Lab 2: Gradient Based Methods: Nonlinear CG Method and Global BB Method

Ting Lin, 1700010644

May 1, 2020

Contents

In this report we introduce some gradient based methods, including nonlinear CG method and GBB method. The advantage of gradient based method is that they only utilize the information of gradient, which will be free from large computation involving matrix. Suitable algorithm will make the gradient based methods more powerful and effective. After introducing the algorithms, several groups of numerical experiments will be performed, displayed and discussed.

1 Nonlinear CG Method

1.1 General Framework

Inspired by Conjugate Gradient (CG) method solving the linear symmetric and positive definite (SPD) system, we propose the following Nonlinear CG method solving general optimization problem.

Algorithm 1 Nonlinear CG

Input: f, x_0, $\varepsilon > 0$
Output: x_k

$k = 0, g_k = g(x_k), f_k = f(x_k)$
while $\underline{\|g_k\|_\infty > \varepsilon(1 + |f_k|)}$ **do**
  $\alpha_k = \text{LineSearch}(f, x_k, d_k)$
  $x_{k+1} = x_k + \alpha_k d_k$
  Compute β_k, $d_{k+1} = -g_{k+1} + \beta_k d_k$
  $k = k + 1$
  $f_k = f(x_k), g_k = g(x_k)$
end

In practice, we slightly modify the line search step in order to assure the direction is descent.

Algorithm 2 Nonlinear CG(ii)

Input: f, x_0, $\varepsilon > 0$
Output: x_k

$k = 0, g_k = g(x_k), f_k = f(x_k)$
while $\|g_k\|_\infty > \varepsilon(1 + |f_k|)$ **do**
  if $g_k^T d_k > 0$ **then**
    $\tilde{d}_k = -d_k/|d_k|$
  else
    $\tilde{d}_k = d_k/|d_k|$
  end
  $\alpha_k = \text{LineSearch}(f, x_k, \tilde{d}_k)$
  $x_{k+1} = x_k + \alpha_k \tilde{d}_k$
  Compute β_k, $d_{k+1} = -g_{k+1} + \beta_k d_k$
  $k = k + 1$
  $f_k = f(x_k), g_k = g(x_k)$
end

In some applications, we can use restart technique to make the convergence faster.

Algorithm 3 Nonlinear CG(iii)

Input: f, x_0, $\varepsilon > 0$
Output: x_k

$k = 0, g_k = g(x_k), f_k = f(x_k)$

while $\|g_k\|_\infty > \varepsilon(1 + |f_k|)$ **do**
  if $g_k^T d_k > 0$ **then**
    $\tilde{d}_k = -d_k/|d_k|$
  else
    $\tilde{d}_k = d_k/|d_k|$
  end
  $\alpha_k = \text{LineSearch}(f, x_k, \tilde{d}_k)$
  $x_{k+1} = x_k + \alpha_k \tilde{d}_k$
  if $M|k$ **then**
    $d_{k+1} = -g_{k+1}$
  else
    Compute β_k, $d_{k+1} = -g_{k+1} + \beta_k d_k$
  end
  $k = k + 1$
  $f_k = f(x_k), g_k = g(x_k)$
end

It remains to discuss the choice of linesearch method. Without specifying we will use Stroing wolfe rule, stop when the max iteration reached. But as we will see that in some case the exact line search method is powerful.

1.2 Choice of Direction Updates

We consider the following updating strategies.

1. FR Formula
$$\beta = \frac{g_{k+1}^T g_{k+1}}{g_k^T g_k}$$
2. PRP Formula
$$\beta = \frac{g_{k+1}^T g_{k+1} - g_{k+1}^T g_k}{g_k^T g_k}$$
3. PRP+ Formula
$$\beta = \max(\beta^{PRP}, 0)$$
4. FR-PRP Formula
$$\textbf{if} \quad |\beta^{PRP}| < \beta^{FR}; \beta = \beta^{PRP}$$
$$\textbf{if} \quad \beta^{PRP} < -beta^{FR}; beta = -\beta^{FR}$$
$$\textbf{otherwise} \quad \beta = \beta^{FR}$$

Also, we consider the Hu-Storey Algorithm, which β is computed from the following system.
The update rule is $d_k^+ = \alpha g_k + \beta d_{k-1}$, where

$$\begin{bmatrix} \alpha \\ \beta \end{bmatrix} = - \begin{bmatrix} g_k^T H g_k & g_k^T H d_{k-1} \\ g_k^T H d_{k-1} & d_{k-1}^T H d_{k-1} \end{bmatrix}^{-1} \begin{bmatrix} g_k^T g_k \\ g_k^T d_{k-1} \end{bmatrix}$$

1.3 The Detailed Algorithm of Hu–Storey's Method

We rewrite the Hu–Steorey's algorithm with some techniques. Here some procedure in the original paper is simplified since in higher dimension it will force HS method more likely to the gradient descent.

2 Global BB Method

2.1 Naive BB Method

Naive BB algorithm only utilized the gradient message and solve the stepsize α by

$$\alpha_k = \arg\ \min_{\alpha > 0} \|\alpha^{-1} s_{k-1} - y_{k-1}\|_2^2$$

or

$$\alpha_k = \arg\ \min_{\alpha > 0} \|s_{k-1} - \alpha y_{k-1}\|_2^2$$

Here $s_{k-1} = x_k - x_{k-1}$ and $y_{k-1} = g_k - g_{k-1}$. Solving the two problem yields two choices of stepsize

$$\alpha_k^{BB1} = \frac{s_{k-1}^T s_{k-1}}{s_{k-1}^T y_{k-1}}, \qquad \alpha_k^{BB2} = \frac{s_{k-1}^T y_{k-1}}{y_{k-1}^T y_{k-1}}.$$

We write it into the whole algorithm, deriving the naive BB method.

Input: f, x_0, $\varepsilon > 0$

Output: x_k

$k = 0, g_k = g(x_k), f_k = f(x_k)$

$Counter = 0$

while $\|g_k\|_\infty > \varepsilon(1 + |f_k|)$ **do**

  if $g_k^T d_k > 0$ **then**

    $\tilde{d}_k = -d_k/|d_k|$

  else

    $\tilde{d}_k = d_k/|d_k|$

  end

  $\alpha_k = \text{LineSearch}(f, x_k, \tilde{d}_k)$

  $x_{k+1} = x_k + \alpha_k \tilde{d}_k$

  $g_{k+1} = g(x_{k+1})$

  $A_k = (g(x_k + \varepsilon \cdot d_k) - g(xk - \varepsilon \cdot d_k))/(2\varepsilon)$

  $B_k = (g(x_k + \varepsilon \cdot g_{k+1}) - g(xk - \varepsilon \cdot g_{k+1}))/(2\varepsilon)$

  $t_k = d_k^T A_k, v_k = g_{k+1}^T B_k, u_k = g_{k+1}^T A_k$

  if $Counter = M$ **then**

    $Counter = 0$

    $d_k = -g_{k+1}, k = k + 1$

    continue

  end

  $Counter = Counter + 1$

  $w_k = t_k v_k - u_k^2$

  $d_k = 1/w_k((u_k g_{k+1}^T d_k - t_k g_{k+1}^T g_{k+1}) * gk + (u_k g_{k+1}^T g_{k+1} - v_k g_{k+1}^T d_k) * dk)$

end

Algorithm 4 Naive-BB

Input: f, x_0

Output: x_k

$k = 1$

Using steepest descent method and line search to obtain x_1.

Compute $g_0 = g(x_0), g_1 = g(x_1)$

while $\|g_k\|_\infty > \varepsilon(1 + |f_k|)$ **do**

  $s_{k-1} = x_k - x_{k-1}$

  $y_{k-1} = g_k - g_{k-1}$

  Compute α_k being either α^{BB1} or α^{BB2}

  $x_{k+1} = x_k - \alpha_k g_k$

  $g_{k+1} = g(x_{k+1}), k = k + 1$

end

2.2 BB Method with non-monotone line search

We add a non-monotone line search to make the algorithm more flexible and practical.

Algorithm 5 Global BB

Input: f, x_0, ε, δ, γ, σ, M
Output: x_k
$k = 1$
Using steepest descent method and line search to obtain x_1.
Compute $g_0 = g(x_0), g_1 = g(x_1)$
while $\|g_k\|_\infty > \varepsilon(1 + |f_k|)$ **do**
  $s_{k-1} = x_k - x_{k-1}$
  $y_{k-1} = g_k - g_{k-1}$
  Compute α_k being either α^{BB1} or α^{BB2}
  if $\alpha_k < \varepsilon$ or $\alpha_k > 1/\varepsilon$ **then**
    $\lambda = \delta$
  else
    $\lambda = 1/\alpha_k$
  end
  while $f(x_k - \lambda g_k) < \max_{j<\min k,M} f(x_{k-j}) - \lambda\gamma g_k^T g_k$ **do**
    $\lambda = \lambda \cdot \sigma$
  end
  $g_{k+1} = g(x_{k+1}), k = k + 1$
end

3 Numerical Experiments

We test the assignded problem in MATLAB, the code is attached and the result is tested in my laptop (I7-6700HQ).

3.1 Problem Setting

In this report we choose the following problem and test the success, calls of f and g, CPU time for each method.

TRIGonometric fuction

$$F(x) = \sum_{i=1}^{n}\{n + i - \sum_{j=1}^{n}[a_{ij}\sin(x_j) + b_{ij}\cos(x_j)]\}^2$$

where $a_{ij} = \delta_{ij}$, $b_{ij} = i\delta_{ij} + 1$, $delta$ is the kronekcer-delta. The initial value is selected as

$$x_0 = (1/n, \cdots, 1/n)^T$$

We choose $n = 100, 1000, 10000$ (denoted as trig2, trig3,trig4)

Extended Powell function

$$F(x) = \sum_{j=1}^{n/4}[(x_{4j-3} + 10x_{4j-2})^2 + 5(x_{4j-1} - x_{4j})^2 + (x_{4j-2} - 2x_{4j-1})^4 + 10(x_{4j-3} - x_{4j})^4]$$

The initial value is

$$x_0 = (3, -1, 0, 3, \cdots,)^T$$

We choose $n = 100, 1000, 10000$.(denoted as ep2, ep3,ep4)

TRIDiagonal function

$$F(x) = \sum_{i=2}^{n}[i(2x_i - x_i - 1)^2]$$

with initial value $x_0 = ones(n, 1)$. We choose $n = 100, 1000, 10000$ (denoted as trid2, trid3,trid4)

MATtrix square root We want to minimize the following question

$$\min_B \|B^2 - A\|_F^2$$

here $A = \textbf{reshape}(\sin((1 : n).^2), [n, n])^2$ and the initial point is $0.2 * \textbf{reshape}(\sin((1 : n).^2), [n, n])$ We choose $n = 10, 32, 100$ (denoted as mat2, mat3,mat4)

3.2 Numerical Results for trig

We first test our methods in the trig problem, here we use the exact line search method and do not restart at all.

Table 1: Results on trig2, trig3, trig4

		FR	PRP+	FR-PRP	HS	GBB
2	CPU TIME	1.02E+00	2.00E-01	2.00E-01	7.20E-01	4.68E-03
	#f	46481	8467	8467	31707	79
	#g	281	52	52	956	75
	Iteration	281	52	52	192	75
3	CPU TIME	3.57E+00	4.90E-01	4.80E-01	1.16E+00	1.01E-02
	#f	64575	9131	8633	20917	78
	#g	390	56	53	631	70
	Iteration	390	56	53	127	70
4	CPU TIME	7.07E+00	2.15E+00	2.24E+00	2.26E+00	6.56E-02
	#f	31707	9629	9629	9795	101
	#g	192	59	59	296	89
	Iteration	192	59	59	60	89

3.3 Numerical Results for ep

We test our methods in ep problem, here we use the inexact line search with max search time 5,and do restart after each 20 iterations.

Table 2: Results for ep2, ep3, ep4

		FR	PRP+	FR-PRP	HS	GBB
2	CPU TIME	7.99E-01	1.02E-01	7.21E-01	1.72E+01	2.17E-02
	#f	6905	887	5677	120045	339
	#g	4607	585	3793	154669	251
	Iteration	1150	137	947	20000	251
3	CPU TIME	6.43E+00	1.01E-01	6.23E+00	8.70E-01	7.57E-02
	#f	19927	317	19359	1985	271
	#g	13293	229	12922	2594	203
	Iteration	3322	55	3229	335	203
4	CPU TIME	5.15E+00	1.11E+00	6.37E+00	6.49E+00	5.17E-01
	#f	2325	487	2757	1873	221
	#g	1563	345	1867	2459	170
	Iteration	389	81	464	316	170

3.4 Numerical Results for trid

We test our methods in ep problem, here we use the inexact line search with max search time 5,and do not restart at all.

Table 3: results for trid2,trid3,trid4

		FR	PRP+	FR-PRP	HS	GBB
2	CPU TIME	9.78E-02	8.30E-02	7.88E-02	2.84E-01	1.21E-02
	#f	855	651	639	2759	267
	#g	578	442	434	2900	216
	Iteration	145	111	109	375	216
3	CPU TIME	4.44E-01	2.33E-01	2.28E-01	1.87E+00	6.06E-02
	#f	3749	1963	1963	15013	1122
	#g	2517	1328	1324	15309	806
	Iteration	629	332	331	1997	806
4	CPU TIME	3.03E+00	2.37E+00	2.30E+00	F	2.21E+00
	#f	7947	6661	6331		7018
	#g	5332	4468	4248		5039
	Iteration	1331	1115	1060		5039

3.5 Numerical Result for mat

We test our all methods except HS method (since it are too slow to converge...). We use the inexact line search with max search time 5, and do restart after each 50 iterations.

Table 4: results for mat2, mat3, mat4

		FR	PRP+	FR-PRP	HS	GBB
2	CPU TIME	2.91E+00	4.10E+00	3.77E+00		1.76E+00
	#f	29065	37423	35439		39519
	#g	19137	24662	23405		28456
	Iteration	4626	6037	5734		28456
3	CPU TIME	2.11E+01	2.21E+01	2.68E+01		5.06E+01
	#f	112095	97607	106599		388925
	#g	74189	64821	70778		283126
	Iteration	18254	16054	17533		283126
4	CPU TIME	2.38E+01	1.14E+02	9.19E+01		~2400
	#f	40209	175331	142455		~4M
	#g	26763	116877	94956		~3M
	Iteration	6670	29208	23729		~4M

3.6 Discussion

We found that BB and CG have different behaviors in different problem, and the restart is a powerful tool in CG method. However, both CG and BB methods are possible to reduce into gradient descent and hence less attractive. Moreover, the HS method sometimes failed in practice, this means we might choose more effective algorithm.