Matematika, 2003, Jilid 19, bil. 2, hlm. 107–119

Numerical Experiments with Matrices Storage Free BFGS Method for Large Scale Unconstrained Optimization

Malik Hj. Abu Hassan, Mansor Monsi & Leong Wah June
Department of Mathematics
University Putra Malaysia
43400 Serdang
Selangor, Malaysia

Abstract We study the numerical performance of a matrices storage free quasi-Newton method for large-scale optimization, which we call the F-BFGS method. We compare its performance with that of the limited memory BFGS, L-BFGS methods developed by Nocedal (1980) and the conjugate gradient methods. The F-BFGS method is very competitive due to its low storage requirement and computational labor and also able to solve large-scale problems with 10^6 variables successfully while other methods fail.

Keywords Large scale optimization, matrices storage free methods, limited memory methods, conjugate gradient methods.

Abstrak *Kami mengaji prestasi berangka bagi suatu kaedah kuasi-Newton yang bebas storan matriks untuk pengoptimuman berskala besar, yang kami panggil kaedah F-BFGS. Kami membandingkan prestasinya dengan kaedah memori terhad BFGS, iaitu kaedah L-BFGS yang dibangunkan oleh Nocedal (1980) dan kaedah kecerunan konjugat. Faedah F-BFGS mempunyai saingan yang inggi disebabkan oleh keperluan storan dan kerja pengiraan yang rendah serta berupaya menyelesaikan masalah berskala besar dengan* 10^6 *pembolehubah dengan jayanya sedangkan kaedah lain gagal.*

Katakunci *Pengoptimuman berskala besar, kaedah bebas storan matriks, kaedah memori terhad, kaedah kecerunan konjugat.*

AMS subject classification. 65K10, 65H10

1 Introduction

Research in large scale optimization has been, in recent years, a major subject of interest within the mathematical programming community, in particular the large scale uncon-

strained optimization:

$$\underset{x\in R^n}{\text{minimize}}\ f(x) \tag{1}$$

where the number of variables n is large, and the analytic expressions for the function f and the gradient g are available. This paper is devoted to the numerical study of some methods for large scale optimization. We briefly review the methods to be tested in §2, where we also describe the problems used in our experiments. We present the numerical results of F-BFGS method and limited memory methods, paying particular attention to storage locations in §3. In §4 we compare the F-BFGS method with two well-known conjugate gradient methods. In §5 we explore the performance of these methods on very large variables number of the tested examples.

2 Preliminaries

2.1 F-BFGS Method

The matrices storage free BFGS method is an adaptation of the BFGS method to large problems. If we denote the iterate by x_k, and define $s_k = x_{k+1} - x_k$ and $y_k = g_{k+1} - g_k$, the implementation described by Malik et al. (2000) is almost identical to that of the standard BFGS method—the only difference is in the matrix update. Instead of storing the matrices H_k, one stores the latest pair of $\{s_i, y_i\}$ that defines them implicitly. The product $H_k g_k$ is obtained by performing a sequence of inner products involving g_k and the most recent vector pair $\{s_i, y_i\}$. After computing the new iterate, the older pair is deleted, and is replaced by the newest one. The algorithm therefore always keeps only the recent pair $\{s_i, y_i\}$ to define the iteration matrix. This approach is suitable for large scale problems because it only requires vector storage for each s_k and y_k. Let us describe the updating process in more detail as follows.

Algorithm 2.1. F-BFGS method

Step 1. Choose x_0, $0 < \beta' < \frac{1}{2}$, $\beta' < \beta < 1$, and initial matrix $H_0 = H_0^{(0)} = I$.
Compute
$d_0 = -g_0$
and
$x_1 = x_0 + \lambda_0 d_0$
where λ_0 satisfies the Wolfe conditions (3)–(4) (the steplength $\lambda = 1$ is tried first).

Step 2. For $k > 0$, compute $s_{k-1}^T y_{k-1}$, $y_{k-1}^T y_{k-1}$, $s_{k-1}^T g_k$ and also

$$\begin{aligned} \rho_{k-1} &= 1/y_{k-1}^T s_{k-1}, \\ \delta_k &= s_{k-1}^T y_{k-1}/y_{k-1}^T y_{k-1}, \\ p &= \rho_{k-1}^2(1/\rho_{k-1} + \delta_k y_{k-1}^T y_{k-1})(s_{k-1}^T g_k) - \rho_{k-1}\delta_k y_{k-1}^T g_k - \rho_{k-1} s_{k-1}^T g_k \end{aligned} \tag{2}$$

Step 3. Compute

$$H_k g_k = \delta_k g_k + [s_{k-1} \quad \delta_{k-1} y_{k-1}]p$$

$d_k = -H_k g_k$

Then, compute

$x_{k+1} = x_k + \lambda_k d_k$

where λ_k satisfies the Wolfe conditions:

$$f(x_k + \lambda_k d_k) \leq f(x_k) + \beta' \lambda_k g_k^T d_k \quad (3)$$
$$g(x_k + \lambda_k d_k)^T d_k \geq \beta g_k^T d_k \quad (4)$$

Step 4. Set $H_k^{(0)} = \delta_k I,\ k := k + 1$, and go to step 2.

2.2 L-BFGS Method

The limited memory BFGS method (L-BFGS) is described by Nocedal (1980), where it is called the SQN method. The user specifies the number m of BFGS corrections that are to be kept, and provides a sparse symmetric and positive definite matrix H_0, which approximates the inverse Hessian of f. During the first m iterations the method is identical to the BFGS method. For $k > m$, H_k is obtained by applying m BFGS updates to H_0 using information from the m previous iterations. The method uses the inverse BFGS formula in the form

$$H_{k+1} = V_k^T H_k V_k + \rho_k s_k s_k^T \quad (5)$$

where

$$\rho_k = 1/y_k^T s_k, \quad V_k = I - \rho_k y_k s_k^T \quad (6)$$

(see Dennis and Schnabel, 1983.)

Algorithm 2.2. L-BFGS method

Step 1. Choose X_0, $0 < \beta' < \frac{1}{2}$, $\beta' < \beta < 1$, and initial matrix $H_0 = I$. Set $k = 0$

Step 2. Compute

$d_k = -H_k g_k$

and

$x_{k+1} = x_k + \lambda_k d_k$

where λ_k satisfies (3)-(4) (the steplength $\lambda = 1$ is tried first).

Step 3. Let $\hat{m} = \min\{k, m-1\}$. Update H_0 for $\hat{m} + 1$ times by using the pairs $\{y_i, s_j\}_{j=k-\hat{m}}^{k}$, i.e let

$$\begin{aligned} H_{k+1} &= (V_k^T \cdots V_{k-\hat{m}}^T) H_0 (V_{k-\hat{m}} \cdots V_k) \\ &+ \rho_{k-\hat{m}} (V_k^T \cdots V_{k-\hat{m}+1}^T)\, s_{k-\hat{m}} s_{k-\hat{m}}^T (V_{k-\hat{m}+1} \cdots V_k) \\ &+ \rho_{k-\hat{m}+1} (V_k^T \cdots V_{k-\hat{m}+2}^T)\, s_{k-\hat{m}+1} (V_{k-\hat{m}+2} \cdots V_k) \\ &\vdots \\ &\rho_k s_k s_k^T \end{aligned} \tag{7}$$

Step 4. Set $k := k + 1$, and go to Step 2

2.3 Conjugate Gradient Methods

In connection with unconstrained problems we consider conjugate gradient algorithms of the form

$x_{k+1} = x_k + \lambda_k d_k$
with

$$d_i = \begin{cases} -g_k & \text{for } k = 0 \\ -g_k + \alpha_k d_{k-1} & \text{for } k \geq 1 \end{cases} \tag{8}$$

where α_k is a scalar representing different methods.
The best-known formulae for α_k are

$$\alpha_k^{\mathrm{FR}} = \frac{\|g_k\|^2}{\|g_{k-1}\|^2} \tag{9}$$

introduced by Fletcher and Reeve, FR(1964)
and

$$\alpha_k^{\mathrm{FR}} = \frac{g_k^T (g_k - g_{k-1})}{\|g_{k-1}\|^2} \tag{10}$$

is introduced by Polak and Ribire, PR(1969). A very simple device for enforcing global convergence is that of using a periodic restart (regular restart) along the negative gradient direction.

2.4 Test Problems and Termination

The evaluation of optimization algorithms on large scale test problems is more difficult than in the small dimensional case. When the number of variables is very large (in the hundreds and thousands), the computational effort of the iteration sometimes dominates the cost of evaluating the function and gradient. However there are also many practical large scale problems for which the function evaluation is exceedingly expensive. In most of our test problems, the function evaluation is inexpensive. The 6 test problems we used with dimensions of 8, 200 and 1000, are Penalty I (Gill and Murray, 1979); Trigonometric, Extended Rosenbrook, Powell, Beale and Wood (Mor et al., 1981).

For all the problems we used the standard starting points given in the references. The termination criteria for all iterations is

$$\|g_k\| < 10^{-5} \times \max(1,\ \|x_k\|) \tag{11}$$

3 Comparison with the Limited Memory BFGS Method

In this section we compare the L-BFGS method with the F-BFGS method. In L-BFGS method the user specifies the amount of storage to be used, by giving a number m, which determines the number of matrix updates that can be stored. Harwell subroutine VA15 is used to implement the L-BFGS method. In both procedures the line search is terminated when (2.2) and

$$|g(x_k + \lambda_k d_k)^T d_k| \leq \beta g_k^T d_k \tag{12}$$

are satisfied ((12) is stronger than (4), which is useful in practice). We use the values $\beta' = 10^{-4}$ and $\beta = 0.9$, which are recommended by Nocedal (1980). All other parameters in the Nocedal procedure were set to their default values, and therefore the method was tested precisely as they recommend. In both methods we use a line search routine based on the cubic interpolation, developed by Mor and Thuente (1994). The initial Hessian is approximated by the identity matrix, and after one iteration is completed, all methods update $\gamma_0 I$ instead of I, where

$$\gamma_0 = y_0^T s_0 / \|y_0\|^2 \tag{13}$$

This is a simple and effective way of introducing a scale in the algorithm (see Shanno and Phua, 1978). Table 1 gives the amount of storage required by the limited memory methods for various values of m and n, and compares it to the storage required by the F-BFGS method.

Table 1. Storage Location

n	F-BFGS	L-BFGS(m=3)	L-BFGS(m=7)
8	42	78	140
200	1002	1806	3414
1000	5002	9006	17014

In the following tables, n denotes the number of variables, n_I the number of iterations, n_f the number of function evaluations, n_g the number of gradient evaluations, m the number of updates allowed. "runtime" includes the time needed to generate the search direction, perform the line search and test convergence and the time to evaluate the function and gradient. For all methods the number of gradient evaluations equals the number of function evaluations. We also define the index of computational labor per iteration, **ICL** (Wolfe, 1978) as

$$\mathbf{ICL} = \frac{n_f + n n_g}{n_I} = \frac{(n+1) n_f}{n_I}.$$

In Table 2 we compare the performance of the limited memory methods when $m = 3$, 7 and the F-BFGS.

For $m = 3$ the two methods are comparable, and we see that as m increases, the differences between the two become large. However for Powell and Wood, functions with

Table 2. F-BFGS vs L-BFGS

	F-BFGS			F-BFGS, $m = 3$			F-BFGS $m = 7$		
	n_I	n_f	runtime	n_I	n_f	runtime	n_I	n_f	runtime
Penalty I									
$n = 8$	39	51	0.16	53	74	0.21	48	60	0.21
$n = 200$	57	74	0.21	61	74	0.32	57	69	0.32
$n = 1000$	68	87	0.36	64	75	0.43	63	75	0.49
Trigonometric									
$n = 8$	29	41	0.10	27	33	0.16	23	28	0.16
$n = 200$	48	58	0.21	56	64	0.32	49	58	0.27
$n = 1000$	50	61	0.43	54	64	0.54	44	51	0.49
RosenBrook									
$n = 8$	42	59	0.16	37	47	0.21	36	45	0.21
$n = 200$	39	61	0.16	37	53	0.21	37	48	0.27
$n = 1000$	36	61	0.21	35	49	0.27	36	45	0.32
Powell									
$n = 8$	184	239	0.65	46	54	0.21	40	44	0.16
$n = 200$	221	278	0.82	42	53	0.21	139	293	0.60
$n = 1000$	131	166	0.71	–	–	–	–	–	–
Wood									
$n = 8$	135	167	0.49	65	88	0.27	48	59	0.21
$n = 200$	181	226	0.65	65	89	0.27	114	137	0.49
$n = 1000$	125	162	0.71	–	–	–	–	–	–
Beale									
$n = 8$	18	21	0.10	15	18	0.16	14	16	0.16
$n = 200$	20	27	0.16	13	14	0.16	16	17	0.16
$n = 1000$	16	19	0.16	13	15	0.16	15	16	0.16

Table 3

	Index of Computational Labour (ICL)		
	F-BFGS	L-BFGS $m = 3$	L-BFGS $m = 7$
Penalty I			
$n = 8$	11.77	12.23	11.25
$n = 200$	206.95	243.84	243.32
$n = 1000$	1280.69	1173.05	1191.67
Trigonometric			
$n = 8$	12.72	11.00	10.96
$n = 200$	242.88	229.71	237.92
$n = 1000$	1221.22	1186.37	1160.25
RosenBook			
$n = 8$	12.64	11.43	11.25
$n = 200$	314.39	242.23	260.76
$n = 1000$	1696.14	1140.67	1251.25
Powell			
$n = 8$	11.69	10.57	9.90
$n = 200$	254.84	253.64	423.69
$n = 1000$	1268.44	–	–
Wood			
$n = 8$	11.13	12.18	11.06
$n = 200$	250.97	275.22	241.55
$n = 1000$	1297.30	–	–
Beale			
$n = 8$	10.50	10.80	10.29
$n = 200$	271.35	216.46	213.56
$n = 1000$	1188.69	1155.00	1067.73

$n = 1000$, both L-BFGS methods fail to converge to the minimizer. In terms of computational effort per iteration, F-BFGS is competitive with the two L-BFGS methods because their **ICL** do not differ much from each other. To investigate the effect of increasing the storage in L-BFGS methods compared with the F-BFGS method, we define 'storage-up' as the ratio of

storage locations for F-BFGS : storage locations for L-BFGS ($m = 3, 7$),

We also define 'speed-up' in terms of n_I and n_f to be the ratios of

Total n_I(F-BFGS) : Total n_I(L-BFGS $m = 3, 7$) and Total n_f(F-BFGS)/ Total n_f(L-BFGS $m = 3, 7$).

Thus if the 'speed-up' factors are less than the 'storage-up' factors, the L-BFGS methods do not gain much from additional storage, whereas a larger number means a substantial improvement. These effects are measured in Figures 1 and 2.

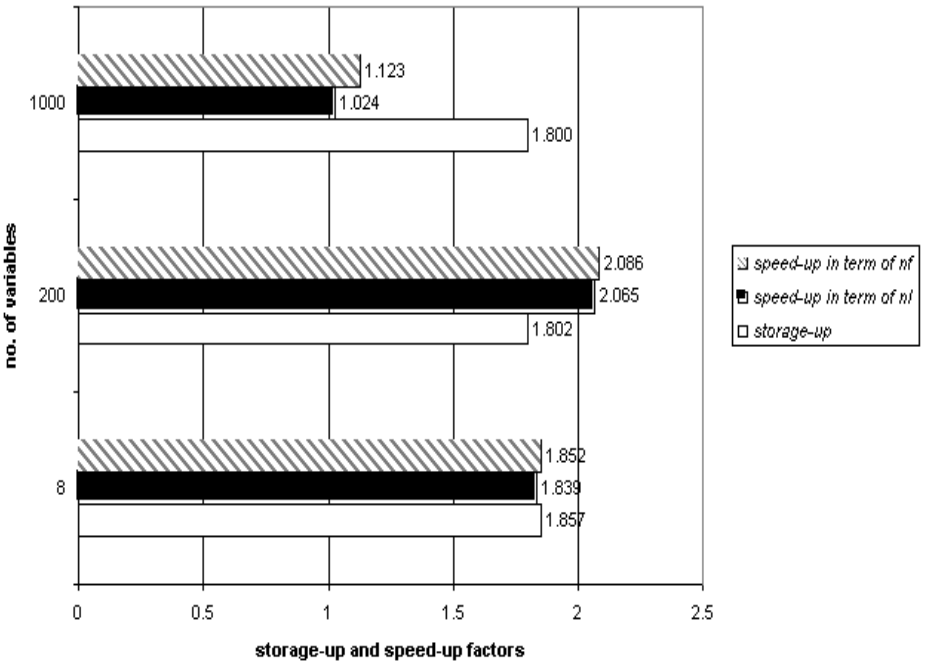

Figure 1: 'Storage-up' vs 'Speed-up' with F-BFGS and L-BFGS($m = 3$)

From Figures 1 and 2 we see that except for $m = 3$, $n = 200$ neither of the 'speed-up' factors are greater than the 'storage-up'. This means that although there is improvement when we switch from F-BFGS method to L-BFGS methods, the gain is not dramatic. Therefore we can conclude that the F-BFGS method is efficient if the resource in storage is low.

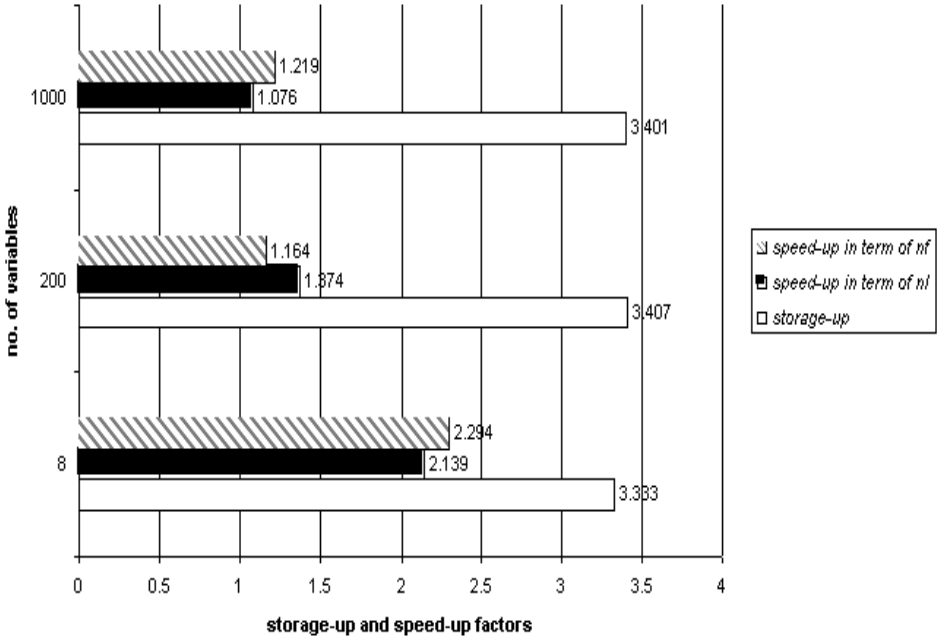

Figure 2: 'Storage-up' vs 'Speed-up' with F-BFGS and L-BFGS($m = 7$)

4 Comparison with Conjugate Gradient Methods

We compare the F-BFGS method with some of the well-known conjugate gradient methods. We tested three methods: F-BFGS method; the conjugate gradient methods (CG) using the Fletcher-Reeve and Polak-Ribiére formula, restarting every n steps, and with $\beta' = 10^{-4}$ and $\beta = 0.9$ in (3) and (12). For the CG methods, a modified Moré's line search subroutine CVSMOD is used. The modification is suggested by Liu et. al. (1998) in subroutine CGFAM. Both CG codes (as described in CGFAM) used in the experiments require $5n$ storage spaces, which are 2 storages unit less than the F-BFGS method.

Table 4 compares the results of F-BFGS and the two conjugate gradient methods.

Table 4. F-BFGS vs CG

	F-BFGS			CG Fletcher-Reeve			CG Polak-Riebere		
	n_I	n_f	runtime	n_I	n_f	runtime	n_I	n_f	runtime
Penalty I									
$n = 8$	39	51	0.16	118	308	0.44	64	239	0.27
$n = 200$	57	74	0.21	226	489	0.88	47	178	0.22
$n = 1000$	68	87	0.36	626	1307	3.07	38	159	0.27
Trigonometric									
$n = 8$	29	41	0.10	26	56	0.16	27	59	0.16
$n = 200$	48	58	0.21	187	379	0.93	33	75	0.21
$n = 1000$	50	61	0.43	231	467	2.19	40	92	0.44
RosenBrook									
$n = 8$	42	59	0.16	43	118	0.21	26	77	0.12
$n = 200$	39	61	0.16	77	181	0.32	18	62	0.10
$n = 1000$	36	61	0.21	79	185	0.43	26	73	0.16
Powell									
$n = 8$	132	167	0.43	59	137	0.27	47	117	0.21
$n = 200$	125	158	0.43	206	421	0.82	218	478	0.87
$n = 1000$	142	184	0.76	1002	2010	4.83	78	183	0.38
Wood									
$n = 8$	221	286	0.76	33	82	0.16	106	282	0.43
$n = 200$	195	259	0.71	74	159	0.32	133	338	0.54
$n = 1000$	258	351	1.48	1019	2060	5.16	40	96	0.27
Beale									
$n = 8$	18	21	0.10	21	52	0.10	10	27	0.10
$n = 200$	20	27	0.16	55	117	0.22	11	30	0.10
$n = 1000$	16	19	0.16	41	91	0.27	11	30	0.10

The performance in terms of function calls is as expected: the F-BFGS with Moré's line search is generally the best. Table 5 compares the methods in term of ICL.

In term of ICL, a clear superiority for F-BFGS method over the two CG methods.

Table 5. F-BFGS vs CG

	Index of Computational Labour (ICL)		
	F-BFGS	CG-FR	CG-PR
Penalty I			
$n = 8$	11.77	23.49	33.61
$n = 200$	206.95	434.91	761.23
$n = 1000$	1280.69	2089.95	4188.40
Trigonometric			
$n = 8$	12.72	19.39	19.67
$n = 200$	242.88	407.37	456.82
$n = 1000$	1221.22	2023.67	2302.3
RosenBook			
$n = 8$	12.64	24.70	26.65
$n = 200$	314.39	472.48	692.33
$n = 1000$	1696.14	2344.11	2810.50
Powell			
$n = 8$	11.39	20.90	22.51
$n = 200$	254.06	410.78	440.72
$n = 1000$	1297.07	2007.99	2348.50
Beale			
$n = 8$	11.65	22.36	23.94
$n = 200$	266.97	431.88	510.81
$n = 1000$	1361.83	2023.61	2402.40
Wood			
$n = 8$	10.50	22.29	24.30
$n = 200$	271.35	427.58	548.18
$n = 1000$	1188.69	2221.73	2730.00

5 Solving Extremely Large Problems

The largest number of variables for the problems tested so far is 1000 variables. In this section, we describe the performance of all these methods on problems with 10^6 variables. The machine used has the same configuration with that used in performing previous experiments. Double precision arithmetic in this machine has a unit round off of approximately 10^{-16}. The results are reported in Table 6.

Table 6. Results of F-BFGS ($n = 10^6$)

Test Problems	n_I	n_f	runtime
Penalty I	33	40	213.79
Trigonometric	–	–	–
Rosenbook	42	68	184.45
Powell	233	298	824.78
Beale	25	40	106.48
Wood	196	252	692.64

F-BFGS method is the only successful method in solving these extremely large scale problems. Although F-BFGS only solved 5 over the total 6 test problems, it is enough to prove the efficiency of the method when applied to extremely large problems. L-BFGS cannot solve these extremely large problems due to the insufficiency of memory to start the runs and they require high storages that are not suitable for extremely large problems. All CG methods fail to converge for all the test problems.

6 Conclusions

In our numerical experiments, we found that L-BFGS methods require more storage and is not suitable when the problem has very large number of variables or the resource in storage is very low. The CG methods require only low storage but more function and gradient called. This is very unsuitable if function and gradient evaluations are expensive or an inaccurate line search is used. Moreover, there is no guarantee of convergence for CG methods globally when applied to large scale problems. The F-BFGS method is appealing for several reasons: it is very simple to implement, it requires moderate number of function and gradient called and low storage requirement. In term of computational labor, F-BFGS is competitive with L-BFGS and is clearly superior over the CG methods. Lastly but not less is that the F-BFGS method successfully solves very large scale problems with 106 variables while other methods fail.

References

[1] Dennis, J.E. Jr. and Schnabel, R.B., *Numerical methods for unconstrained optimization and nonlinear equations*, Prentice-Hill Inc., New Jersey, 1983.

[2] Fletcher, R. and Reeves, C.M., *Function minimization by conjugate gradients*, Computing Journal 7 (1964) 149-154.

[3] Gill, P.E. and Murray, W., *Conjugate-gradient methods for large scale nonlinear optimization*, Technical report SOL 79-15, Department of Operation Research, Stanford University, Stanford, 1979.

[4] Malik, Hj. A.H., Mansor, M. and Leong, W.J., *Matrices-storage free BFGS method in large scale nonlinear unconstrained optimization*, Proceeding of International Conference on Mathematics and its Application in the New Millennium, UPM (2000), 430-436.

[5] Liu, G., Nocedal, J. and Wartz, R, Subroutine CGFAM, Argonne National Lab., Argonne, Oct. 1998.

[6] Moré, J.J., Garbow, B.S. and Hillstrom, K.E., *Testing unconstrained optimization software*, ACM Transaction on Mathematical Software 7 (1981) 17-41.

[7] Moré, J.J. and Thuente, D., *Line search algorithms with guaranteed sufficient decrease*, ACM Transaction on Mathematical Software 20 (1994) 286-307.

[8] Nocedal, J., *Updating quasi-Newton matrices with limited storage*, Mathematics of Computation 35 (1980) 773-782.

[9] Polak, E. and Ribiére, G., *Note sur la convergence de méthods de directions conjuguées*, Revue Francaise d'Informatique et de Recherche Opérationnelle 16 (1969) 35-43.

[10] Shanno, D.F. and Phua, K.H., *Matrix conditioning and nonlinear optimization*, Mathematical Programming 14 (1978), 149-160.

[11] Wolfe, M.A., *Numerical methods for unconstrained optimization*, Van Nostrand Reinhold Company, 1978.