Lab 1: Newton and Quasi-Newton Methods

Ting Lin, 1700010644

April 8, 2020

Contents

In this section we introduce several Newton and Quasi-Newton methods, and some of their variants, with necessary numerical experiments. Before the concrete algorithms, we first need to introduce the basic framework of optimization, and some methods in line search. Then we introduce Newton methods, typically damped Newton (i.e. Newton with line search) with two modifications(so-called **mixed** and **LM**). In the third part we introduce Quasi-Newton method, which reduce the requirement of excessive computational complexity on Hessian. We end this report by showing numerical experiments on some examples with different scales, testing the efficiency and difference of each method. For notation convention, we use f, g, h to denote the target, gradient and hessian if no ambiguity.

1 Basic Framework and Line Search

1.1 Basic Framework of Optimization

The basic framework of optimization discussed in this report are as follows:

Algorithm 1 Framework of optimization method

Input: f, x_0
Output: x_{out}

Given initial Point x_0, $k = 0$
while stop critertion does not meet **do**
  Compute the descent direction d_k
  if Inexact Line Search return α_k sucessfully **then**
    $x_{k+1} = x_k + \alpha_k d_k$
  else
    Exact Line Search to obtain x_{k+1}
  end
  $k = k + 1$
end

Usually the stop criterion might be chosen as $|x_k - x_{k-1}| \leq tol$, or $|g_k| \leq tol$ or $|f(x_k) - f(x_{k-1})| \leq tol$. In this report we choose the first criterion in implementation. We first introduce the line search method. By setting $\phi(a) = f(x_k + ad_k)$

1.2 Exact Line Search Method: .618 method

We first introduce the .618 method for exact line search. An auxiliary algorithm is **Find-Initial-Interval**, aiming to find a initial interval l, r such that there exists an $m \in (l, r)$, and $f(m) < f(l), f(r)$.

Algorithm 2 Find-Initial-Interval

Input: ϕ // ϕ need to be coerocive.
Output: Initial Search Interval: (l, r)

$x = 1$
if $\phi(x) < \phi(0)$ **then**
  while $\phi(x) < \phi(0)$ **do**
    $x = 2x$
  end
  return $(0, x)$
else
  if $\phi(-x) < \phi(0)$ **then**
    while $\phi(-x) < \phi(0)$ **do**
      $x = 2x$
    end
    return $(-x, 0)$
  else
    return $(-1, 1)$
  end
end

Clearly, Algorithm 1.2 will stop if ϕ is coerocive, i.e.

$$\lim_{|x|\to\infty} \phi(x) = \infty$$

. Here and throughout this report, we will assume our function is coerocive, so are our numerical experiments.

Based on the algorithm showed above, we introduce the .618 method.

Algorithm 3 .618 method

Input: ϕ ; // ϕ need to be coerocive.
Output: Stepsize: α

(l, r) = Find-Initial-Interval(ϕ)
while $r - l < tol$ **do**
  $m_1 .618 * l + .382 * r, \qquad m_2 = .618 * r + .382 * l$
  if $m_1 > m_2$ **then**
    $l = m_1$
  else
    $r = m_2$
  end
end
return $\frac{l+r}{2}$

1.3 Inexact Line Search Method

However, exact line search method is too expansive for practical applications, hence we need a more efficient method. The **inexact line search method** can be thought as a great alternative. We give three type of inexact line search

method: backtracking and zoom(interpolation).

Algorithm 4 Backtracking

Input: ϕ
Output: α
Initialize α
$iter = 0$
while Rule(α) is False and iter < MAXITER **do**
    $\alpha = 0.9 * \alpha$
    iter = iter + 1
end

The $Rule(\alpha)$ can be chosen as one of following (Here $g_k = g(x_k)$):

- Armijo: $\phi(\alpha) - \phi(0) \leq \rho\alpha g_k^T d_k$
- Goldstein: $(1-\rho)\alpha g_k^T d_k \leq \phi(\alpha) - \phi(0) \leq \rho\alpha g_k^T d_k$
- Strong Wolfe: Armijo + $|g(\alpha)^T d_k| \leq -\sigma g_k^T d_k$
- Wolfe: Armijo + $g(\alpha)^T d_k \geq \sigma g_k^T d_k$

Next we introduce the zoom method, which is based on interpolation.

Algorithm 5 Backtracking

Input: ϕ
Output: α
Initialize α
$iter = 0$
while Rule(α) is False and iter < MAXITER **do**
    Generate α^+ from α by interpolation
    iter = iter + 1
    $\alpha = \alpha^+$
end

Here we only consider the quadratic interpolation, that is, to find a quadratic polynomial $p(x)$, such that

$$p(0) = \phi(0), p(\alpha) = \phi(\alpha), p'(0) = \phi'(\alpha).$$

Then

$$\alpha^+ := \arg\min_\alpha p(\alpha).$$

Direct calculation shows

$$\alpha^+ = \frac{2\alpha^2\phi'(0)}{\phi(\alpha) - \phi(0) - \phi'(0)\alpha}.$$

2 Newton Method

2.1 Naive Newton Method

Naive Newton method is just a naive Newton method, utilizing the Hessian information directly. The following algorithm shows how to get the newton direction. It can be proved that if the initial value is close to the local minimum

Algorithm 6 Naive Newton

Input: f, x_0
Output: x_{out}

k = 0
while not converge **do**
  Calculate $H = h(x_k)$, $g = g(x_k)$, here h and g is hessian and gradient.
  $x_{k+1} = x_k - H^{-1}g$ `/* Suppose` H `is invertible here */`
  $k = k + 1$
end
return x_{k-1}

$x*$, then Newton method yields second-order convergence.

2.2 Damped Newton Method

However, the above Newton Method might be too bold, hence we apply a damped technique by using line search after getting each Newton method. Here we return to Algorithm 1.1 with direction being normalized Newton direction

$$d_k = \frac{d_k^N}{|d_k^N|},$$

where d_k^N means Newton direction in Algorithm 2.1.

2.3 Modification

We introduce two modification to avoid that the Newton direction is not a descent direction, otherwise the line search method would fail. Two simple strategies is adopted in the report and the implementation, as listed below.

These two algorithm remedy the naive Newton by let the actual direction being between Newton and minus-gradient. The mixed method chooses either one of them, and LM method changes more gently. However, this will force the Newton method have only first-order convergence rate.

3 Quasi-Newton Methods

In this section we introduce the quasi-Newton method, which only need compute Hessian once. We compute $H_0 = H(x_0)$ in advance. Here we omit the detailed deduction but only show the result in our algorithm. The first algorithm is use the objective value and gradient value to determine H_{k+1}, known as Broyden's family.

Several special cases should be mentioned. If $\varphi = 0$, we obtain DFP formula, if $\varphi = 1$ we obtain BFGS formula. They are both Rank 2 modification. Interestingly, if we choose $\varphi = xx$ we obtain the SR1 formula, it is an important

Algorithm 7 Newton-direction-mixed

Input: f, x
Output: d

Calculate $H = h(x_k)$, $g = g(x_k)$ /* If $g = 0$, we are done! */
if H is not invertible **then**
  return $-g/|g|$
end
Calculate $d = -H^{-1}g$
if $d^T g > 0.3|d||g|$ **then**
  return $-d/|d|$
end
if $d^T g < -0.3|d||g|$ **then**
  return $d/|d|$
end
return $-g/|g|$.

Algorithm 8 Newton-direction-LM

Input: f, x
Output: d

Calculate $H = h(x_k)$, $g = g(x_k)$ /* If $g = 0$, we are done! */
$\nu = 1$
$d = -H^{-1}g$
while H is not invertible or $d^T g > -0.3|d||g|$ **do**
  $d = -(H + \nu I)^{-1}g$
  $\nu = 2\nu$
end

Algorithm 9 Broyden's family

Input: x_k, x_{k+1}, H_k
Output: H_{k+1}

$g_k = g(x_k), \quad g_{k+1} = g(x_{k+1})$
$s = x_{k+1} - x_k, \quad y = g_{k+1} - g_k$
Output: H_{k+1}

$$H^{DFP} = H_k + \frac{s_k s_k^T}{s_k^T y_k} - \frac{Hk y_k y_k^T H_k}{y_k^T H_k y_k}$$

$$H^{BFGS} = H_k + (1 + \frac{y_k^T H_k y_k}{y_k^T s_k})\frac{s_k s_k^T}{y_k^T s_k} - (\frac{s_k y_k^T H_k + H_k y_k s_k^T}{y_k^T s_k})$$

return $H_{k+1} = H^{DFP} + \varphi(H^{BFGS} - H^{DFP})$

Algorithm 10 Quasi-Newton direction

Input: x_k, x_{k+1}, H_k
Output: d_{k+1}
78 Obtain H_{k+1} by Broyden's family
79 Replace the true Hessian by H_{k+1} in Newton-direction-mixed or Newton-direction-LM, obtaining d_{k+1}.
80 **return** d_{k+1}.

Rank 1 modification.

$$H_{k+1}^{SR1} = H_k + \frac{(s_k - H_k y_k)(s_k - H_k y_k)^T}{(s_k - H_k y_k)^T y_k}$$

3.1 Cost Estimates

In this subsection we estimate the cost by counting the call time of f g and h.

Line Search For each line search iteration, the cost is

- .618 method: #f: 1, #g: 0
- Armijo & Goldstein: #f:1, #g:0
- (s-)Wolfe: #f:1, #g:1

The backtracking and interpolation will not create new call.

Newton Each iteration costs: #f:0, #g:1, #h:1
LM and mixed modification methods do not create new call.
Quasi-newton method only costs #h:1, and use #g:1 each iteration.
However, in practice in our report, we use numerical gradient and numerical Hessian, using each $2n$ and $4n^2$ calls of f. Thus we only record the call time of f.

4 Numerical Experiments

In the rest of our report, we test several concrete problems. We use MATLAB to implement, running on my laptop with core I7-8700HQ.

4.1 Problem Setting

We test three type of problems. Let n denote the dimension of the problem, and the first two problems has the type :$f = \sum_{i=1}^{n} f_i^2$.

Brown and Dennis Function $n = 4$, $f_i = (x_1 + t * x_2 - e^t)^2 + (x_3 + \sin(t) * x_4 + \cos(t))^2$, where $t = \frac{i}{5}$ for $i = 1, \cdots, m$.
We choose the initial value $[25, 5, -5, 1]^T$, and $m = 4, 10, 20, 30, 40, 50$ throughout this paper.

Discrete Integral Equation $n = m$,

$$f_i = x_i + \frac{h}{2}[(1-t_i)\sum_{j\le i} t_j(x_j+t_j+1)^3 + t_i\sum_{j>i}(1-t_j)(x_j+t_j+1)^3]$$

where $h = \frac{1}{n+1}, t_j = th$.

The initial value is chosen as $x_j = t_j(t_j - 1)$.

Minimal Surface Equation We turn to seek a solution of the following minimal surface equation,

$$\min_u \int_\Omega \sqrt{1+|\nabla u|^2}du$$

with boundary condition $u(x,y)|D = x^2 + y^2$. Here Ω is $[0,1]^2$ and D is its boundary. For a given square grid $U_{i,j}(0 \le i,j \le n)$, we turn to minimize the following discrete functional.

$$\sum_{i=1}^{n}\sum_{j=1}^{n}(1+(\frac{U_{ij}-U_{i-1,j}}{h})^2+(\frac{U_{ij}-U_{i,j-1}}{h})^2)^{1/2} + \sum_{i=1}^{n}\sum_{j=1}^{n}(1+(\frac{U_{ij}-U_{i+1,j}}{h})^2+(\frac{U_{ij}-U_{i,j+1}}{h})^2)^{1/2}.$$

The initial value is randomly chosen in $N(0,1)$.

4.2 Implementation Details

In this subsection we show some implementation details, some of them might be discussed in the previous section and some of them are just to clarify our choice.

1. Line Search Rule: We choose strong Wolfe, with $\rho = .3,\ \ \sigma = .6$. The max iteration is set as 100. Once the line search method fails, we turn to use exact line search (.618 method) to seek for step size.
2. Modification: for damped Newton method, we use both "mixed" and "LM". For quasi-Newton method, we use only "mixed".

4.3 Numerical Result

In this subsection we test our algorithms in the problems. We first show the power of damped Newton method. We use "bd[m]" to denote the Brown and Dennis function with size m, and likewise we use "die[m]" and "mse[m]".

We record the iteration, subiteration, the call of f (including numerical gradient and hessian) and the value to test whether our algorithm is successful.

	NONE					MIXED					LM				
	iter	sub	time(s)	#f	value	iter	sub	time(s)	#f	value	iter	sub	time(s)	#f	value
bd4	18	533	0.19	10113	1.05E-05	19	607	0.19	13151	1.05E-05	22	707	0.21	15081	1.05E-05
bd10	10	319	0.09	5188	1.44E+00	11	396	0.13	7117	1.44E+00	11	432	0.14	7347	1.44E+00
bd20	12	415	0.10	5118	8.58E+04	9	312	0.09	5556	8.58E+04	47	1632	0.52	28900	8.58E+04
bd30	16	710	0.21	8377	9.77E+08	28	1061	0.41	20845	9.77E+08	18	620	0.18	9676	9.77E+08
bd40	16	744	0.15	7082	5.86E+12	71	2793	0.98	54692	5.86E+12	59	2265	0.78	42129	5.86E+12
bd50	46	1798	0.61	32616	2.67E+16	16	721	0.21	11832	2.67E+16	239	9247	3.40	171884	2.67E+16

	NONE					MIXED					LM				
	iter	sub	time(s)	#f	value	iter	sub	time(s)	#f	value	iter	sub	time(s)	#f	value
die2	11	245	0.14	4565	8.88E-20	11	245	0.14	4565	8.88E-20	11	245	0.13	4565	8.88E-20
die10	12	299	0.44	16848	2.83E-18	12	299	0.46	16848	2.83E-18	12	299	0.49	16848	2.83E-18
die20	13	383	1.39	42879	1.87E-18	12	295	1.29	40361	2.58E-18	12	350	1.43	40771	1.29E-18
die30	12	349	2.50	62727	2.71E-18	11	293	2.30	59142	1.82E-18	12	312	2.94	74996	4.53E-19
die40	15	390	6.79	144560	9.30E-19	13	336	5.47	116936	7.56E-20	13	387	5.49	119199	6.14E-18
die50	17	479	12.56	225927	3.84E-18	10	254	7.69	137170	4.22E-18	13	355	9.92	178765	3.46E-18

	NONE					MIXED					LM				
	iter	sub	time(s)	#f	value	iter	sub	time(s)	#f	value	iter	sub	time(s)	#f	value
mse3	19	112	0.14	2971	1.46E+00	19	112	0.14	2971	1.46E+00	19	112	0.12	2971	1.46E+00
mse5	17	68	0.73	21029	2.08E+00	14	35	0.62	17079	2.08E+00	16	70	0.90	22715	2.08E+00
mse7	18	175	3.94	111099	2.40E+00	18	71	3.58	101257	2.40E+00	16	116	3.38	97620	2.40E+00

Table 1: Damped Newton Method with several modifications.

	$\varphi = 0$					$\varphi = 1/2$				
	iter	sub	time(s)	#f	value	iter	sub	time(s)	#f	value
bd4	1484	51901	20.55242	1383707	1.05E-05	1381	47958	20.44	1466973	1.05E-05
bd10	2000	80026	27.54	1678135	1.44E+00	5530	221197	63.52	3639777	1.44E+00
bd20	1617	64644	18.35	919048	8.58E+04	66	2645	0.80	43293	8.58E+04
bd30	170	6832	2.09	107212	9.77E+08	70	2831	0.88	45195	9.77E+08
bd40	234	9407	2.92	148971	5.86E+12	106	4291	1.53	72682	5.86E+12
bd50	182	7384	2.97	119346	2.67E+16	836	33341	13.63	588864	2.67E+16
	$\varphi = 1$					SR1				
bd4	2814	110293	48.46304	3477151	1.05E-05	899	31943	9.540389	555412	1.05E-05
bd10	332	13311	3.732782	213557	1.4432255	1222	48885	13.79036	768891	1.4432255
bd20	47	1885	0.507918	25765	85822.202	1136	45439	14.88648	712013	85822.202
bd30	110	4431	1.596179	73979	976882218	128	5149	1.490632	74972	976882218
bd40	60	2397	1.121881	38197	5.856E+12	212	8507	2.992672	150455	5.856E+12
bd50	523	20950	10.44849	380114	2.67E+16	510	20436	7.583633	333912	2.67E+16

Table 2: Broyden's family for Brown and Dennis function, choosing DFP, BFGS, $\varphi = 1/2$ and SR1.

	$\varphi = 0$					$\varphi = 1/2$				
	iter	sub	time(s)	#f	value	iter	sub	time(s)	#f	value
die2	15	368	0.288981	5997	1.17E-18	12	248	0.14	3927	2.64E-18
die10	48	1815	2.10	71406	6.36E-17	55	2095	2.45	82235	1.49E-17
die20	53	1941	4.21	123978	1.15E-16	61	2298	5.19	152758	5.76E-17
die30	48	1812	8.43	212699	9.31E-17	58	2173	7.30	183230	2.55E-16
die40	49	1853	12.99	249706	7.68E-17	51	1933	13.11	260561	1.32E-16
die50	21	733	5.58	97410	1.45E-16	48	1813	16.96	295933	2.40E-16
	$\varphi = 1$					SR1				
die2	22	686	0.357146	10552	4.30E-18	15	368	0.18309	6002	4.28E-18
die10	64	2455	3.034302	99611	7.06E-17	32	1175	1.420462	46141	1.78E-17
die20	55	2058	4.215484	125558	1.88E-16	36	1261	2.794558	79110	4.31E-18
die30	62	2372	9.121304	233425	2.20E-16	31	1132	4.239021	112188	1.31E-17
die40	67	2573	14.63883	306334	2.31E-16	33	1213	8.042353	170885	2.21E-17
die50	17	573	7.425253	130492	3.16E-18	25	893	9.095294	149673	8.53E-17

Table 3: Broyden's family for Discrete Integration Equation, choosing DFP, BFGS, $\varphi = 1/2$ and SR1.

	$\varphi = 0$					$\varphi = 1/2$				
	iter	sub	time(s)	#f	value	iter	sub	time(s)	#f	value
mse3	871	1831	1.75	36299	1.46E+00	772	1606	1.63	32186	1.46E+00
mse5	5151	15436	29.50	718067	2.08E+00	5367	16089	31.51	748101	2.08E+00
mse7	3624	10831	46.44	1094006	2.40E+00	1646	4907	21.16	502604	2.40E+00
	$\varphi = 1$					SR1				
mse3	837	1731	1.68	34841	1.46E+00	765	1541	1.51	31531	1.46E+00
mse5	2568	7680	14.35	359115	2.08E+00	1123	3330	6.83	158131	2.08E+00
mse7	1212	3624	17.15	373014	2.40E+00	800	2347	9.46	249606	2.40E+00

Table 4: Broyden's family for Minimal Surface Equations, choosing DFP, BFGS, $\varphi = 1/2$ and SR1.

5 Conclusion

In this lab we test the Newton method and Quasinewton method in several types of problems. As we see, under appropriate modification, the newton method is powerful with fewer iteration and subiteration. For the brown and dennis function, we found that the quasi newton is less-efficient, since it might be degenerate to the gradient type method. However, we find in the DIE problem the quasi newton is powerful, especially DIE50. The observation shows that the behaviour of Newton type method is heavily dependent on our modification technique. Otherwise it is likely to fail. Also, for the MSE problem, we find that the method will get stuck in some local minima, Newton type method cannot avoid this case.