First Order Method in Nonlinear Optimization

*Report 2 on the course "Numerical Optimization".

1st Chen Yihang

Peking University

1700010780

Abstract

We implements different versions of conjugate gradient method, including FR, PRP, PRP+, CD, DY, HS, FR-PRP, LS methods, and global Borwein-Barzilai method. All algorithms are tested on four different test functions: trigonometric, extended Powell, tridiagonal and matrix square root 1 functions. We presents the reuslts in the tables and plot the error along the optimization trajectory in the figures. In the experiments, we use the default parameters in the references, and we use the default inexact line search parameters unless necessary.

We find that, LS is the most efficient choice, followed by PRP-type method and HS method, while FR and DY are the most inefficient choice. Besides, global BB method requires more iterations especially when the Hessian of the optimal point is ill-conditioned, which might jeopardize the performance significantly.

CONTENTS

I. Optimization method

A. Nonlinear conjugate gradient method

The nonlinear conjugate gradient methods have uniform form:

$$x_{k+1} = x_k + \alpha_k d_k \tag{I.1a}$$

$$d_{k+1} = -g_{k+1} + \beta_k d_k, \quad d_0 = -g_0 \tag{I.1b}$$

where f, g, k denote objective function value, gradient and update direction.

The difference among these methods can be summarized below

$$\beta_k^{\mathrm{FR}} = \frac{g_{k+1}^\top g_{k+1}}{g_k^\top g_k} \tag{I.2a}$$

$$\beta_k^{\mathrm{PRP}} = \frac{g_{k+1}^\top (g_{k+1} - g_k)}{g_k^\top g_k} \tag{I.2b}$$

$$\beta_k^{\mathrm{PRP+}} = \max\left\{ \frac{g_{k+1}^\top (g_{k+1} - g_k)}{g_k^\top g_k}, 0 \right\} \tag{I.2c}$$

$$\beta_k^{\mathrm{CD}} = -\frac{g_{k+1}^\top g_{k+1}}{g_k^\top d_k} \tag{I.2d}$$

$$\beta_k^{\mathrm{DY}} = \frac{g_{k+1}^\top g_{k+1}}{d_k^\top (g_{k+1} - g_k)} \tag{I.2e}$$

$$\beta_k^{\mathrm{HS}} = \frac{g_{k+1}^\top (g_{k+1} - g_k)}{d_k^\top g (g_{k+1} - g_k)} \tag{I.2f}$$

More details of these methods can be found in the Wikipedia page Nonlinear conjugate gradient method.

B. Generalized conjugate gradient algorithms

1) FR-PRP method: We have observed in numerical tests that the FR method with inexact line searches sometimes slows down away from the solution: the steps become very small and this behavior can continue for a very large number of iterations, unless the method is restarted. While PRP method can overcome this issue.

The Hestenes-Stiefel and Polak-Ribire methods appear to perform very similarly in practice, and are to be preferred over the Fletcher-Reeves method. Nevertheless, in a remarkably laborious paper, Powell was able to show that the Polak-Ribire method with exact line searches can cycle infinitely without approaching a solution point. The same result applies to the Hestenes-Stiefel method, since the two methods are identical when $(g_k, d_{k-1}) = 0$, which holds when line searches are exact.

The paper [Gilbert and Nocedal, 1992] suggests the following modification

$$\beta_k^{\mathrm{FR-PRP}} = \max(-\beta_k^{\mathrm{FR}}, \min(\beta_k^{\mathrm{FR}}, \beta_k^{\mathrm{PRP}})) \tag{I.3}$$

An potential further modification would be introducing $c > 1$, and use the following modification

$$\beta_k^{\mathrm{FR-PRP-c}} = \max(-c\beta_k^{\mathrm{FR}}, \min(c\beta_k^{\mathrm{FR}}, \beta_k^{\mathrm{PRP}})) \tag{I.4}$$

2) LS algorithm: Liu ans Storey proposed a new conjugate gradient algorithm which combines conjugate gradient method and Newton's method in [Hu and Storey, 1991].

LIU-STOREY ALGORITHM

- **Step 1**: Set $k = 1, d_1 = -g_1$.
- **Step 2**: Line search. Compute $x_{k+1} = x_k + \alpha_k d_k$; set $k = k + 1$.
- **Step 3**: If $\|g_k\| \leq \varepsilon_1(1 + |f_k|)$ or $|f_k - f_{k-1}| < \varepsilon_2$, then **Stop**; otherwise, go to **Step 4**
- **Step 4**: If $k > M$, go to **Step 8**; otherwise, go to **Step 5**.
- **Step 5**: Compute

$$t_k = d_{k-1}^\top (g(x_k + \delta d_{k-1}) - g_k)/\delta \tag{I.5a}$$

$$u_k = g_k^\top (g(x_k + \delta d_{k-1}) - g_k)/\delta \tag{I.5b}$$

$$v_k = g_k^\top (g(x_k + \gamma d_{k-1}) - g_k)/\gamma \tag{I.5c}$$

where ("eps" means machine epsilon, and equal to 2^{-52} in MATLAB)

$$\delta = \sqrt{\mathrm{eps}}/\|d_{k-1}\|, \quad \gamma = \sqrt{\mathrm{eps}}/\|g_{k-1}\|. \tag{I.6}$$

- **Step 6**: If

$$t_k > 0, \quad v_k > 0, \quad 1 - u_k^2/(t_k v_k) \geq 1/(4r), \quad (v_k g_k^\top g_k)/(t_k/d_{k-1}^\top d_{k-1}) \leq r \tag{I.7}$$

where $r = 1/\sqrt{\mathrm{eps}}$, then go to **Step 7**; otherwise, go to **Step 8**.

- **Step 7**: Let

$$d_k = \frac{(u_k g_k^\top d_{k-1} - t_k g_k^\top g_k) g_k + (u_k g_k^\top g_k - v_k g_k^\top d_{k-1}) d_{k-1}}{w_k} \tag{I.8}$$

- **Step 8**: Set x_k to x_1; go to **Step 1**.

We set $\varepsilon_1 = 10^{-6}, \varepsilon_2 = 2^{-52}, M = 10$.

As a matter of fact, **Step 5** is a Hessian-free version of the following update

$$t_k = s_{k-1}^\top H_k s_{k-1}, \quad v_k = g_k^\top H_k g_k, \quad u_k = g_k^\top H_k s_{k-1} \tag{I.9}$$

C. BB stepsize with nonmonotone line search

We introduce the global BB algorithm from [Raydan, 1997]. . Standard methods for optimization usually generate a sequence of iterates for which a sufficient decrease in the objective function f is enforced at every iteration. In many cases, the global strategy consists of accepting the steplength, in the search direction, if it satisfies the well-known Armijo–Goldstein–Wolfe conditions.

There are some disadvantages to forcing the Armijo–Goldstein–Wolfe conditions when combined with the Barzilai and Borwein gradient method. One of the disadvantages is that forcing decrease at every iteration will destroy some of the local properties of the method. The choice of steplength is related to the eigenvalues of the Hessian at the minimizer and not to the function value. Moreover, since the search direction is always the negative gradient direction, forcing decrease at every iteration will reduce the method to the steepest descent method, which is known for being slow.

Therefore, we will enforce a much weaker condition of the form

$$f(x_{k+1}) \leq \max_{0\leq j\leq M} f(x_{k-j}) + \gamma g_k^\top (x_{k+1} - x_k) \tag{I.10}$$

where M is a nonnegative integer and γ is a small positive number.

GLOBAL BARZILAI AND BORWEIN (GBB) ALGORITHM.

Given x_0, α_0, integer $M \geq 0, \gamma \in (0,1), \delta > 0, 0 < \sigma_1 < \sigma_2 < 1, 0 < \varepsilon < 1$. Set $k = 0$.

- **Step 1**: If $\|g_k\| \leq 10^{-6}(1 + |f_k|)$. **Stop**.
- **Step 2**: If $\alpha_k \leq \varepsilon$ or $\alpha_k \geq 1/\varepsilon$, then set $\alpha_k = \delta$, where δ is chosen in the following way:

$$\delta = \min\left(10^5, \max\left(1, \|g_k\|_\infty^{-1}\right)\right)$$

- **Step 3**: Set $\lambda = 1/\alpha_k$.
- **Step 4**: (nonmonotone line search) $f(x_k - \lambda g_k) \leq \max_{0\leq j\leq \min(k,M)}(f_{k-j}) - \gamma\lambda g_k^\top g_k$, then set $\lambda_k = \lambda$, $x_{k+1} = x_k - \lambda_k g_k$, and go to **Step 6**.
- **Step 5**: Choose $\sigma \in [\sigma_1, \sigma_2]$, set $\lambda = \sigma\lambda$, and go to **Step 4**.
- **Step 6**: Set $\alpha_{k+1} = -(g_k^\top (g_{k+1} - g_k))/(\lambda_k g_k^\top g_k)$, $k = k + 1$, and go to **Step 1**.

The GBB method cannot cycle infinitely between **Step 4** and **Step 5**, since γ decreases to zero.

We set $\gamma = 10^{-4}, \varepsilon = 10^{-10}, \sigma_1 = 0.1, \sigma_2 = 0.5, \alpha_0 = 1$ and $M = 10$.

II. TEST FUNCTION

We test our methods by four functions from [Hu and Storey, 1991]. The implementation of the function will significantly influence the performance of our codes. However, we do not focus too much on improving the computational budget of each evaluation, since it is not our main concern in analyzing the optimization process.

A. *Trigonometric function*

$$f(x) = \sum_{i=1}^{n} \left\{ n + i - \sum_{j=1}^{n} \left[\delta_{ij} \sin(x_j) + (i\delta_{ij} + 1) \cos(x_j) \right] \right\}^2, \quad n = 1, 2, \cdots \tag{II.1a}$$

$$x_0 = (\frac{1}{n}, \frac{1}{n}, \cdots, \frac{1}{n})^\top \tag{II.1b}$$

A efficient implementation of this function needs to precompute the quantity $\sum_{i=1}^{n} \cos(x_i)$.

B. *Extended Powell function*

$$f(x) = \sum_{j=1}^{n/4} [(x_{4j-3} + 10x_{4j-2})^2 + 5(x_{4j-1} - x_{4j})^2 + (x_{4j-2} - 2x_{4j-1})^4 + 10(x_{4j-3} - x_{4j})^4], \quad n = 4, 8, \cdots \tag{II.2a}$$

$$x_0 = (3, -1, 0, 3, 3, -1, 0, 3, \cdots, 3, -1, 0, 3)^\top \tag{II.2b}$$

In testing the Powell function, we are inclined to use the following restart condition

$$|g_k^\top g_{k-1}| \geq 0.2 \|g_k\|^2. \tag{II.3}$$

C. *Tridiagonal function*

$$f(x) = \sum_{i=2}^{n} [i(2x_i - x_{i-1})^2] \tag{II.4a}$$

$$x_0 = (1, 1, 1, \cdots, 1)^\top \tag{II.4b}$$

We explicitly obtain the tridiagonal coefficient matrix in order to accelerate the calculation. Speficically, the function can be written as

$$f(x) = x^\top \begin{pmatrix} 2 & -4 & & & & \\ -4 & 11 & -6 & & & \\ & -6 & 16 & \ddots & & \\ & & \ddots & \ddots & -2(n-1) & \\ & & & -2(n-1) & 5n-4 & 2n \\ & & & & 2n & 4n \end{pmatrix} x \tag{II.5}$$

Besides, we find that initialization in [Hu and Storey, 1991] is inferior another initialization

$$x_0 = \frac{1}{n^2}(1, 1, 1, \cdots, 1)^\top \tag{II.6}$$

We present the results in Table IV.2.

D. Matrix square root 1

This is a matrix-square-root problem. Given $A \in \mathbb{R}^{m\times m}$, we wish to find B such that $B^2 = A$. This can be solved by minimizing the function $\|B^2 - A\|_\infty$. Let $x = \text{vec}(B), a = \text{vec}(A)$, where

$$\text{vec}(X)(m(i-1)+j) = X(i,j), \quad i,j = 1, \cdots, m, \tag{II.7}$$

Then the test function is

$$f(x) = \sum_{i=1}^{n} \left[a(i) - \sum_{j=1}^{m} x(j+lm)x(k+(j-1)m) \right]^2, \quad l = \text{mod}(i-1, m), \quad k = 1 + \text{int}((i-1)/m) \tag{II.8}$$

Let the optimal value be

$$x_\star(i) = \sin(i^2), \quad i = 1, 2, \cdots, m^2. \tag{II.9}$$

and the initial point is

$$x_0(i) = x_\star(i) - 0.8\sin(i^2), \quad i = 1, 2, \cdots, m^2. \tag{II.10}$$

Since $f(B) = \|B^2 - A\|_F^2$, the gradient can be computed as

$$df = \text{trace}\{2(B^\top B^\top B + BB^\top B^\top - A^\top B - BA^\top)dB\}$$

Thus,

$$\frac{df}{dB} = 2(B^\top B^\top B + BB^\top B^\top - A^\top B - BA^\top)^\top \tag{II.11}$$

The computation by taking gradient w.r.t. the matrix B directly is much more efficient than its vectorized version.

III. Experimental settings

A. Stopping criterion

Without other specification, we set the stopping condition to be one of the following conditions are satisfied

1) $\|g_k\|_\infty \leq 10^{-6}(1 + |f_k|)$.
2) $|f_k - f_{k-1}| <$ eps $= 2^{-52}$. (machine epsilon)
3) iter $\geq$ maxIter $= 10^4$.

For large-scale optimization problems, condition 1 is hard to be satisfied. Hence, we might modify this condition to be

$$\|g_k\|_\infty \leq 10^{-5}(1 + |f_k|)$$

instead.

Here we use infinity norm instead of 2-norm of g_k for three reasons:

1) All the numerical experiments on the textbooks adopt the infinity norm of the gradients.
2) The infinity norm will not be affected by the scale of the problem.
3) We mainly compare the performance of each algorithm given the scale n, so the stopping condition, as long as being unified, will not influnce the observations.

B. Line search algorithm

For CG and LS method, we use inexact line search routine based on third order polynomial interpolation, and strong Wolfe condition is applied. If inexact method cannot produce desirable step size. We then choose exact line search algorithm and set the lowest bound of step size to be "eps". If again, this method cannot produce desirable step size, we simply choose the step size of the last iteration. The codes of this part is presented below

```
alpha0 = alpha;
try
    [alpha, info] = bolinesearch(fun, x, d);
```

```
catch
    info(1) = 1;
end
if info(1) % stepsize is not properly selected.
    Rule.opt = [0 10 25 eps];
    [alpha, info] = bolinesearch(fun, x, d, Rule);
    if ~info(1) || abs(alpha) < eps
        alpha = alpha0;
    end
end
```

For LS method, the paper [Hu and Storey, 1991] suggests to use specified initial step size

$$\lambda_0 = \min\left\{2, -2[f(x_k - \text{Est})]/g_k^\top s_k\right\} \tag{III.1}$$

where Est is a lower bound of f, we simply set it to be 0.

We compare the efficiency between this stategy and our default line search settings. Compare Table III.1 with IV.1, IV.2, it seems that the initial step can reduce the number of iteration, but will increase the numeber of function evaluations. Hence it is not adopted in our final experiments.

For GBB method, the backtracking method to find λ_k is based on modified Armijo condition by polynomial interpolation. The main difference is that we replace $f(x_k)$ with $\max_{0\le j\le \min(k,M)}(f_{k-j})$ in all parts of the algorithm. Details of the backtracking method can be found in Algorithm 6.3.5 in [Dennis Jr and Schnabel, 1996].

IV. Numerical results

We test Problem II-A,II-B,II-C when $n = 100, 1000, 10000$, and Problem II-D when $m = 10, 32, 100$. The numerical results are presented from Table IV.1 and IV.2. The trajectory is presented in Figure IV.1, IV.2, IV.3 and IV.4, we have the following observations:

1) Generally, conjugate gradient methods requires less iterations than GBB method. Among all the conjugate gradient method, FR, and DY (sometimes CD) methods are requires significantly more iterations, and mixed CG, i.e. LS method, is the most effective choice in terms of the number of iterations. Other choices, such as PRP+, FR-PRP, are also competitive.
2) When the Hessian of the optimal point is ill-conditioned, such as in Problem II-B and II-C, more iterations are required by the global GBB method than the conjugate gradient method. However,

TABLE III.1
LS, ANOTHER LINE SEARCH SCHEME

Quantity	II-A	II-B	II-C, , init II.4b	II-C, , init II.6	II-D
	1.84e-06	2.92e-08	9.19e-14	1.18e-14	7.19e-12
$\|f_k\|$	2.27e-07	1.72e-07	4.16e-14	7.10e-15	6.41e-10
	2.22e-08	3.98e-07	1.57e-13	6.60e-15	1.18e-05
	9.46e-07	5.78e-07	9.16e-07	8.57e-07	8.40e-07
$\|\|g_k\|\|_\infty$	9.64e-07	4.75e-07	8.03e-07	9.37e-07	9.74e-07
	4.41e-07	2.90e-07	1.80e-06	1.68e-06	5.02e-05
	44	44	86	56	306
iter	53	51	292	133	2761
	58	56	927	117	10000
	1226	1208	2408	1568	8566
feval	1478	1400	8174	3724	77306
	1604	1540	25952	3276	279998
	3	12	0	0	1
restart	3	14	1	0	1
	10	14	2	0	1
	0.023701	0.030237	0.1014361	0.0684341	0.2732823
time	0.207111	0.195339	1.9583623	0.9036704	8.4181423
	0.889574	1.866172	82.5820375	10.6879497	650.00593

since one iteration in BB method demands less computational budget, as in Problem II-C suggests, the BB method is still competitive in CPU time. However, if the Hessian is singular at the solution as in II-B, then conjugate gradient methods clearly out perform GBB (except for FR and DY).

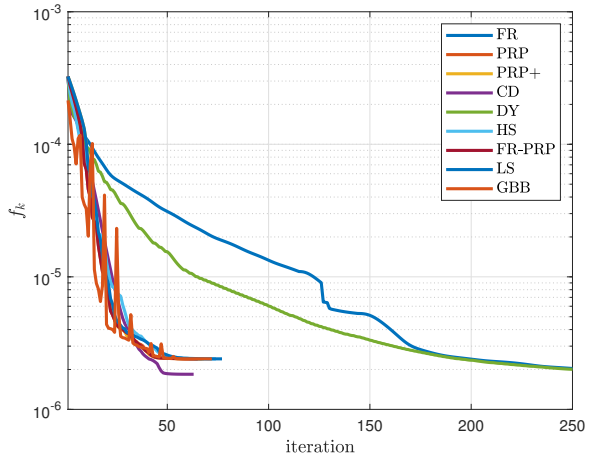
3) Experiments seems to suggests that for ill-conditioned optimization problem, adjusting the direction of each step adaptively (PRP+, LS) performs better than adjusting the step size alone (GBB). Although, adjusting the step size alone requires less computational budget in each iteration than adaptive direction method.

TABLE IV.1
PROBLEM 1/2

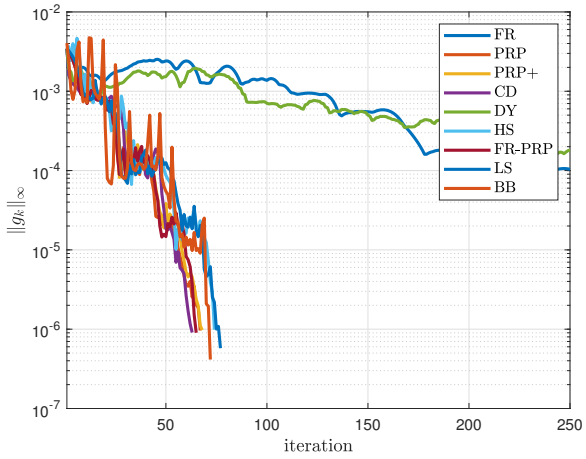
Problem	Quantity	n	FR	PRP	PRP+	CD	DY	HS	FR-PRP	LS	BB
II-A	$\|f_k\|$	100	1.84e-06	2.41e-06	2.41e-06	1.84e-06	1.84e-06	2.41e-06	2.41e-06	2.41e-06	2.41e-06
		1000	1.84e-07	1.95e-07	1.95e-07	1.96e-07	1.97e-07	1.78e-07	2.00e-07	2.00e-07	2.26e-07
		10000	1.87e-08	2.03e-08	2.01e-08	2.02e-08	2.05e-08	1.90e-08	2.03e-08	3.10e-08	2.17e-08
	$\|g_k\|_\infty$	100	9.91e-07	9.62e-07	9.76e-07	9.08e-07	9.43e-07	9.87e-07	9.02e-07	5.76e-07	4.14e-07
		1000	5.73e-07	8.34e-07	8.43e-07	9.61e-07	9.44e-07	6.96e-07	9.82e-07	9.00e-07	4.63e-07
		10000	9.07e-07	7.45e-07	7.57e-07	6.50e-07	9.93e-07	7.86e-07	9.48e-07	9.80e-07	6.56e-07
	iter	100	2034	67	68	63	608	74	65	77	72
		1000	377	74	74	74	788	79	73	102	90
		10000	230	88	89	82	395	86	83	48	103
	feval	100	10300	338	343	318	3168	375	328	466	151
		1000	2186	374	375	373	4201	395	367	605	192
		10000	1310	443	448	414	2304	432	418	274	225
	restart	100	0	0	0	0	0	0	0	39	0
		1000	1	0	0	0	0	0	0	61	0
		10000	2	0	0	0	0	1	0	36	0
	time	100	0.741582	0.0212266	0.0242344	0.0207354	0.265195	0.0226551	0.0211819	0.029502	0.0177944
		1000	0.384897	0.0601441	0.0529417	0.0482616	0.7425712	0.0627332	0.0689437	0.0928285	0.0326308
		10000	1.308382	0.4444714	0.4239693	0.3885348	2.4352026	0.4261139	0.4312413	0.2951051	0.2240482
II-B	$\|f_k\|$	100	4.95e-10	1.69e-09	2.63e-09	2.68e-09	1.25e-09	5.20e-09	1.71e-09	2.89e-08	2.60e-08
		1000	4.93e-09	2.85e-07	1.24e-07	9.90e-09	5.78e-09	1.10e-07	2.03e-08	2.16e-07	1.80e-07
		10000	5.03e-08	4.75e-06	1.55e-06	2.14e-07	9.24e-08	1.95e-06	3.10e-06	1.26e-06	1.76e-06
	$\|g_k\|_\infty$	100	9.94e-07	9.25e-07	1.21e-07	9.51e-07	9.99e-07	2.25e-07	9.75e-07	8.63e-07	5.84e-07
		1000	9.97e-07	7.72e-07	3.09e-07	9.78e-07	9.91e-07	8.10e-07	9.62e-07	7.04e-07	4.11e-07
		10000	9.99e-07	8.36e-07	4.77e-07	9.74e-07	9.69e-07	5.91e-07	8.67e-07	5.28e-07	5.03e-07
	iter	100	4301	222	122	944	1574	155	1385	87	834
		1000	4388	111	156	1870	3340	162	971	115	321
		10000	4168	164	131	1054	1983	304	234	129	696
	feval	100	21867	1143	637	4758	7951	839	6962	552	1978
		1000	22529	592	806	9390	16791	862	4888	748	744
		10000	21013	857	676	5307	9992	1572	1210	842	1611
	restart	100	0	0	0	0	0	11	0	37	0
		1000	0	0	0	0	0	15	0	37	0
		10000	1	1	0	0	0	9	0	39	0
	time	100	1.808024	0.0803862	0.0440682	0.3240168	0.5996952	0.0770168	0.5427954	0.0411649	0.0976749
		1000	5.021645	0.1276586	0.1730055	2.1260722	3.6752114	0.2025432	1.0883448	0.1538129	0.1497006
		10000	34.63926	1.6700172	1.2549402	9.3187068	17.4213357	2.6842781	2.0474927	1.3835165	2.8147962

TABLE IV.2
PROBLEM 3/4

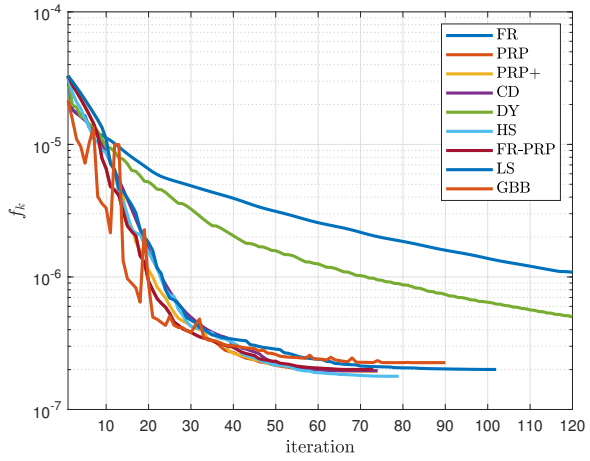
Problem	Quantity	n	FR	PRP	PRP+	CD	DY	HS	FR-PRP	LS	BB
II-C, init II.4b	$\|f_k\|$	100	7.72e-15	6.91e-15	6.91e-15	1.19e-14	9.01e-15	1.07e-14	8.81e-15	8.56e-15	9.09e-14
		1000	4.38e-15	5.29e-15	5.29e-15	4.69e-15	4.70e-15	5.42e-15	5.61e-15	6.41e-15	9.23e-15
		10000	8.00e-15	1.01e-14	1.01e-14	9.65e-15	6.87e-15	1.33e-14	1.11e-14	8.59e-15	1.90e-13
	$\|g_k\|_\infty$	100	7.08e-07	7.25e-07	7.25e-07	7.03e-07	6.61e-07	7.22e-07	6.59e-07	7.87e-07	6.73e-07
		1000	9.53e-07	9.68e-07	9.68e-07	9.54e-07	9.70e-07	9.60e-07	9.62e-07	9.92e-07	4.29e-07
		10000	2.09e-06	3.17e-06	3.17e-06	1.42e-06	9.38e-07	1.66e-06	2.40e-06	1.49e-06	9.68e-07
	iter	100	78	78	78	78	78	78	78	82	211
		1000	295	295	295	295	295	295	295	296	602
		10000	966	964	964	963	966	958	960	966	3405
	feval	100	392	392	392	392	392	392	392	574	453
		1000	1477	1477	1477	1477	1477	1477	1477	2072	1377
		10000	4858	4952	4952	4817	4884	4792	4880	6762	7966
	restart	100	0	0	0	0	0	0	0	0	0
		1000	0	0	0	0	0	0	0	0	0
		10000	0	0	0	0	0	0	0	0	0
	time	100	0.050677	0.0406198	0.0458265	0.0343289	0.0361344	0.046279	0.0367742	0.0529851	0.0284235
		1000	0.642863	0.6568641	0.6139423	0.6396829	0.6301451	0.6434906	0.6158024	0.8313085	0.4940453
		10000	16.99589	17.0478304	17.5149429	19.126065	22.5562743	26.110554	29.740924	43.0978657	62.832323
II-C, init II.6	$\|f_k\|$	100	1.26e-14	1.27e-14	1.27e-14	1.89e-14	1.89e-14	1.04e-14	1.03e-14	7.38e-15	1.62e-13
		1000	5.23e-15	5.92e-15	5.92e-15	5.92e-15	5.92e-15	5.95e-15	5.23e-15	5.96e-15	1.89e-13
		10000	6.37e-15	6.37e-15	6.37e-15	6.37e-15	6.37e-15	6.37e-15	6.37e-15	6.37e-15	3.52e-13
	$\|g_k\|_\infty$	100	8.47e-07	8.74e-07	8.74e-07	9.98e-07	9.88e-07	7.35e-07	7.22e-07	7.34e-07	8.91e-07
		1000	9.81e-07	9.30e-07	9.30e-07	9.21e-07	9.17e-07	8.23e-07	9.77e-07	7.85e-07	9.20e-07
		10000	1.62e-06	1.62e-06	1.62e-06	1.62e-06	1.62e-06	1.62e-06	1.62e-06	1.62e-06	9.10e-07
	iter	100	53	53	53	52	52	53	53	54	94
		1000	134	133	133	133	133	133	134	133	258
		10000	118	118	118	118	118	118	118	118	55
	feval	100	267	267	267	262	262	267	267	378	201
		1000	672	667	667	667	667	667	672	931	600
		10000	592	592	592	592	592	592	592	826	128
	restart	100	0	0	0	0	0	0	0	0	0
		1000	0	0	0	0	0	0	0	0	0
		10000	0	0	0	0	0	0	0	0	0
	time	100	0.067785	0.0433709	0.045824	0.0393586	0.0492866	0.0417602	0.0462207	0.0431665	0.0213233
		1000	0.372252	0.3314737	0.3272801	0.3161074	0.3585586	0.3465939	0.3596169	0.4728808	0.2972237
		10000	2.509842	2.5324375	2.5694988	2.5673052	2.5389592	2.6677426	2.55041	3.7531675	0.5097433
II-D	$\|f_k\|$	100	1.70e-04	8.74e-12	7.08e-12	9.66e-12	2.51e-12	9.91e-12	8.81e-12	7.12e-12	1.12e-10
		1024	9.27e-03	2.31e-10	2.11e-10	3.24e-11	1.44e-09	1.48e-10	1.61e-10	6.38e-10	2.95e-04
		10000	1.07e-02	1.47e-05	1.49e-05	4.62e-05	8.24e-04	1.60e-05	1.46e-05	1.20e-05	3.16e-03
	$\|g_k\|_\infty$	100	5.99e-03	8.69e-07	9.72e-07	9.59e-07	9.84e-07	9.76e-07	7.74e-07	9.04e-07	8.77e-07
		1024	1.83e-02	9.86e-07	9.94e-07	9.96e-07	8.78e-06	9.33e-07	9.97e-07	9.69e-07	2.52e-04
		10000	5.56e-03	4.99e-05	5.68e-05	2.77e-04	1.50e-03	5.83e-05	5.05e-05	4.23e-05	2.09e-04
	iter	100	10000	450	432	457	1267	426	427	297	704
		1024	10000	3599	3876	4853	10000	3725	3615	2857	10000
		10000	10000	10000	10000	10000	10000	10000	10000	10000	10000
	feval	100	58854	2265	2176	2301	6561	2148	2150	2039	1634
		1024	59579	18059	19419	24301	50892	18672	18112	19988	23274
		10000	58763	50145	50147	50106	54026	50160	50143	69978	23367
	restart	100	0	0	0	0	0	4	0	25	0
		1024	0	1	0	0	0	3	0	14	0
		10000	0	0	0	0	0	0	0	25	0
	time	100	6.538244	0.2000637	0.19538	0.1818703	0.6018051	0.1662023	0.1972158	0.1370235	0.0811191
		1024	12.03537	3.0929244	3.4007555	5.2101028	11.2643516	3.8510641	3.7159254	3.3310899	2.9656018
		10000	217.8643	178.1999789	183.8311629	187.0675	224.197684	217.82597	222.44201	326.0496891	125.10982



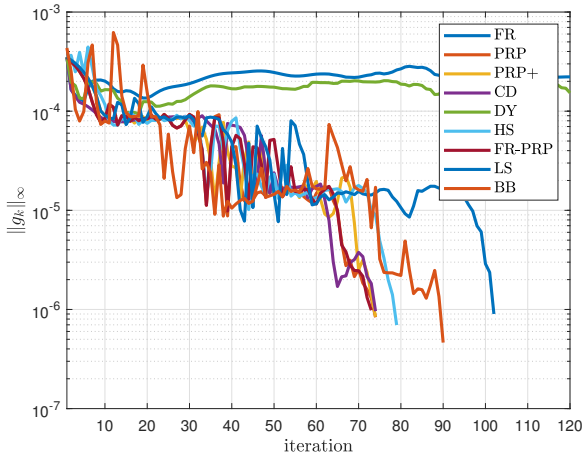
(a) $n = 100, |f_k|$



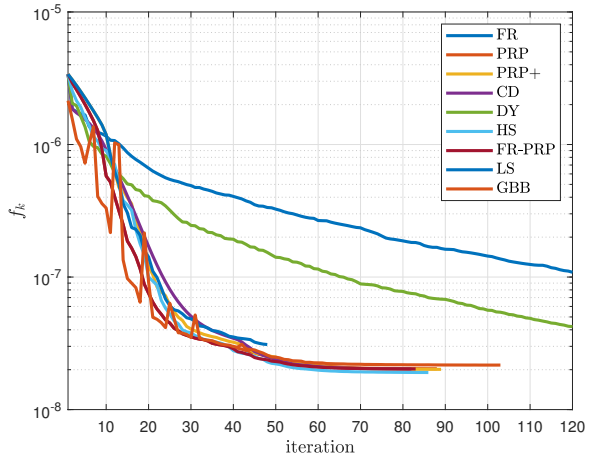
(b) $n = 100, \|g_k\|_\infty$



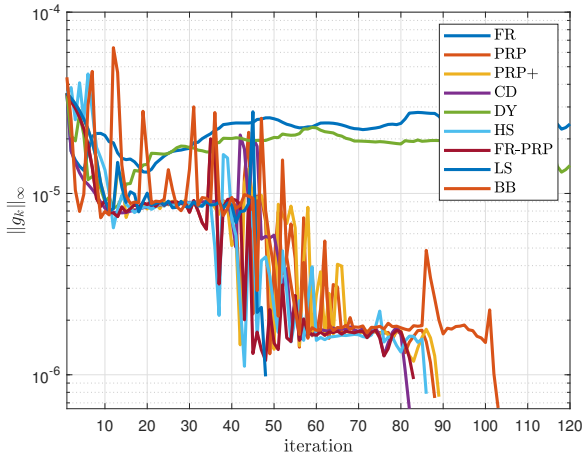
(c) $n = 1000, |f_k|$



(d) $n = 1000, \|g_k\|_\infty$

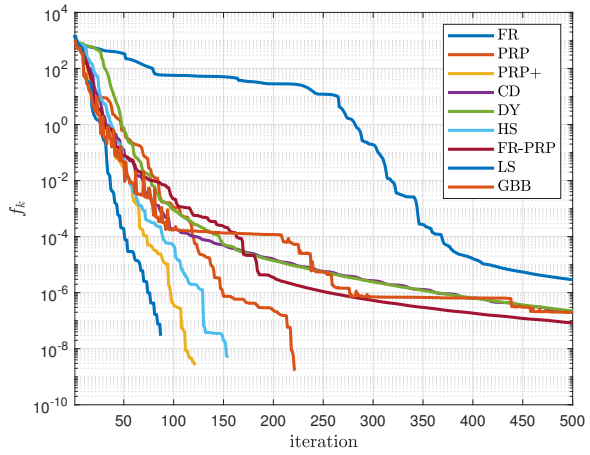


(e) $n = 10000, |f_k|$

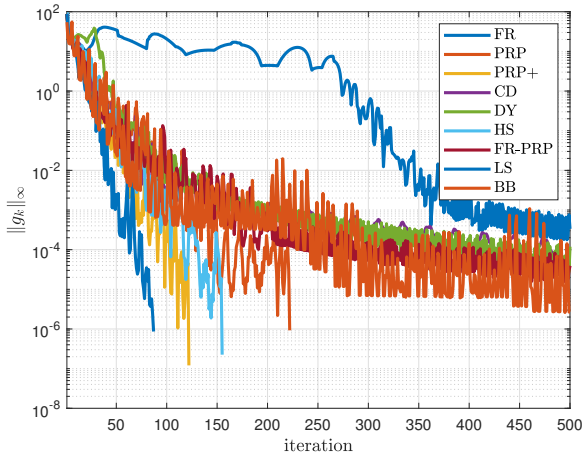


(f) $n = 10000, \|g_k\|_\infty$

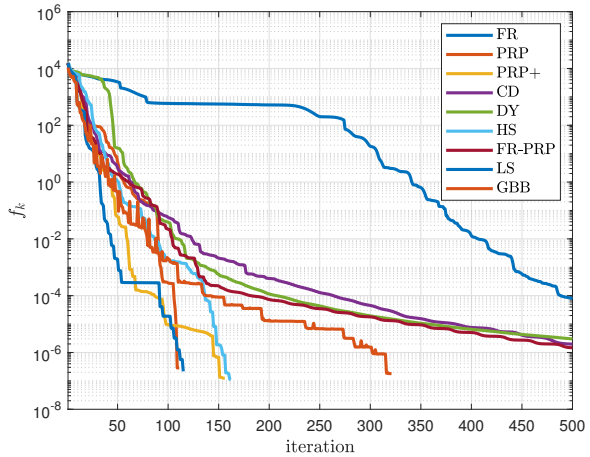
Fig. IV.1. Tridiagonal function



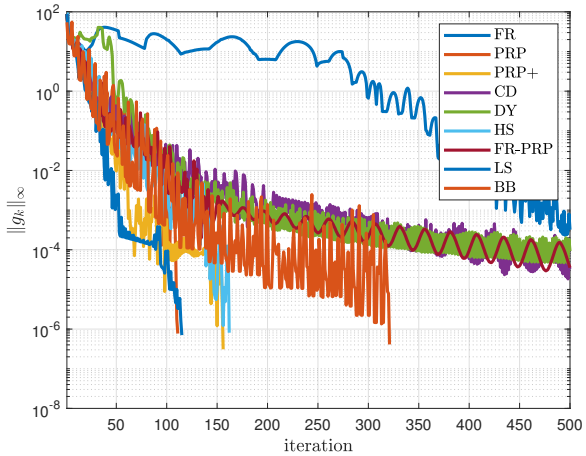
(a) $n = 100, |f_k|$



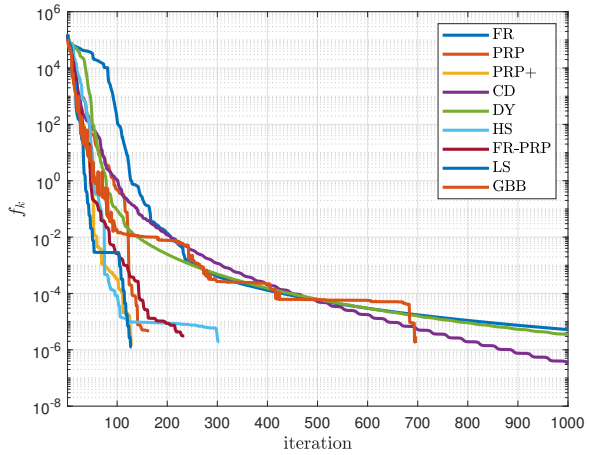
(b) $n = 100, \|g_k\|_\infty$



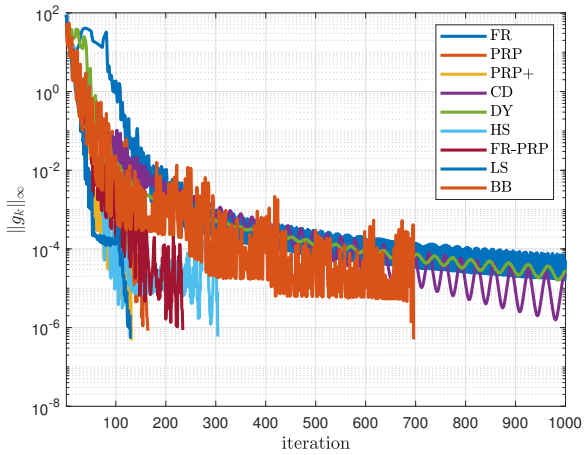
(c) $n = 1000, |f_k|$



(d) $n = 1000, \|g_k\|_\infty$

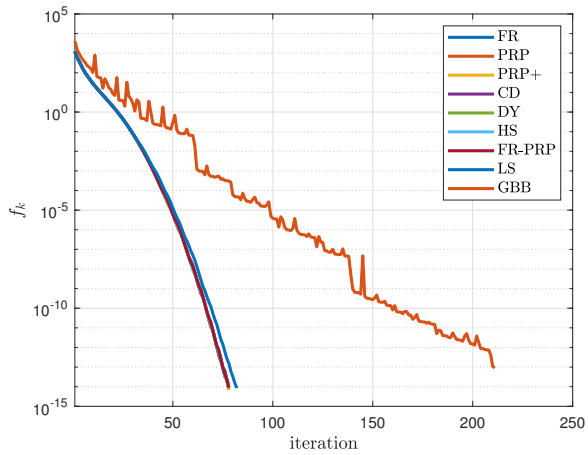


(e) $n = 10000, |f_k|$

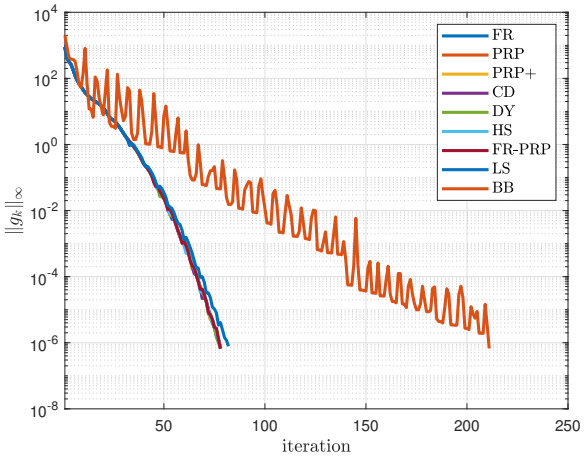


(f) $n = 10000, \|g_k\|_\infty$

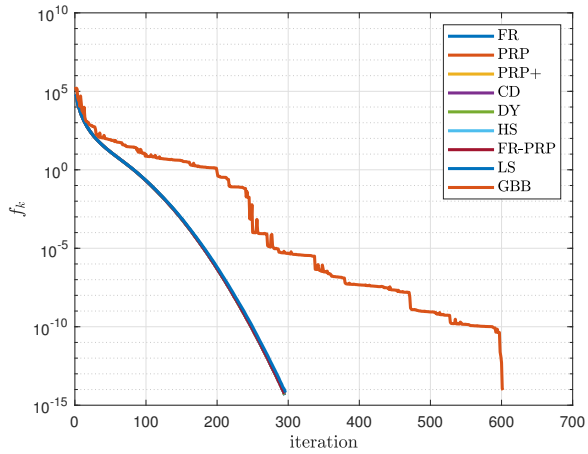
Fig. IV.2. Extended Powell function



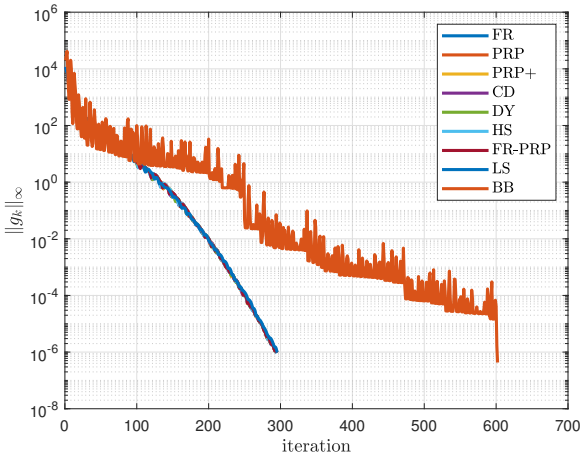
(a) $n = 100, |f_k|$



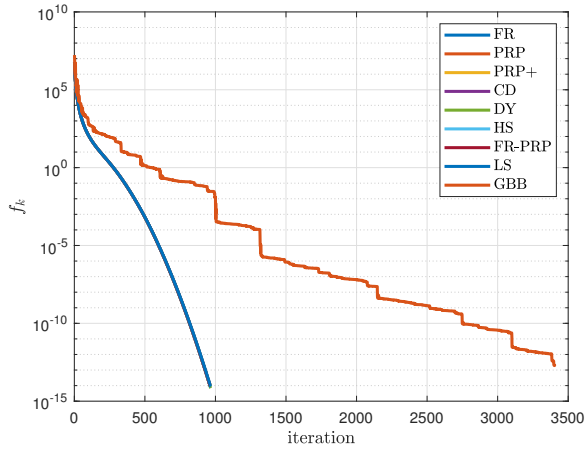
(b) $n = 100, \|g_k\|_\infty$



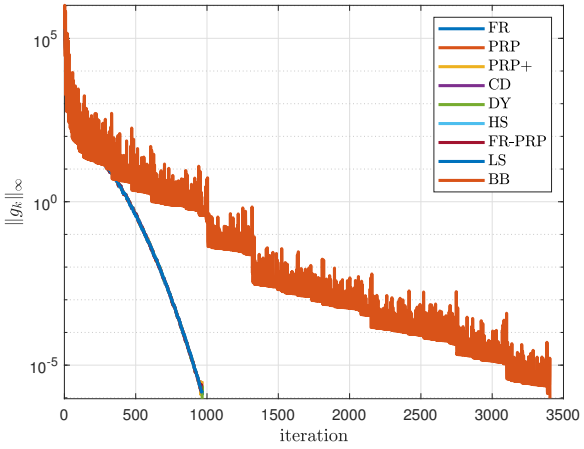
(c) $n = 1000, |f_k|$



(d) $n = 1000, \|g_k\|_\infty$

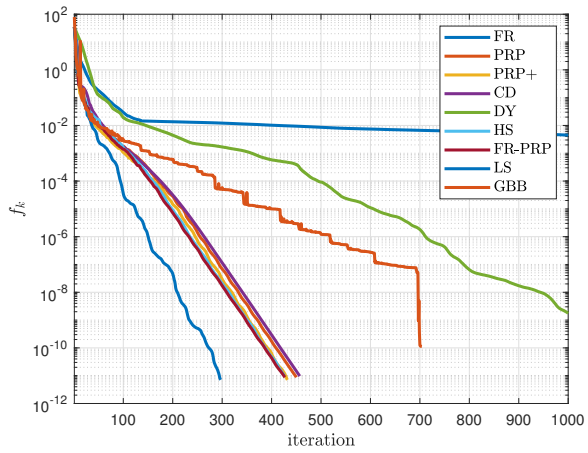


(e) $n = 10000, |f_k|$

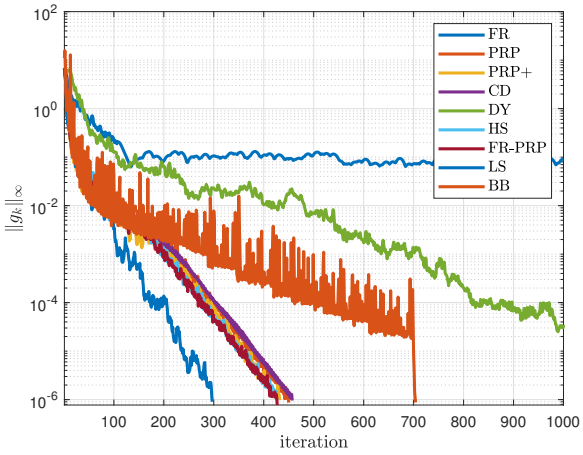


(f) $n = 10000, \|g_k\|_\infty$

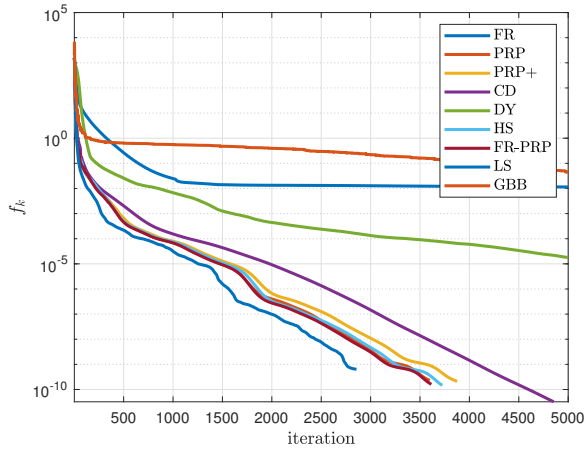
Fig. IV.3. Tridiagonal function



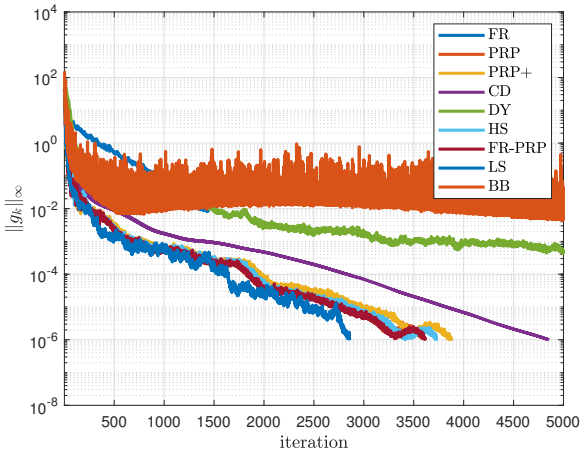
(a) $n = 100, |f_k|$



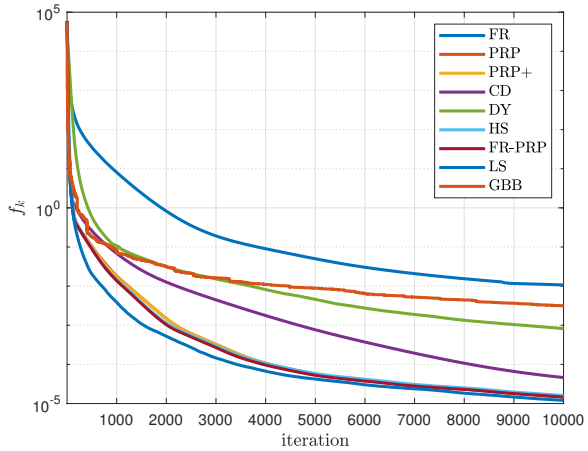
(b) $n = 100, \|g_k\|_\infty$



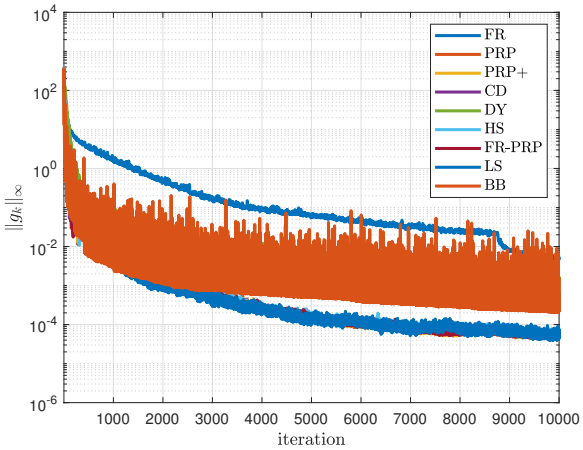
(c) $n = 1024, |f_k|$



(d) $n = 1024, \|g_k\|_\infty$



(e) $n = 10000, |f_k|$



(f) $n = 10000, \|g_k\|_\infty$

Fig. IV.4. Matrix square root 1

REFERENCES

[Dennis Jr and Schnabel, 1996] Dennis Jr, J. E. and Schnabel, R. B. (1996). *Numerical methods for unconstrained optimization and nonlinear equations*, volume 16. Siam.

[Gilbert and Nocedal, 1992] Gilbert, J. C. and Nocedal, J. (1992). Global convergence properties of conjugate gradient methods for optimization. *SIAM Journal on optimization*, 2(1):21–42.

[Hu and Storey, 1991] Hu, Y. and Storey, C. (1991). Efficient generalized conjugate gradient algorithms, part 2: Implementation. *Journal of Optimization Theory and Applications*, 69(1):139–152.

[Raydan, 1997] Raydan, M. (1997). The barzilai and borwein gradient method for the large scale unconstrained minimization problem. *SIAM Journal on Optimization*, 7(1):26–33.