCTION

Over the past 20 years, the successive quadratic programming (SQP) algorithm has been especially useful for a wide variety of processing applications. In particular, the reduced space SQP (rSQP) algorithm has seen many applications to large-scale engineering models. Designed for large NLP problems with few degrees of freedom, this approach has been implemented and applied to engineering problems ranging from flowsheet optimization, on-line process optimization, dynamic systems and even applications with PDE models [1,2].

One difficulty with large and complex optimization problems is the derivation. This is an essential requirement for using an efficient optimization algorithm. Not only rSQP but also other NLP algorithms require the calculation of the gradient vector of the first derivatives, and (depending on the method) perhaps the Hessian matrix of second derivatives as well. If the derivatives are calculated incorrectly, even in only one component, then the optimization method may either converge slowly (perhaps too slowly for practical use) or, more commonly, may fail to converge to a solution [3]. Hence it is essential to have accurate derivative calculations to solve optimization problems effectively.

The current Newton and quasi-Newton optimization software uses finite-difference approximation for computing the derivatives. When the model function is ill-conditioned, these approximation can be poor, and cause the optimization to fail. To evaluate the derivatives in rSQP algorithm efficiently, we use an automatic differentiation (AD) approach, the forward mode of which was first proposed by *R. E. Wengert* [4] and the reverse mode was first published by *G. M. Ostrowski* [5]

This research applies AD with rSQP and shows how the combined algorithm can improve the quality of optimization of large-scale nonlinear programs by computing the derivatives reliably and more accurately.

The introductory material and algorithm of rSQP is presented in Section 2. The description of automatic differentiation and corresponding algorithm used in rSQP is in Section 3, based on the introduction of four differentiation methods. In Section 4, the numerical results and conclusions are presented.

Manuscript received July 4, 2005. This work was supported by the Changling refinery of Sinopec.

Li Jin 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).

Tan Yuejin, 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).

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

II. REDUCED SEQUENTIAL QUADRATIC PROGRAMMING

The optimization algorithm developed by *Schmid* and *Biegler* [5] is designed to solve large process optimization problems of the form:

$$\min_{x\in R^n} f(x) \tag{1}$$

$$s.t.\ \ c(x)=0 \tag{2}$$

$$x^L \le x \le x^U \tag{3}$$

where $f : R^n \to R$ and $c : R^n \to R^m$ are assumed to be smooth functions with *n*, $m\times(n-m)$, and the first derivatives of *f* and *c* are available. The successive quadratic programming (SQP) method for solving (1)-(3) generates, at an iterate x_k, a search direction d_k by solving

$$\min_{d\in R^n} g(x_k)^T d + \frac{1}{2} d^T W(x_k) d \tag{4}$$

$$s.t.\ \ c(x_k)+A(x_k)^T d = 0 \tag{5}$$

$$x^L \le x_k + d \le x^U \tag{6}$$

where *g* denotes the gradient of *f*, $W(x)$ denotes the Hessian of the Lagrangian function $L(x,\lambda)=f(x)+\lambda^T c(x)$. *A* denotes the $n\times m$ matrix of constraint gradients

$$A(x_k)=[\nabla c_1(x),...,\nabla c_m(x)] \tag{7}$$

For brevity, we denote $A(x_k)$ as A_k, $g(x_k)$ as g_k, etc. A new iterate is then computed as

$$x_{k+1}=x_k+\alpha_k d_k \tag{8}$$

where α_k is a steplength parameter chosen so as to reduce the value of the merit function. A merit function is used to balance the two goals of decreasing the objective function (1) and satisfying the constraints (2) of the NLP.

The quadratic programming (QP) subproblem (4)-(6) can be reduced to solving a smaller QP in the space of the independent variables. This is done by introducing a nonsingular matrix of order n, given by $[Y_k \quad Z_k]$ where $Y_k \in R^{n\times m}$ and $Z_k \in R^{n\times(n-m)}$, and $A_z^T Z_k = 0$. Thus, Z_k is a basis of the tangent space of the constraints. The solution to (4)-(6), can then be expressed as

$$d_k = Y_k p_Y + Z_k p_Z \qquad (9)$$

for some vectors $p_Y \in R^m$ and $p_Z \in R^{n-m}$. From above, the linear constraints (5) become

$$c_k + A_k^T Y_k p_Y = 0 \qquad (10)$$

If we assume that A_k has full column rank then the nonsigularity of $[Y_k \quad Z_k]$ implies that the matrix $A_k^T Y_k$ is nonsingular, so that p_Y is determined by

$$p_Y = -[A_k^T Y_k]^{-1} c_k \qquad (11)$$

To form the basis matrices Y and Z, we induce the partition

$$A(x)^T = [N(x) \quad C(x)] \qquad (12)$$

where the $m\times m$ basis matrix $C(x)$ is assumed to be nonsingular. $Z(x)$ and $Y(x)$ are now defined to be

$$Z(x) = \begin{bmatrix} I \\ -C(x)^{-1}N(x) \end{bmatrix},\ Y(x) = \begin{bmatrix} 0 \\ I \end{bmatrix} \qquad (13)$$

This choice is particularly popular and advantageous when $A(x)$ is large and sparse, because a sparse LU decomposition of $C(x)$ can be computed efficiently.

The QP subproblem can then be expressed exclusively in terms of the variables p_Z. Substituting (9) into (4)-(6) with $Y_k p_Y$ determined, gives the QP:

$$\min_{p_Z \in R^{n-m}} (Z_k^T g_k + w_k)^T p_Z + \frac{1}{2} p_Z^T B_k p_Z \qquad (14)$$

$$s.t.\ x^L - x_k - Y_k p_Y \le Z_k p_Z \le x^U - x_k - Y_k p_Y \qquad (15)$$

where B_k approximates $Z_k^T W_k Z_k$ and w_k approximates $Z_k^T W_k Y_k p_Y$. Comparing the second-order terms B_k and w_k form (14) and W_k from (4) shows that this decomposition reduces the matrix sizes in the QP from $n\times n$ to $(n-m)\times(n-m)$.

We can now outline the sequential quadratic programming method, but the above algorithm has been left in a very general form. We consider the choice of basis matrices Y_k and Z_k, the correction terms w_k and $\bar{w}_k$, the conditions under which BFGS updating takes place, the choice of the damping parameter ζ_k, and the procedure for updating the weight μ_k in the merit function. So the detailed description of algorithm, which is used in this research and mainly based on [1], is given as follows.

Algorithm Ⅰ

1. Choose constants $\eta \in (0,1/2)$ and τ, τ' with $0 < \tau < \tau' < 1$, and positive constants Γ for conditions

$$w_k := \begin{cases} w_k & \text{if } \|w_k\| \le \dfrac{\Gamma}{\|p_Y\|^{1/2}} \|p_Y\| \\ w_k \dfrac{\Gamma \|p_Y\|^{1/2}}{\|w_k\|} & \text{otherwise} \end{cases} \qquad (16)$$

and γ_{fd} for conditions

$$\|p_Y\| \le \gamma_{fd} \|p_Z\| / \sigma^{1/2} \qquad (17)$$

For $\bar{w}_k := \begin{cases} \bar{w}_k & \text{if } \|\bar{w}_k\| \le \alpha_k \|p_Y\| / \gamma_k \\ \bar{w}_k \dfrac{\alpha_k \|p_Y\|}{\gamma_k \|\bar{w}_k\|} & \text{otherwise} \end{cases} \qquad (18)$

and $\|p_Y\| \le \gamma_k^2 \|p_Z\| \qquad (19)$

select a summable sequence of positive numbers $\{\gamma_k\}$.

Set $k := 1$, and choose a starting point x_1, an initial value μ_1 for the penalty parameter, an $(n-m)\times(n-m)$ symmetric and positive definite starting matrix B_1 and an $(n-m)\times n$ starting matrix S_1.

2. Evaluate f_k, g_k, c_k, and A_k, and compute Y_k and Z_k.

3. Set *findiff=false* and compute p_Y by solving the system $(A_k^T Y_k) p_Y = -c_k$. (range space step) (20)

4. Calculate w_k using Broyden's method, from $w_k = S_k Y_k p_Y$ and $\bar{w}_k = \alpha_k S_{k+1} Y_k p_Y$ and (16).

5. Choose the damping parameter ζ_k from

$$\zeta_k [2\cos\theta_k |g_k^T Z_k w_k| + w_k^T B_k^{-1} Z_k^T g_k + \zeta_k w_k^T B_k^{-1} w_k] \le \rho \|c_k\|_1 \qquad (21)$$

and

$$\zeta_k = \min\{1, \zeta_k\} \qquad (22)$$

and compute p_Z from

$$B_k p_Z = -[Z_k^T g_k + \zeta_k w_k]. \quad \text{(null space step)} \qquad (23)$$

6. If (17) is satisfied and (19) is *not* satisfied, set *findiff=true* and recomputed w_k from

$$w_k = Z_k^T [\nabla L(x_k + Y_k p_Y, \lambda_k) - \nabla L(x_k, \lambda_k)] \qquad (24)$$

7. If *findiff=true*, use this new value of w_k to choose the damping parameter ζ_k from Equations (21) and (22), and recomputed p_Z from (23)

8. Define the search direction by $d_k = Y_k p_Y + Z_k p_Z$ and set $\alpha_k = 1$.

9. Test the line search condition

$$\phi_{\mu_k}(x_k + \alpha_k d_k) \le \phi_{\mu_k}(x_k) + \eta \alpha_k D\phi_{\mu_k}(x_k; d_k) \qquad (25)$$

10. If (25) is not satisfied, choose a new $\alpha_k \in [\tau\alpha_k, \tau'\alpha_k]$ and go to 9; otherwise set

$$x_{k+1} = x_k + \alpha_k d_k \qquad (26)$$

11. Evaluate f_{k+1}, g_{k+1}, c_{k+1}, and A_{k+1}, and compute Y_{k+1} and Z_{k+1}.

12. Compute the Lagrange multiplier estimate

$$\lambda_{k+1} = -[Y_{k+1}^T A_{k+1}]^{-1} Y_{k+1}^T g_{k+1} \qquad (27)$$

and update μ_k so as to satisfy

$$\mu_k = \begin{cases} \mu_{k-1} & \text{if } \mu_{k-1} \ge \|\lambda_k\|_\infty + 2\rho \\ \|\lambda_k\|_\infty + 3\rho & \text{otherwise} \end{cases} \qquad (28)$$

13. Update S_{k+1} using

$$S_{k+1} = S_k + \frac{(\bar{y}_k - S_k\bar{s}_k)\bar{s}_k^T}{\bar{s}_k^T\bar{s}} \quad (29)$$

where $\bar{y}_k = Z_k^T[\nabla L(x_{k+1}, \lambda_{k+1}) - \nabla L(x_k, \lambda_{k+1})]$ (30)

and $\bar{s}_k = x_{k+1} - x_k$ (31)

If *findiff=false*, calculate $\bar{w}_k$ by Broyden's method through

$w_k = S_k Y_k p_Y$ and $\bar{w}_k = \alpha_k S_{k+1} Y_k p_Y$ (32)

otherwise calculate $\bar{w}_k$ by

$$\bar{w}_k = Z_k^T[\nabla L(x_k + \alpha_k Y_k p_Y, \lambda_{k+1}) - \nabla L(x_k, \lambda_{k+1})] \quad (33)$$

14. If $(s_k^T y_k \le 0)$ or if (*findiff=true* and (17) is not satisfied) or if (*findiff=false* and (19) is not satisfied), set $B_{k+1} = B_k$. Else, compute

$s_k = \alpha_k p_Z$, (34)

$y_k = Z_k^T[\nabla L(x_{k+1}, \lambda_{k+1}) - \nabla L(x_k, \lambda_{k+1})] - \bar{w}_k$ (35)

and compute B_{k+1} by the BFGS formula

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k} \quad (36)$$

15. Set $k := k+1$, and go to (3).

The above algorithm is implement based on rSQP++ [7], an objected-oriented framework for solving NLPs using SQP method. The framework is being developed to support primarily rSQP.

III. Differentiation Methods

In this section, A variety of differentiation methods for computing the derivatives of the function *f*(*x*) are considered. There are four basic methods of differentiation as described below [8].

A. Hand-Coded

Computation of derivatives by hand is difficult and time consuming, especially when the problem becomes more complex and/or increases in size so that the evaluation of gradients and Hessians become expensive both in time and storage.

B. Finite-Differencing

The derivative of function *f*(*x*) with respect to the *i*th component of *x* at a point x_0 is approximated by

$$\left.\frac{\partial f(x)}{\partial x_i}\right|_{x=x_0} \approx \frac{f(x_0 + he_i) - f(x_0)}{h} \quad (37)$$

here e_i is the *i*th Cartesian basis vector. The difficulty with this method is that its accuracy is hard to measure and the selection of an appropriate value for *h* can cause convergence problems when high accuracy solutions are desired, particularly on ill-conditioned problems.

C. Symbolic Differentiation

Symbolic differentiation packages such as Maple, Reduce, and Macsyma produce exact derivatives. Consider the function

$f(x) = x_1 x_2 x_3 x_4 x_5 x_6$ (38)

we have
$$\nabla f(x) = \begin{pmatrix} x_2x_3x_4x_5x_6 \\ x_1x_3x_4x_5x_6 \\ x_1x_2x_4x_5x_6 \\ x_1x_2x_3x_5x_6 \\ x_1x_2x_3x_4x_6 \\ x_1x_2x_3x_4x_5 \end{pmatrix} \quad (39)$$

This is correct, but it is not a very efficient method to compute the derivatives of the function *f*(*x*). Since there are many common sub-expression in the different derivative expressions, a fact not commonly taken into account by these packages. In addition, symbolic differentiation cannot handle functions that are evaluated using branches or loops.

D. Automatic Differentiation

Automatic Differentiation is a method for the automatic generation of derivatives. It provides numerical derivative values at given arguments. It is based on the application of the chain rule and is a useful and efficient method for evaluating a function with a large number of derivatives. Differentiation of a function, no matter how complex it might be in form, can be performed automatically. Consider an algorithm *A* which transform some input (*a*, *b*, *c*,…) into some output (*u*, *v*,…). This can be shown as:

$a, b, c, \ldots \rightarrow A \rightarrow u, v, \ldots$

The objective of automatic differentiation is to transform the algorithm *A* into an algorithm *A*' which is more powerful than *A* in the sense that, in addition to *u*, *v*,… it can produce derivative values:

$$a, b, c, \ldots \rightarrow A \rightarrow u, v, \frac{\partial u}{\partial a}, \ldots$$

The transformation of *A* to *A*' could be accomplished by an algorithm *Diff*:

$A \rightarrow Diff \rightarrow A'$

Automatic differentiation can be applied, not only to function defined by a simple expression, but also to function that are defined in terms of other algorithms.

Automatic differentiation is based on the fact that every function, no matter how complicated, is executed on a computer as a sequence of elementary operations. By applying the chain rule

$\frac{\partial y}{\partial t} = \frac{\partial y}{\partial x} \cdot \frac{\partial x}{\partial t}$ where $y = f(x)$ and $x = g(t)$

to compositions of these elementary functions over and over again, one can compute the derivative information for a function *f*(*x*) exactly and efficiently.

There are basically two different ways of applying the chain rule. They are called forward and backward differentiation. In forward mode the derivatives with respect to the independent variables are maintained. The backward mode of automatic differentiation maintains the derivatives of the final result with respect to intermediate quantities. These values, usually referred to as adjoints, measure the sensitivity of the final result with respect to some intermediate quantities. The backward method is more complicated, in that one must propagate derivative information from the final result back to the original independent variables, one derivative at a time.

But for computing first derivatives, the backward mode is faster than the forward mode. In theory, backward automatic differentiation computes function and gradient values in time proportional to that for computing just the function. If the evaluation of function involves N operations then that of the gradient involves qN operations where q is at most 5.

1) Forward Differentiation Algorithm

This mode of differentiation follows the algorithm

Algorithm Ⅱ

```
For $i=1,\ldots,n$
    $x_i = e_i$
For $i=1$ to $M$
    $x_{n+i} = F_{n+i}(x_j, x_k)$
    $\nabla x_{n+i} = \nabla F_{n+i}(x_j, x_k)$
    $f = x_{n+M}$
    $\nabla f = \nabla x_{n+M}$
```

2) Backward Differentiation Algorithm

This mode of differentiation follows the algorithm

Algorithm Ⅲ

```
For $i=1,\ldots,n$
    $\bar{x}_i = 0$
For $i=1$ to $M$
    $x_{n+i} = F_{n+i}(x_j, x_k)$
    $\bar{x}_{n+i} = 0$
    $f = x_{n+M}$
    $\bar{x}_{n+M} = 1$
For $i=M$ to 1
    $\bar{x}_j = \bar{x}_j + \frac{\partial F_{n+i}}{\partial x_j}\bar{x}_{n+i}$
    $\bar{x}_k = \bar{x}_k + \frac{\partial F_{n+i}}{\partial x_k}\bar{x}_{n+i}$
  For $i=1$ to $n$
    $\nabla f_i = \bar{x}_i$
where $\bar{x}_i = \frac{\partial x_{n+M}}{\partial x_i} = \frac{\partial f(x)}{\partial x_i}$
```

The computational process can be represented as a computational graph. It represents the computation of $f(x)$ in terms of elementary operations (like +, −, *, /) and standard library functions. Each vertex of the computational graph is an input variable, a constant or an intermediate variable, and each are corresponds to an elementary partial derivatives. The user needs only to program the automatic differentiation software then produces a new program that calculates both the function and gradient.

In backward differentiation only the representation of the original function is needed. It is faster than the forward mode to evaluate the gradient of the function $f(x)$, but involves the storage of the computational graph. The large storage requirement can possibly make this technique infeasible on small computer systems and/or large problem.

E. AD used in rSQP

There are already available some software package for automatic differentiation. Among these I can name GRESS [9], JAKEF [9], PADRE2 [9], ADIFOR[9], and ADOL-C [10]. GRESS (Gradient Enhanced Software System) is a precompiler that produces code that propagates first-order partial derivatives using either the forward or reverse mode. The GRESS package is a system that makes it possible to perform comprehensive sensitivity analysis of Fortran models. JAKEF is another Fortran precomplier. JAKEF is a version of JAKE that was developed at Argonne National Laboratory and written using Fortran 77. Input to JAKEF is a Fortran subroutine and then JAKEF generates a Fortran subroutine to compute the gradient or Jacobian of the function. JAKEF uses the reverse mode to propagate a Fortran function and then produces a modified subroutine. This subroutine calculates the original function, its partial derivatives, and an estimate of the rounding error generated. PADRE2 can be used to calculate first and second partial derivatives in either mode. I consider ADOL-C for my research since it is a package for AD of algorithm written in C/C++, for I use rSQP++, an object-oriented framework for solving NLPs using SQP methods. The ADOL-C is based on operator overloading. Using this technique, one can log for each operation during the program execution the operator and the variables that are involved. Hence, one obtains a new internal representation of the function evaluation. Based on the generated execution log ADOL-C computes the desired derivatives.

IV. Numerical Results and Conclusions

The alkylation process optimization [11] of Bracken and McCormick is a famous process optimization problem solved by Westerberg et al., Berna et al., and Vasantharajan et al. The process is illustrated schematically in Fig.1. There are 10 variables (all of them have upper and lower bounds), 3 equality constraints, and 8 inequality constraints in this problem. Details of modeling and mathematical formulation can be found in Berna et al.

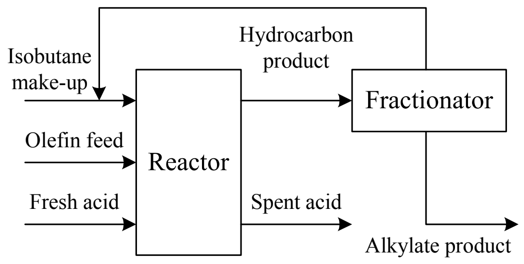

Fig. 1. Bracken and McCormick alkylation process

Two different initial states were specified for the optimization: one is identical with Westerberg et al., and the other is retrieved form Berna et al. and Vasantharajan et al. Associated with different initial states, the problem was designated as problems 1a and 1b, respectively.

Problem 2 is a distillation process optimization originally developed by Shao et al [12]. The distillation model considered in this problem describes a benzene-toluene system in which there are 18 theoretical trays. Further details can be found in Shao [12]. Different from the model considered by Shao, however, the model equation in this paper were written in an open-equation format which is similar to the model considered in [11]. To break the internal convergence loop for thermodynamic property computation in the model of Shao, equations from Bailey [13] were used for regressing phase equilibrium ratio K and enthalpy values. The corresponding coefficients were obtained based on data generated from PRO/Ⅱ simulation programs. All equations were converted into equality equations with slack variables. The resulting optimization has 290 variables, 288 equality constraints, and upper and lower bounds for all variables.

TABLE 1 COMPARISON OF PERFORMANCE ON TEST PROBLEMS

PROBLEM	ALGORITHM Ⅰ		rSQP WITH AD	
	CPU TIME, S	ITERATIONS	CPU TIME, S	ITERATIONS
1a	1.933	22	0.973	9
1b	2.693	32	2.224	18
2	655.877	14	67.017	10

The combined algorithm is used to solve the optimization problem 1 and 2 and the numerical results are listed in Table1. From Table 1, it is clearly shown that the combined algorithm requires fewer iterations than algorithm Ⅰ. Reduction of about 30% in CPU time is also observed. These encouraging results indicate that the combined algorithm performance is improved significantly due to more efficient and accurate curvature information provided by MAD. It is also shown that, no matter how the size of optimization problems would be, accurate derivatives are consistently beneficial to the optimization.

The significant enhancement in speed on two problems with distinct difference in size (10 and 290 variables) shows that the strategy is an approach suitable for large-scale process optimization problems.

REFERENCES

[1] Biegler L. T., Nocedal J., Schmid C., A reduced Hessian method for large-scale constrained optimization. *SIAM Journal of Optimization*, 1995, 5,314

[2] Ternet D. J., Biegler L. T., Recent improvements to a multiplier-free reduced Hessian successive quadratic programming algorithm. *Computers and Chemical Engineering,* 1998, 22(7-8): 963-978

[3] Fateh H., Automatic differentiation for large-scale nonlinear programming problems. *PhD GEORGE MASON UNIVERSITY*, 1995

[4] R. E. Wengert. A Simple Automatic Derivative Evaluation Program. Comm. *ACM*, 1964(7): 463-464

[5] G. M. Ostroskii, J. M. Wolin, W. W. Borisov. Uber die Berechning Von Ableitungen. Wissenschaftliche Zeitschrift der Technischen Hochschule fur Chemie, *Leuna-Merseburg*, 1971(13): 382-384

[6] Biegler L. T., Nocedal J., Schmid C., A reduced Hessian method for large-scale constrained optimization. SIAM Journal of Optimization, 1995, 5(2): 314-347

[7] Roscoe A. Bartlett, rSQP++ An Object-Oriented Framework for Reduced Space Successive Quadratic Programming. *http://dynopt.cheme.cmu.edu/roscoe/rSQPppOview/ rSQPppOview.ps*

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

[9] D. W. Juedes. A Taxonomy of Automatic Differentiation Tools. *SIAM*, 1991:315-329

[10] Andreas Griewank, David Juedes, Hristo Mitev, Jean Utke, Olaf Vogel, Andrea Walther, ADOL-C: A Package for the Automatic Differentiation of Algorithms Written in C/C++, Version 1.8.2, March 1999.

[11] Zhong Weitao, Shao Zhijiang, Zhang Yuyue, Qian Jixin, Applying Analytical Derivative and Sparse Matrix Techniques to Large-Scale Process Optimization, *Chinese Journal of Chemical Engineering*, 2000, 8(3): 212-217

[12] Shao Z. J. On-line Optimization of Continuous Industrial Process. *Ph.D. Thesis. Zhejiang University*, 1997. (in Chinese)

[13] Jiang Aipeng, Shao Zhijiang, Qian Jixin, Optimization of reaction parameters based on rSQP and hybrid automatic differentiation algorithm, *Journal of Zhejiang University (Engineering Science)*, 2004, 38(12): 1606-1610 (in Chinese)