A class of Methods Combining L-BFGS and Truncated Newton*

Lennart Frimannslund† Trond Steihaug‡

March 31, 2006

Abstract

We present a class of methods which is a combination of the limited memory BFGS method and the truncated Newton method. Each member of the class is defined by the (possibly dynamic) number of vector pairs of the L-BFGS method and the forcing sequence of the truncated Newton method. We exemplify with a scheme which makes the hybrid method perform like the L-BFGS method far from the solution, and like the truncated Newton method close to the solution. The cost of a method in the class of combined methods is compared with its parent methods on different functions, for different cost schemes, namely the cost of finite difference derivatives versus AD derivatives, and whether or not we can exploit sparsity. Numerical results indicate that the example hybrid method usually performs well if one of its parent methods performs well, to a large extent independent of the cost of derivatives and available sparsity information.

Keywords: Limited memory BFGS, truncated Newton, automatic differentiation, sparsity.

1 Introduction

We consider the unconstrained optimisation problem

$$\min_{x \in \mathbb{R}^n} f(x), \tag{1.1}$$

*This work was supported by the Norwegian Research Council (NFR).

†Department of Informatics, University of Bergen, Box 7800, N-5020 Bergen, Norway. E-mail: lennart.frimannslund@ii.uib.no

‡Department of Informatics, University of Bergen. E-mail: trond.steihaug@ii.uib.no

where $f : \mathbb{R}^n \to \mathbb{R}$ is two times continuously differentiable. If the Hessian is available one may use Newton's method for solving (1.1). At the heart of a Newton iteration is the solution of the step equation,

$$\nabla^2 f(x^k)\ p^k = -\nabla f(x^k). \tag{1.2}$$

If only gradients are available, one can use quasi-Newton methods, such as the BFGS method, which typically maintains an approximation H to the inverse of the Hessian, which gives the step equation

$$-H^k \nabla f(x^k) = p^k, \tag{1.3}$$

See e.g. [12] and the references therein. Both Newton's method and quasi-Newton methods require $O(n^2)$ storage. If memory is limited and the product of the Hessian with an arbitrary vector is available, one may use an iterative method to solve (1.2). Solving (1.2) full accuracy at each iteration corresponds to the regular Newton's method. Alternatively, one can solve (1.2) inexactly when far form the solution, and to an increasing degree of accuracy as one approaches the optimal solution. The latter method is called a truncated Newton (TN) method [6, 7]. If only gradients are available and memory is limited, one may use a discrete truncated Newton method (DTN) such as in [19, 7]. DTN uses gradients to approximate the product of the Hessian with an arbitrary vector. An alternative is a limited memory quasi-Newton method such as the limited memory BFGS (L-BFGS) method [17, 11, 8], which maintains a user-defined portion of the information contained in the Hessian approximation of the full BFGS method.

Several attempts have been made to create a method which combines the properties of the (discrete) truncated Newton method and the L-BFGS method. It has been proposed to use the difference pairs accumulated by L-BFGS as a preconditioner for the iterative solution of the step equation [16], using difference pairs from within a conjugate gradient (CG) solution process to provide fresh difference pairs for the L-BFGS method [3, 13], and using inexact or exact Hessian information to create an incomplete or modified Cholesky preconditioner for determining the matrix H_0^k used by the L-BFGS method [10, 20]. This idea stems from proposition 1.7.3 in [1]. In this paper we present a novel class of methods which combines the truncated Newton method and the L-BFGS method. We determine the matrix H_0^k using Hessian information in conjunction with an iterative equation solver such as the conjugate gradient (CG) method. The reason we consider the L-BFGS specifically, is that is has the advantage over other limited memory quasi-Newton methods that the product involving H_0 is isolated from the rest of the computations [2]. Specifically, we will numerically test the performance of a method in the class

which behaves like the L-BFGS method when far from the solution, and like truncated Newton when close to the solution. The reason for this is the same logic that lies behind truncated Newton, that accurate second-order information is more important close to the solution than far from it.

This paper is organised as follows. In section 2 we describe the truncated Newton method, BFGS and L-BFGS. In section 3 we outline the hybrid approach, give numerical results in section 4 and give some concluding remarks in section 5.

2 Truncated Newton and Limited Memory BFGS

A truncated Newton method makes use of an iterative solver to solve (1.2), inexactly. Let $\tilde{p}^k$ be an inexact solution to (1.2). If the relative residual of the step equation, that is,

$$\frac{\|\nabla^2 f(x^k)\tilde{p}^k + \nabla f(x^k)\|}{\|\nabla f(x^k)\|},$$

is forced to conform to a forcing sequence, a sequence $\eta^k,\ k = 1, \ldots$ where

$$\eta^k < 1 \quad \text{for all } k,$$

then the truncated Newton method can be shown to converge [6]. An example of an effective forcing sequence is

$$\frac{\|\nabla^2 f(x^k)\tilde{p}^k + \nabla f(x^k)\|}{\|\nabla f(x^k)\|} \leq \eta^k, \quad \eta^k = \min\left\{1/k, \|\nabla f(x^k)\|\right\}, \quad (2.1)$$

which was introduced in [7]. Most iterative solvers do not require an explicit representation of the coefficient matrix itself, instead only the product of the matrix with an arbitrary vector. Discrete Newton methods take advantage of this by observing that

$$\nabla^2 f(x)v \approx \frac{\nabla f(x + \epsilon v) - \nabla f(x)}{\epsilon}, \quad (2.2)$$

and use this finite difference estimate as the product needed by the iterative equation solver. If one combines these two techniques, one gets a discrete truncated Newton method.

Limited-memory BFGS or L-BFGS [17, 11] is a modification of the BFGS method. BFGS usually works by by maintaining an approximation to the inverse of the Hessian. It performs a rank-two update at

each iteration. Specifically, given the approximation H^k at iteration k,

$$H^k \approx \left(\nabla^2 f(x^k)\right)^{-1},$$

and given

$$y^k = \nabla f(x^{k+1}) - \nabla f(x^k) \quad \text{and} \quad s^k = x^{k+1} - x^k,$$

then H^{k+1} is taken to be

$$H^{k+1} = (I - \rho^k s^k (y^k)^T) H^k (I - \rho^k y^k (s^k)^T) + \rho^k s^k (s^k)^T, \qquad (2.3)$$

where

$$\rho^k = \frac{1}{(y^k)^T s^k}.$$

The initial approximation to the inverse of the Hessian is up to the user. From the update formula (2.3) one can say that BFGS "remembers" the effect of all difference pairs $(y^i, s^i),\ i = 1, \ldots, k$. In L-BFGS the number of difference pairs (y^i, s^i) is a user-defined parameter, and only the most recent, say, m differences are kept. This leads to significant reduction in memory usage when $m \ll n$, since the product of H^k with an arbitrary vector can be computed from a two-loop formula without the need for storing an $n \times n$ matrix, only storing m difference pairs. If $k < m$, then the method uses the k difference pairs it has available.

L-BFGS Update Formula Let μ be a vector of the appropriate length, and μ_i be its ith component. The two-loop procedure for computing $r = H^k v$ is given below:

$q = v$

for $i = k - 1$ **step** -1 **until** $k - m$

  $\mu_i = \rho^i \cdot (s^i)^T q$

  $q = q - \mu^i y^i$

end

$r = H_0^k q$

for $i = k - m$ **step** 1 **until** $k - 1$

  $\beta = \rho^i \cdot r^T y^i$

  $r = r - (\beta - \mu_i) s^i$

end

Note that the choice of matrix H_0^k is, again, not defined but up to the user. One may choose $H_0^k = I$ to avoid an extra matrix-vector product, but a choice which has proved to be successful is to set

$$H_0^k = \frac{(s^{k-1})^T y^{k-1}}{(y^{k-1})^T y^{k-1}} \cdot I, \tag{2.4}$$

which is supported by numerical testing in [11], where several choices for H_0 are tested. In [8] similar numerical tests indicate that a diagonal matrix based of the diagonal of a BFGS-type approximation to the Hessian may perform even better than (2.4).

The reduced cost of the above formula compared to regular BFGS comes at the cost of convergence rate, which is linear for the L-BFGS method. L-BFGS can also require a very large number of iterations on ill-conditioned problems.

Globalization For Newton-based methods to be globally convergent, that is, converge to the solution from an arbitrary starting point, one approach is to use line searches. We will use line searches satisfying the strict Wolfe conditions (see e.g. [18], chapter 3.1), that is, given the solution to (1.2), at iteration k then the next iterate, x^{k+1} is taken to be

$$x^{k+1} = x^k + \alpha^k p^k,$$

for α^k satisfying

$$f(x + \alpha^k p^k) \leq f(x^k) + c_1 \alpha^k \nabla f(x^k)^T p^k, \tag{2.5}$$

and

$$|\nabla f(x^k + \alpha^k p^k)^T p^k| \leq c_2 |\nabla f(x^k)^T p^k|, \tag{2.6}$$

with $c_1 = 10^{-4}$ and $c_2 = 0.9$ [11]. In our experiments we will use a line search procedure based on that of [15].

3 Hybrid Approach

Consider the assignment

$$r = H_0^k q, \tag{3.1}$$

between the two for-loops in the L-BFGS formula for $r = H^k v$. If H_0^k were the inverse of the Hessian at x^k, then (3.1) would correspond to

$$\nabla^2 f(x^k)\ r = q. \tag{3.2}$$

Let $\tilde{r}$ be an inexact solution to (3.2), with relative residual

$$\frac{\|q - \nabla^2 f(x^k)\tilde{r}\|}{\|q\|}.$$

We wish to combine TN with L-BFGS by replacing r from (3.1) in the two-loop formula with $\tilde{r}$, obtained by applying an iterative equation solver to (3.2). As a starting point for the iterative method we will use the formula normally used by L-BFGS, that is

$$\frac{(s^{k-1})^T y^{k-1}}{(y^{k-1})^T y^{k-1}} \cdot q. \tag{3.3}$$

This gives the following algorithm:

Given $x^0,\ k = 0$.
while $\|\nabla f(x^k)\| >$ tolerance,

$k \leftarrow k + 1$.

$v = -\nabla f(x^k)$.

Perform first L-BFGS for-loop, resulting in vector q.

Solve $\nabla^2 f(x^k) r = q$ to some tolerance with (3.3) as the initial guess, resulting in vector $\tilde{r}$.

Set $r = \tilde{r}$.

Perform second L-BFGS loop, resulting in search direction p^k.

Find α^k satisfying (2.5) and (2.6).

Set $x^{k+1} \leftarrow x^k + \alpha^k p^k$.

Optionally adjust m.

Update difference pairs $(s^i, y^i),\ i = \max\{1, k-m\}, \ldots, k$.

end

If $m = 0$ the code and (3.2) is solved to tolerance (2.1), then the code reduces to truncated Newton with (2.1) as forcing sequence. If no iterations are performed on (3.2) and the initial solution (3.3) is returned, then the code becomes L-BFGS. In our numerical experiments we for the hybrid method use the rule that (3.2) should be solved to tolerance

$$\frac{\|q - \nabla^2 f(x^k)\tilde{r}\|}{\|q\|} \leq \tau \tag{3.4}$$

where k is the (outer) iteration number, and τ is given by the rule:

- If $\|q\| \geq 1$, $\tau = 1$.
- If $\|q\| < 1$, $\tau = \max\left\{1/k, \|q\|\right\}$.

Note that this particular rule should not necessarily be used for $m = 0$, were we recommend (2.1) instead. In our experiments we use (2.1) for the truncated Newton method. See [6].

In our experiments we solve (3.2) with the conjugate gradient method. We do not require in general that the number of difference pairs m is constant, but in our experiments we use $m = 3$.

4 Numerical Testing

We wish to test the theoretical cost of the chosen hybrid method compared to the theoretical cost of the corresponding L-BFGS and truncated Newton methods. We calculate the following costs:

- L-BFGS: One function evaluation and one gradient evaluation per iteration.
- Truncated Newton: One function evaluation, one gradient evaluation, and a variable amount of Hessian-vector products per iteration.
- Hybrid method: One function evaluation, one gradient evaluation, and a variable amount of Hessian-vector products per iteration.

When it comes to line search, in our initial experiments the value $\alpha^k = 1$ was usually accepted. If $\alpha^k = 1$ is the first test value for α^k [11], then one does not need interpolation or similar procedures which incur extra cost most of the time. If $\alpha^k = 1$ is not accepted, then the cost of a line search may vary to a very large extent depending on the implementation. In our experiments we add 1/10 times the cost of one gradient and one function evaluation to the cost of an iteration, which corresponds to one extra function and gradient evaluation by the line search once every ten iterations. When it comes to the cost of gradients and Hessian-vector products, we test four different situations. These are whether or not AD is available, and whether or not the sparsity structure of the Hessian can be exploited. If the Hessian is sparse and the sparsity structure is known, then the Hessian can be obtained cheaply from a compressed Hessian matrix with techniques like that of Curtis, Powell and Reid (CPR) [4]. If CPR techniques are available we take the view that if the number of Hessian-vector products needed in the iterative method is larger than ρ, the number of products needed to determine $\nabla^2 f$ from a compressed Hessian matrix, then the cost is only that of ρ such products. This may or may not be a realistic view, but this depends on the relative cost of derivative computations to the operations used to form Hessian vector products

Case	AD	Compr. Hess.	$C(\nabla f)$	$C(\nabla^2 f \cdot v)$
1	No	Yes	$nC(f)$	$(n+1)C(f)$
2	No	No	$nC(f)$	$(n+1)C(f)$
3	Yes	Yes	$5C(f)$	$12C(f)$
4	Yes	No	$5C(f)$	$12C(f)$

Table 4.1: The four scenarios plotted for each test function.

from a compressed Hessian. We take the view that when AD is available, the gradient can be computed at five times the cost of a function evaluation, and that a Hessian-vector product costs 12 function evaluations (see e.g. [18], chapter 7.2). Actual costs in time and memory are discussed in [14], and these numbers are in accordance with the numbers used here. Define $C(f)$ as the cost of one function evaluation. As mentioned, for each problem we test four situations. These are listed in Table 4.1. The first column in the table lists the number for each case. If we for instance look at case 1, then we see that AD is not available, but we can determine Hessian-vector products from a compressed Hessian. Since AD is not available, we calculate the cost of gradients and Hessian vector products as they would be if computed with finite difference formulas. The cost of a gradient is then usually $(n + 1)$ times that of a function evaluation, but since we (in the context of an iterative optimisation method) already have the function value, we set the cost to be n times that of a function value. Similarly for Hessian vector products, this usually requires two gradients, but since we already have one of the necessary gradients in the optimisation method, we write only the cost of one gradient. This cost scheme covers relative costs of gradients to Hessian vector products in the discrete truncated Newton method, where a Hessian vector product costs the same as an (extra) gradient evaluation. Similarly, it covers the situation where an analytically available gradient costs the same as an analytically available Hessian-vector product if we ignore the relative cost of derivatives to the cost of function values.

We test the three methods on the three problems of [5], as well as six problems from the CUTEr collection [9], with convergence criterion

$$\|\nabla f(x)\| \leq 10^{-4}.$$

For each problem we list four figures, with equivalent cost in function evaluations along the x-axis, and the norm of the gradient along the y-axis. For each problem we have AD-derivatives or hand-coded

derivatives, so the costs in the figures are estimates. The four figures correspond to the four cases of Table 4.1, with case 1 and 2 in the first row, and 3 and 4 in the second. The dash-dotted, sometimes oscillating curve corresponds to L-BFGS, the solid line with stars (one star for each outer iteration) corresponds to truncated Newton, and the dashed line corresponds to the hybrid approach. On the HILBERTB function (Figure 4.7) the L-BFGS and hybrid curves are indistinguishable.

On the first three functions, when CPR techniques are available the curves corresponding to the truncated Newton method take sharp dips. This sometimes happens for the hybrid method as well, notably on problem two, but at a slightly later stage than for TN. When CPR techniques are not available the methods perform quite similarly, with the hybrid method frequently in second place, regardless of which method is the fastest.

On the extended Rosenbrock function the curve of the hybrid method is takes a dip when that of TN does, at a slightly later stage. When CPR techniques are not available, it performs as good as identically to L-BFGS (which performs the best) when AD is not available, and close to it when AD is available (case 4).

On the DIXMAANL problem, the hybrid method performs similarly to TN, which performs the best in three of four cases.

On the chained Rosenbrock problem the hybrid method performs the best in case two, slightly worse than L-BFGS in case 4 and is able to benefit from CPR techniques like it did on the second problem, that is, in a similar fashion as TN but at a slightly later stage.

On the HILBERTB problem, the hybrid method together with L-BFGS performs the best in all cases.

On the penalty I function we get very interesting curves, which are in a sense optimal for a hybrid method. Initially, the hybrid method follows the steepest curve, namely that of L-BFGS, and when L-BFGS stagnates continues at the pace of TN.

A similar, but less obvious picture appears for the penalty II function.

5 Concluding Remarks

We have presented a class of hybrid L-BFGS/TN methods, and tested the performance of one of the members in the class compared with its corresponding parent methods. In some of our tests, the hybrid method followed the best curve of its parent methods closely, in other tests it fell between its two parents.

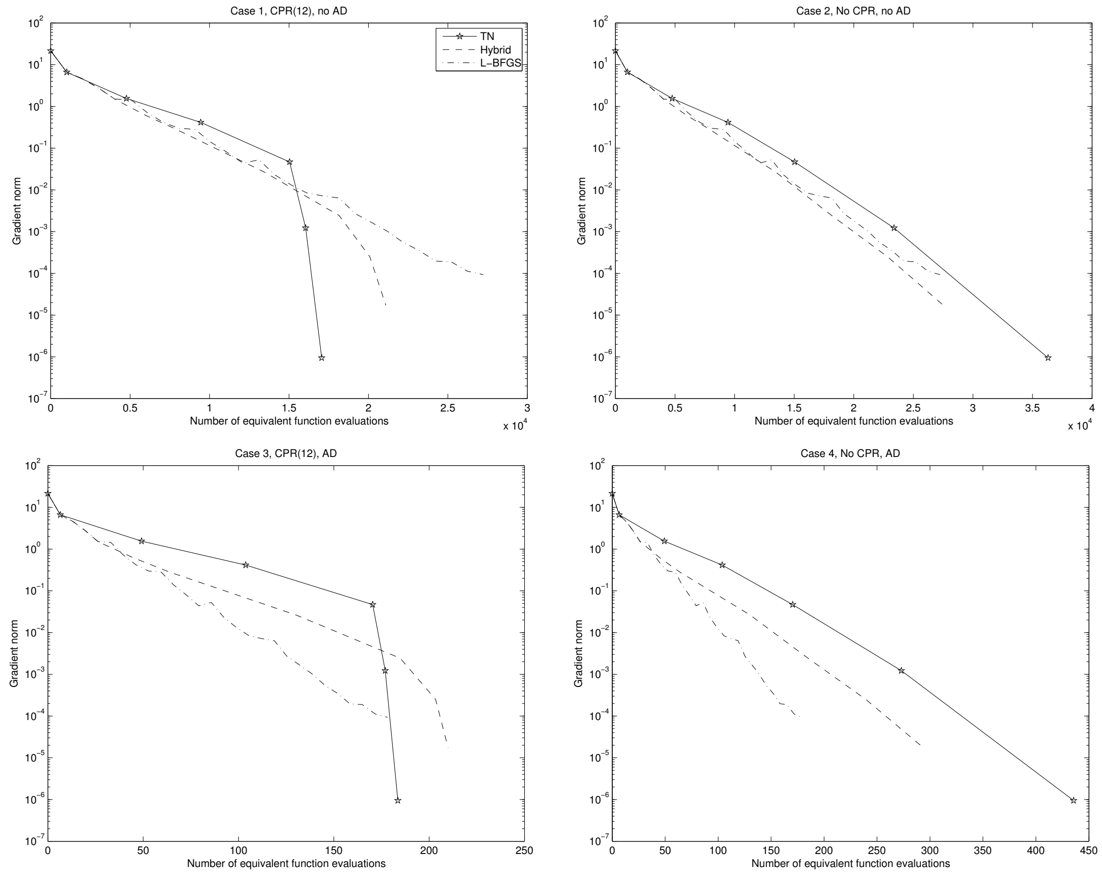


Figure 4.1: Problem 1 from [5], well conditioned, $\rho = 12,\ n = 916$.

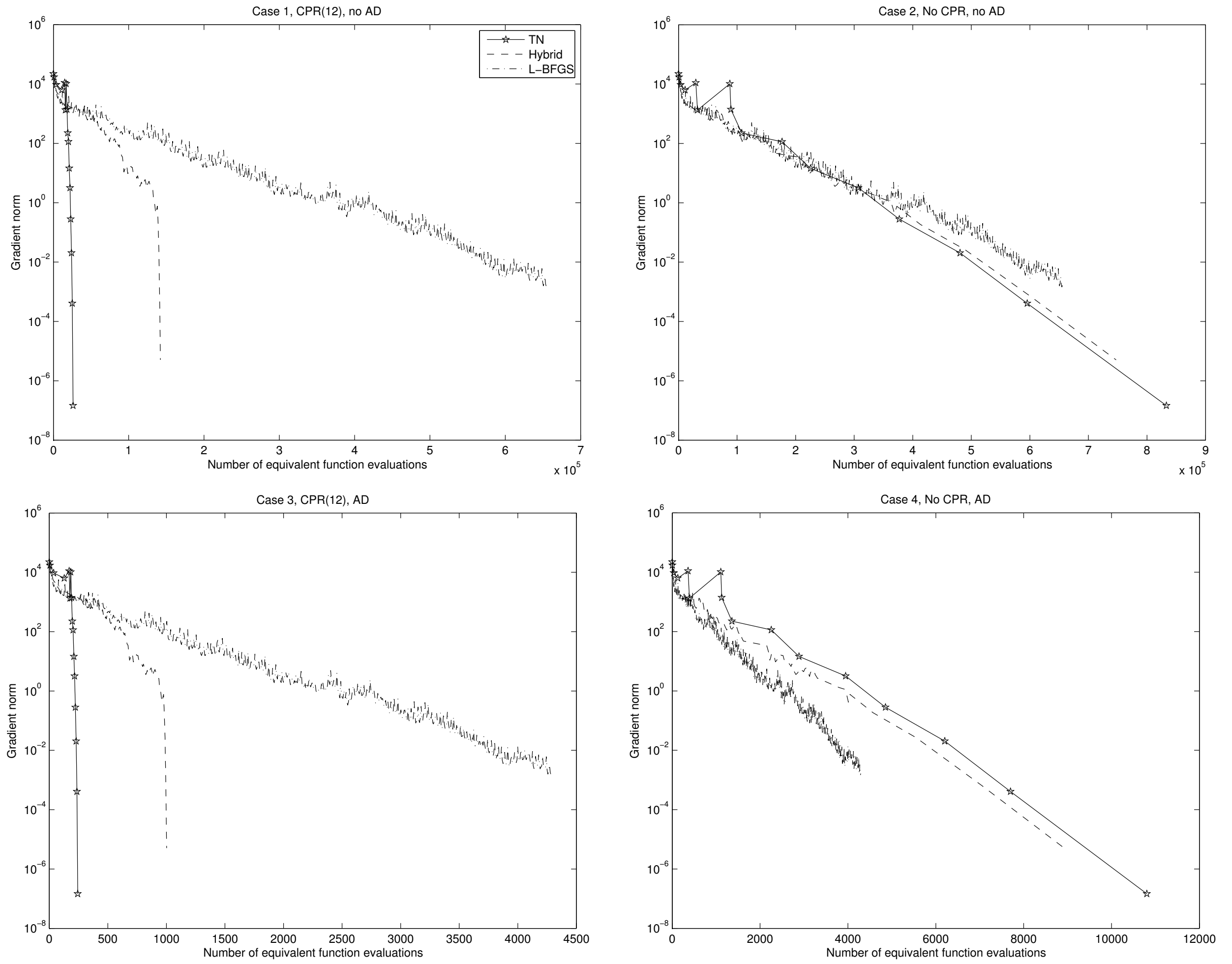


Figure 4.2: Problem 2 from [5], ill-conditioned, $\rho = 12,\ n = 916$.

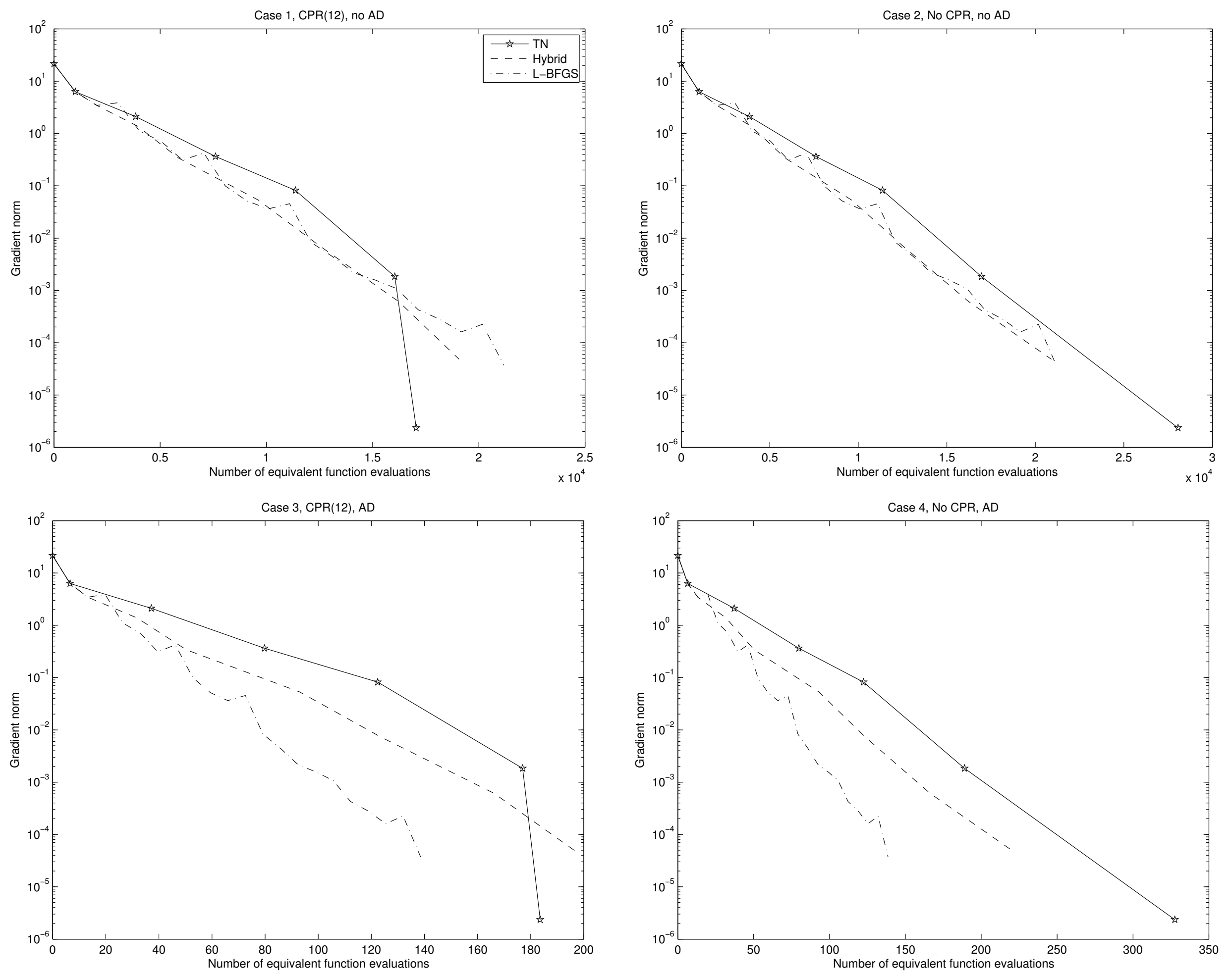


Figure 4.3: Problem 3 from [5], quadratic, $\rho = 12,\ n = 916$.

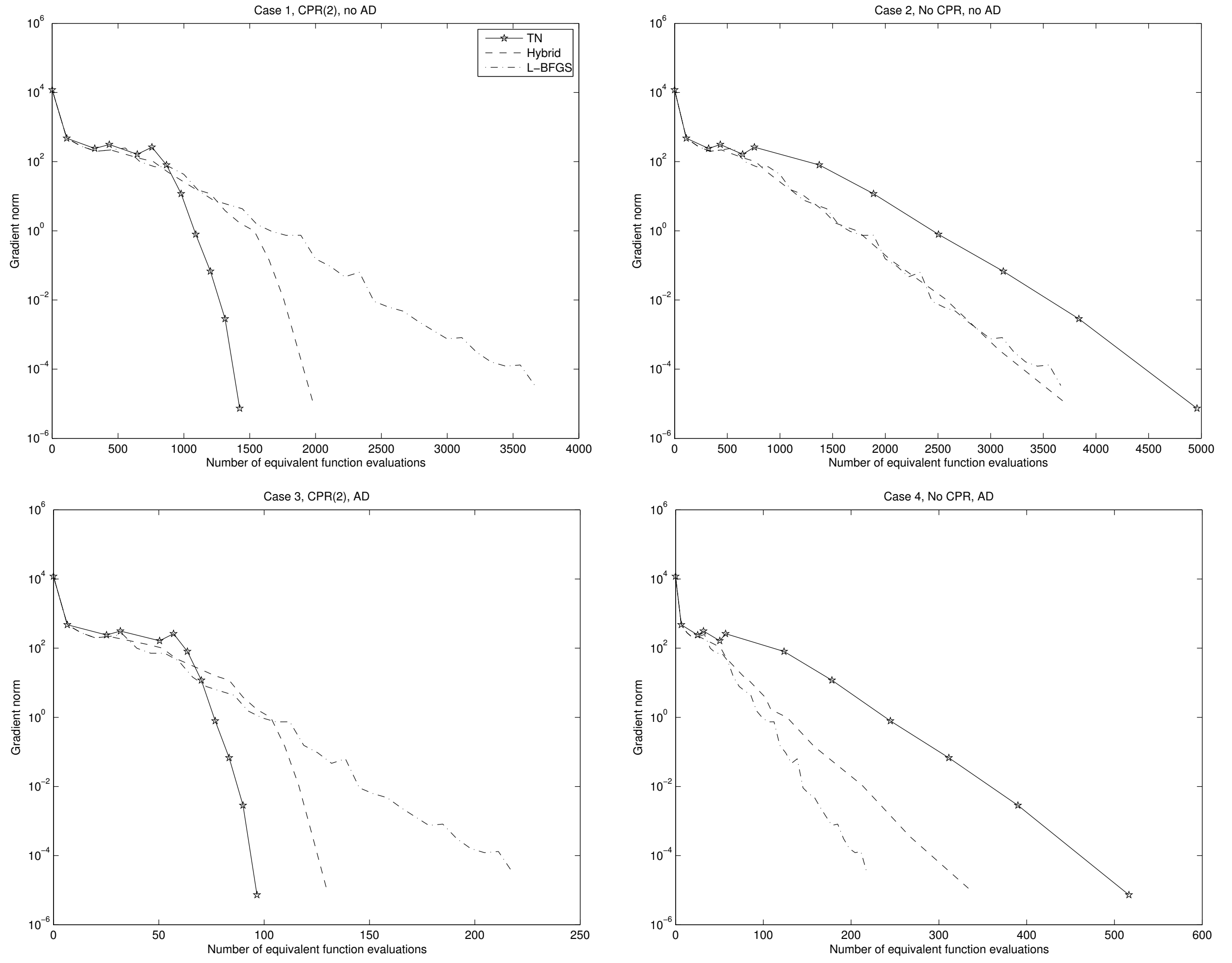


Figure 4.4: Extended Rosenbrock (EXTROSNB), $\rho = 2,\ n = 100$.

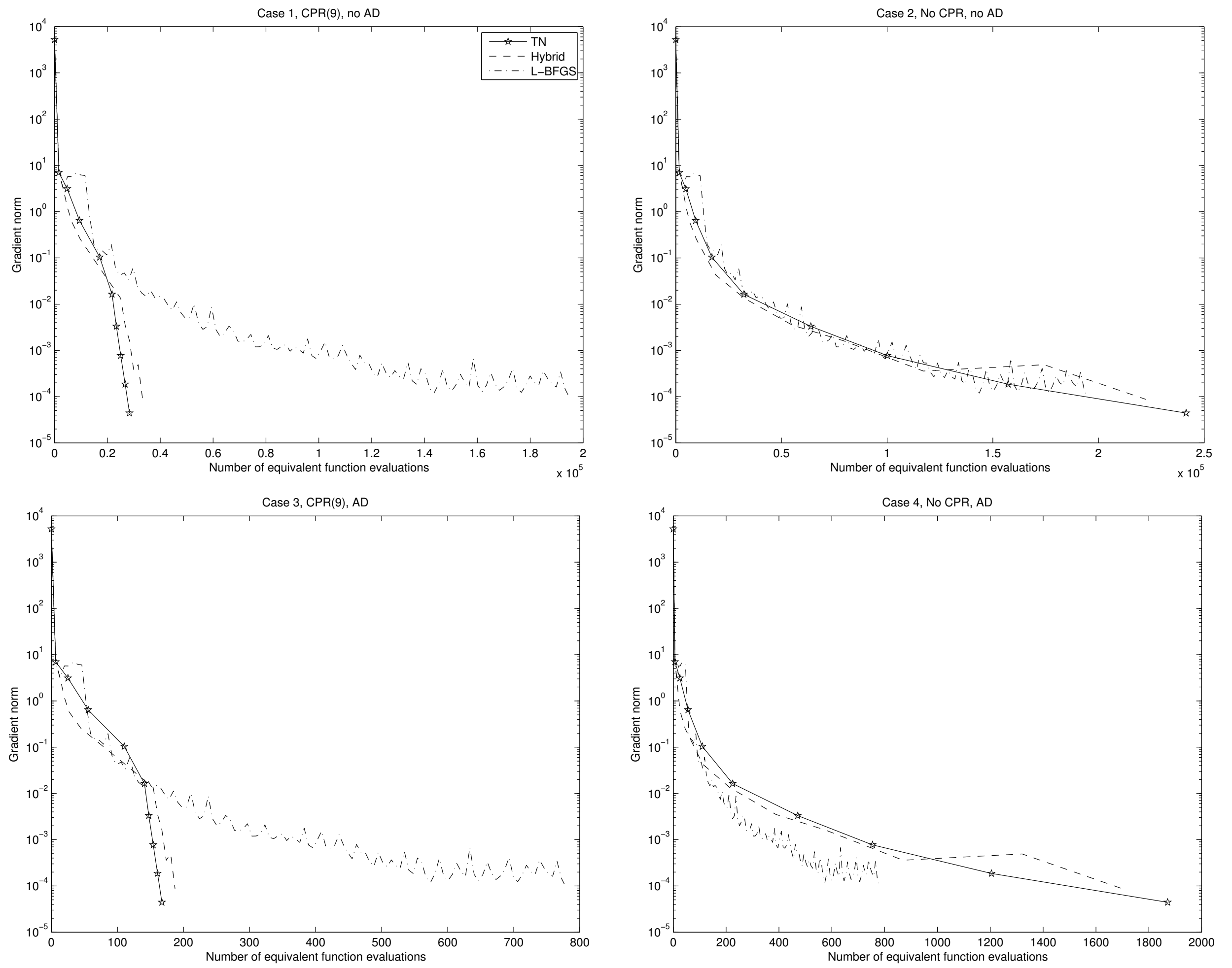


Figure 4.5: DIXMAANL, $\rho = 9,\ n = 1500$.

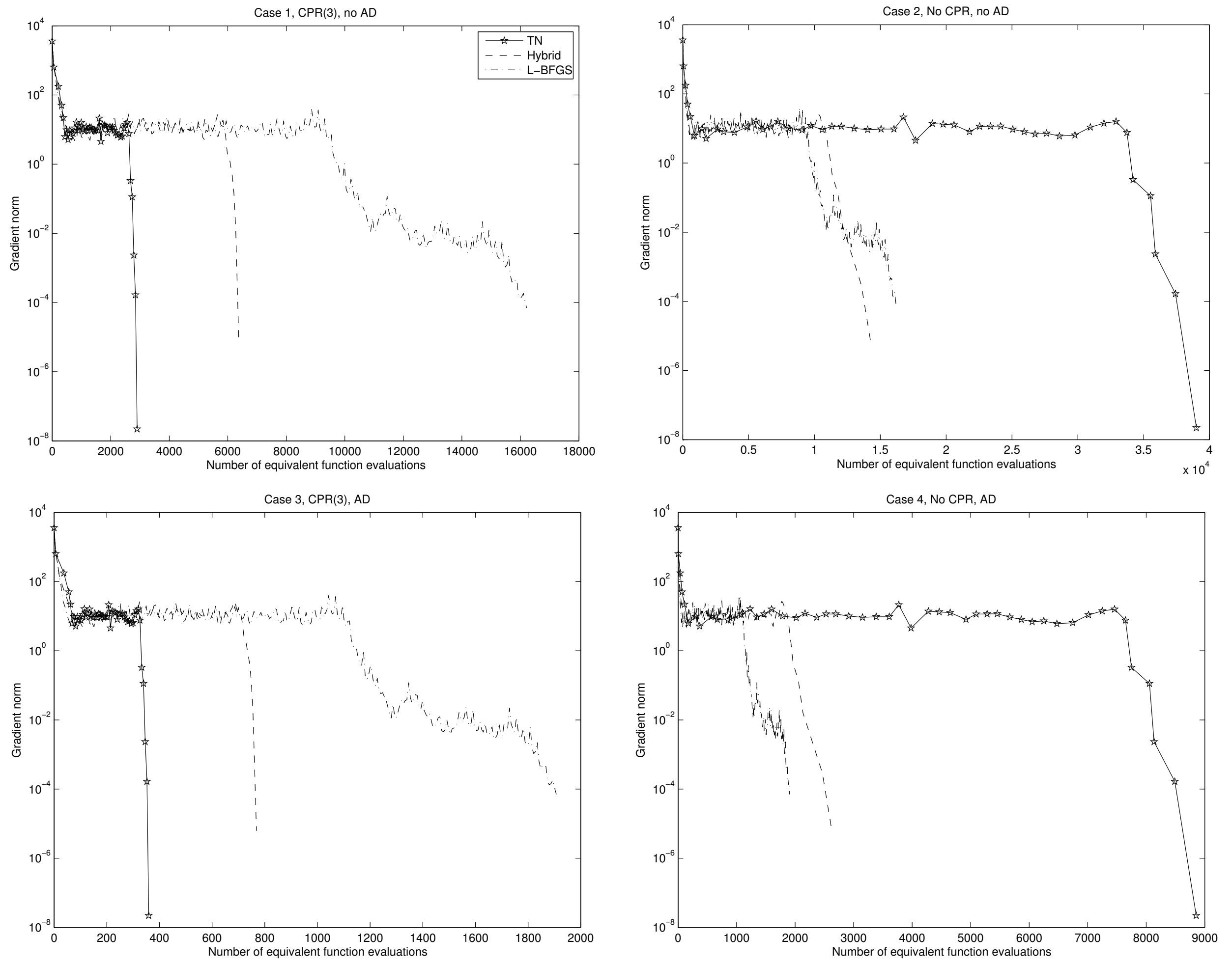
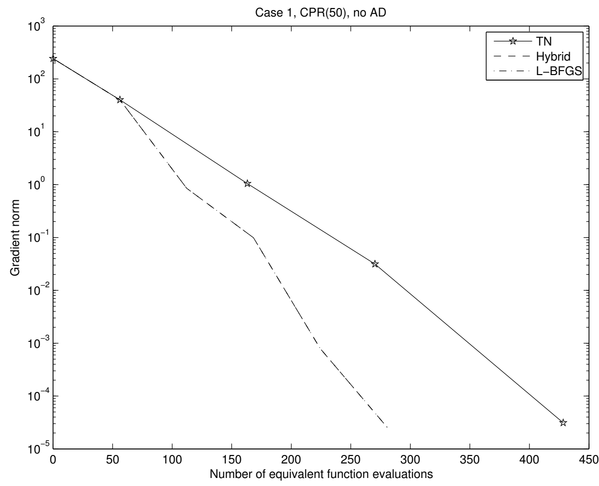
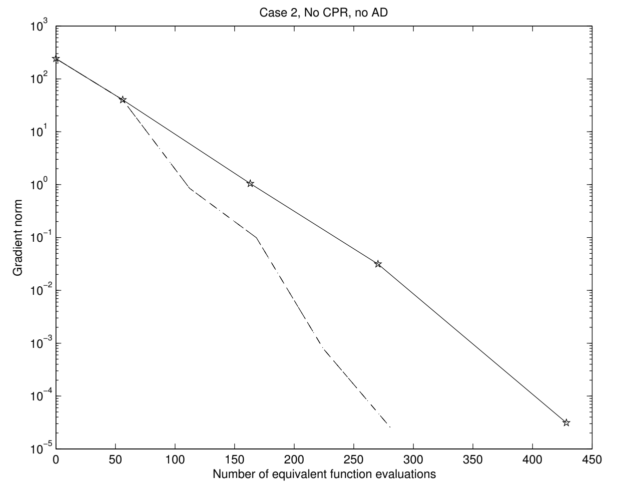
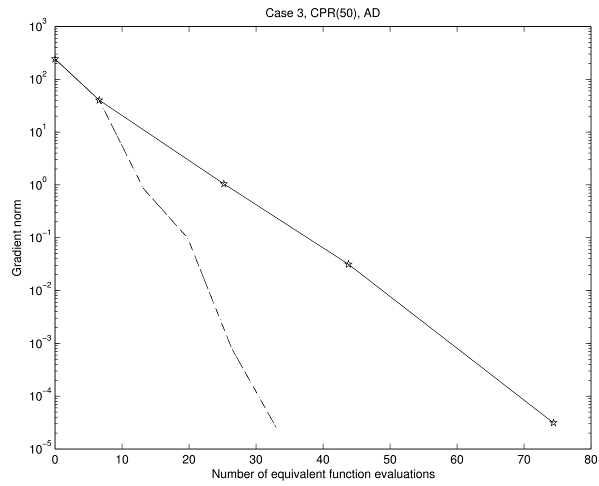


Figure 4.6: Chained Rosenbrock (CHNROSNB), $\rho = 3,\ n = 50$.







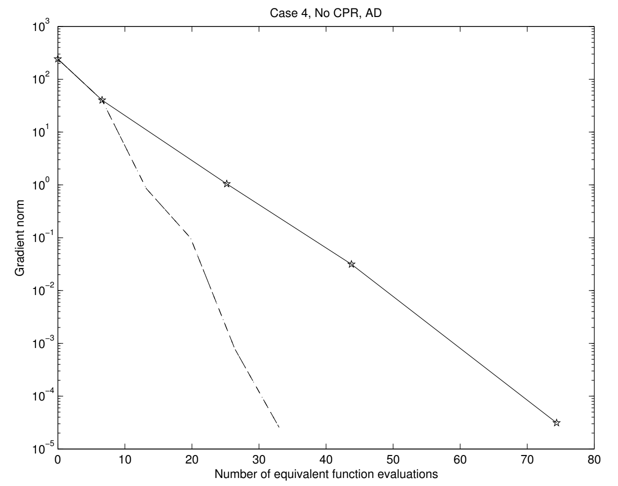


Figure 4.7: HILBERTB, $\rho = 50,\ n = 50$.

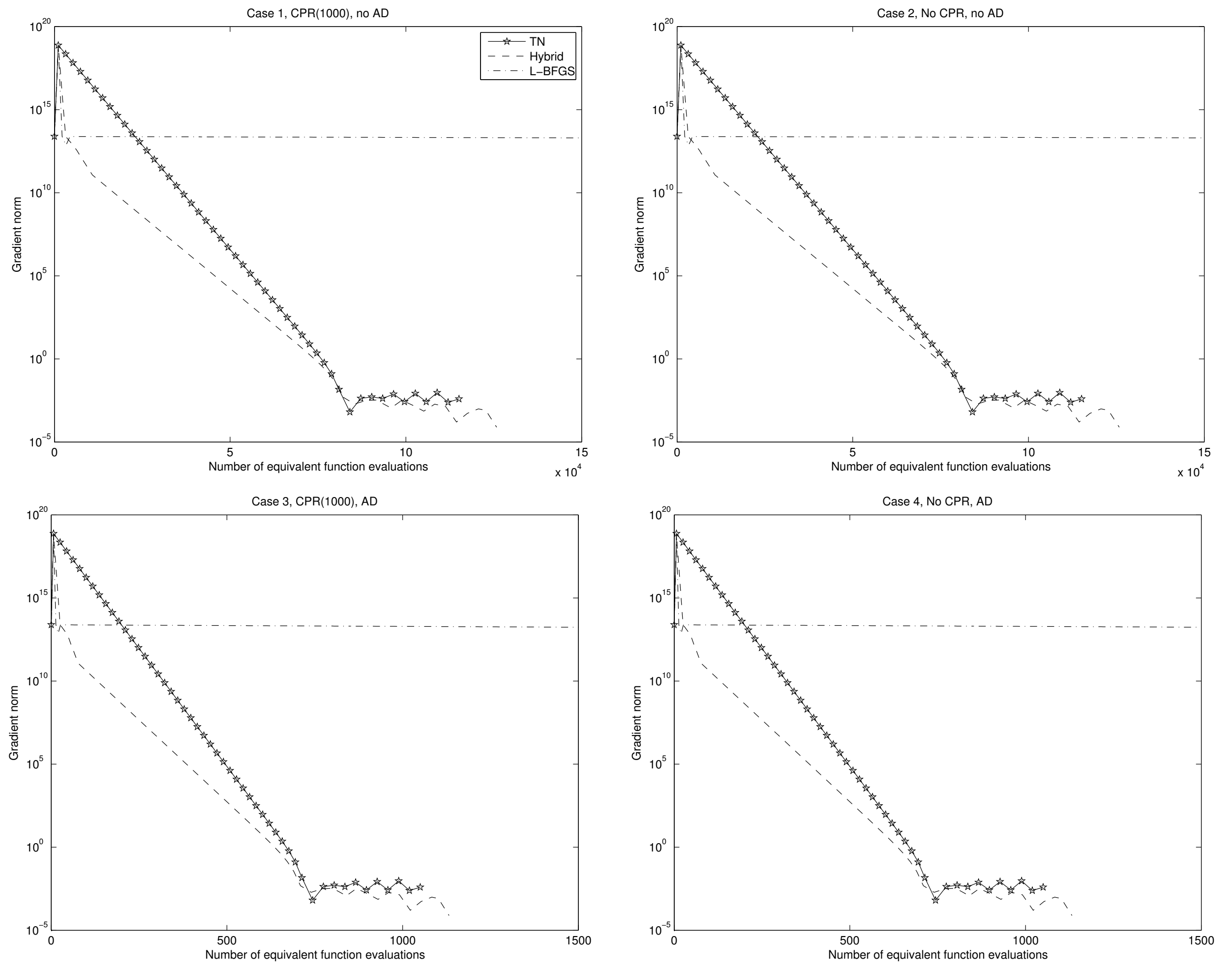


Figure 4.8: PENALTYI, $\rho = 1000,\ n = 1000$.

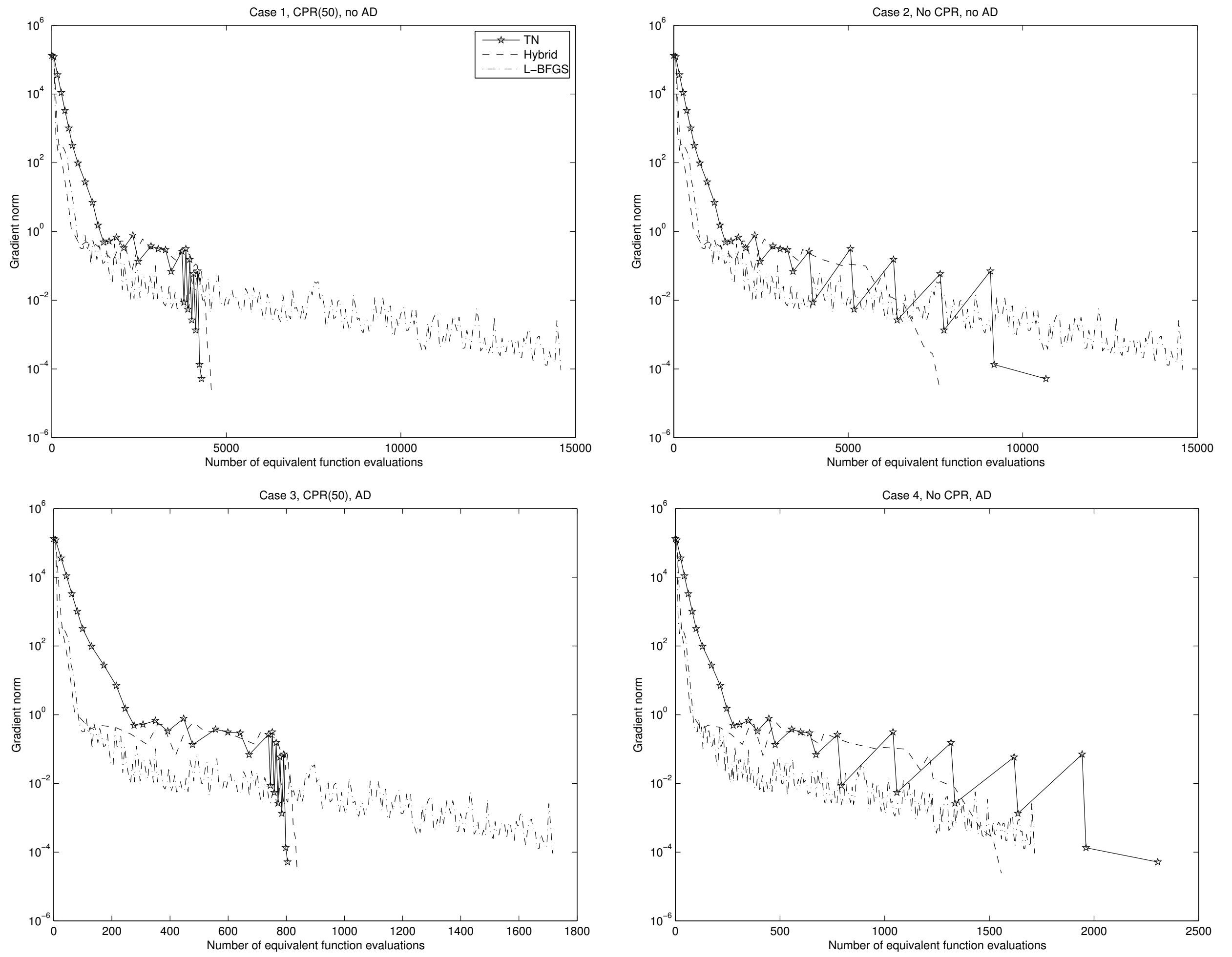


Figure 4.9: PENALTYII, $\rho = 50,\ n = 50$.

The exception to this rule comes when TN is able to exploit CPR techniques effectively at a very early stage by performing many cheap CG iterations, although the tested hybrid method converges very quickly when it starts performing many CG iterations as well.

We feel the our preliminary numerical results are very promising, and that there should exist forcing sequences and possibly dynamic choices of m which should result in effective methods using little memory.

We have not discussed preconditioning of the iterative equation solver (CG in our tests) itself, and this an aspect that should be looked into in the context of our class. Similarly, other choices than CG should also be investigated.

References

[1] D. P. Bertsekas. *Nonlinear Programming.* Athena Scientific, Belmont, MA, 1995. 2nd edition 1999.

[2] Richard H. Byrd, Jorge Nocedal, and Robert B. Schnabel. Representations of quasi-Newton matrices and their use in limited memory methods. *Mathematical Programming*, 63:129–156, 1994.

[3] Richard H. Byrd, Jorge Nocedal, and Ciyou Zhu. Towards a discrete Newton method with memory for large-scale optimization. Technical Report OTC 95/01, Optimization Technology Center, 1996.

[4] A. R. Curtis, M. J. D. Powell, and J. K. Reid. On the estimation of sparse jacobian matrices. *J. Inst Maths Applics*, 13:117–119, 1974.

[5] R. S. Dembo and T. Steihaug. A test problem generator for large-scale unconstrained optimization. *ACM Transactions on Mathematical Software*, 11(2):97–102, 1985.

[6] Ron Dembo, Stanley Eisenstat, and Trond Steihaug. Inexact Newton methods. *SIAM Journal on Numerical Analysis*, 19(2):400–408, 1982.

[7] Ron S. Dembo and Trond Steihaug. Truncated-Newton algorithms for large-scale unconstrained optimization. *Mathematical Programming*, 26:190–212, 1983.

[8] Jean Charles Gilbert and Claude Lemaréchal. Some numerical experiments with variable-storage quasi-Newton algorithms. *Mathematical Programming*, 45:407–435, 1989.

[9] Nicholas I. M. Gould, Dominique Orban, and Philippe L. Toint. CUTEr (and SifDec), a constrained and unconstrained testing environment, revisited. Technical Report RAL–TR–2002–009, Computational Science and Engineering Department, Rutherford Appleton Laboratory, 2002.

[10] Lianju Jiang, Richard H. Byrd, Elisabeth Eskow, and Robert B. Schnabel. A preconditioned l-BFGS algorithm with application to molecular energy minimization. Technical Report CU-CS-982-04, Department of Computer Science, University of Colorado, Boulder, Colorado 80309, 2004.

[11] Dong C. Liu and Jorge Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45:503–528, 1989.

[12] Ladislav Lukšan and Emilio Spedicato. Variable metric methods for unconstrained optimization and nonlinear least squares. *Journal of Computational and Applied Mathematics*, 124:61–95, 2000.

[13] José Luis Morales and Jorge Nocedal. Enriched methods for large-scale unconstrained optimization. *Computational Optimization and Applications*, 21:143–154, 2002.

[14] Jorge J. Moré. Automatic differentiation tools in optimization software. In George Corliss, Christèle Faure, Andreas Griewank, Laurent Hascoët, and Uwe Naumann, editors, *Automatic Differentiation of Algorithms*, 2002.

[15] Jorge J. Moré and David J. Thuente. Line search algorithms with guaranteed sufficient decrease. *ACM Transactions on Mathematical Software*, 20(3):286–307, September 1994.

[16] Stephen G. Nash. Preconditioning of truncated-Newton methods. *SIAM Journal on Scientific and Statistical Computing*, 6(3):599–616, 1985.

[17] Jorge Nocedal. Updating quasi-Newton matrices with limited storage. *Mathematics of Computation*, 35(151):773–782, 1980.

[18] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer–Verlag, 1999. ISBN 0–387–98793–2.

[19] Diane P. O'Leary. A discrete Newton algorithm for minimizing a function of many variables. *Mathematical Programming*, 23(1):20–33, 1982.

[20] Dexuan Xie and Tamar Schlick. Efficient implementation of the truncated Newton method for large scale chemistry applications. *SIAM Journal on Optimization*, 10(1):132–154, 1999.