Применение квазиньютоновских алгоритмов для решения больших задач

Г. И. Забиняко
Институт вычислительной математики и математической геофизики СО РАН, Новосибирск, Россия
Контактный e-mail: zabin@rav.sscc.ru

В статье представлены алгоритмы для решения задач безусловной минимизации на основе квазиньютоновских алгоритмов. В одном алгоритме на итерациях строится матрица, являющаяся приближением к гессиану. В квазиньютоновском алгоритме с ограниченной памятью строятся аппроксимации для обращенной матрицы Гессе, но матрица явно не формируется, а запоминается по ходу итерации некоторое количество векторов, определяющих квазиньютоновские поправки. Проведено сопоставление алгоритмов относительно точности и трудоемкости. Для решения задач большой размерности разработан параллельный вариант квазиньютоновского алгоритма с ограниченной памятью на основе технологии OpenMP. Выполнена проверка эффективности параллельного алгоритма на тестовых задачах большой размерности.

Ключевые слова: квазиньютоновские алгоритмы, квазиньютоновские алгоритмы с ограниченной памятью, безусловная минимизация, технология OpenMP.

Введение

Рассматривается применение квазиньютоновских методов для безусловной минимизации непрерывно дифференцируемых функций. Квазиньютоновские методы основываются на последовательной аппроксимации кривизны нелинейной функции вдоль направлений без явного задания вторых производных. В результате выполнения k шагов метода оцениваются некоторые приближения к матрице вторых производных B_k или к матрице, обратной матрице вторых производных H_k [1]. Наиболее эффективным считается алгоритм Бройдена — Флетчера — Гольдфарба — Шано (BFGS-алгоритм).

Рассмотрим задачу минимизации функции $f(\mathbf{x})$ для $\mathbf{x} \in R^n$ с использованием на итерациях матриц B_k. Пусть $\mathbf{g}_k$ — градиент функции f в точке $\mathbf{x}_k$, $\mathbf{p}_k$ — квазиньютоновское направление поиска, которое определяется из решения системы уравнений $B_k\mathbf{p}_k = -\mathbf{g}_k$. Введем обозначения: $\mathbf{y}_k = \mathbf{g}_{k+1} - \mathbf{g}_k$, $\mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k = \alpha_k\mathbf{p}_k$. Тогда при переходе от k-й к $k+1$-й итерации

$$B_{k+1} = B_k + \frac{1}{(\mathbf{g}_k, \mathbf{p}_k)} \cdot \mathbf{g}_k\mathbf{g}_k^T + \frac{1}{\alpha_k(\mathbf{y}_k, \mathbf{p}_k)} \cdot \mathbf{y}_k\mathbf{y}_k^T,$$

где α_k — шаг вдоль направления $\mathbf{p}_k$. Для того чтобы приближения B_{k+1} правильно отражали кривизну f вдоль определенного направления, необходимо обеспечить соблюдение квазиньютоновского условия $B_{k+1}\mathbf{s}_k = \mathbf{y}_k$ за счет выбора на итерациях подходящего α_k.

Для решения системы уравнений $B_k\mathbf{p}_k = -\mathbf{g}_k$ матрицы представляются в факторизованной форме $B_k = L_k D_k L_k^T$, где L_k — нижнетреугольная, а D_k — диагональная матрицы. Для поддержания на итерациях матриц B_k в факторизованной форме можно воспользоваться, например, процедурой, основанной на методе отражения [2]. В результате возникает возможность получить оценку обусловленности матриц B_k в виде $\mathrm{cond}(B_k) \geq d_{\max}/\min$, где $d_{\max}$ и $d_{\min}$ — максимальный и минимальный элементы D_k. При выполнении итерации можно подправлять d_k так, чтобы обусловленность матриц B_k оставалась приемлемой.

При использовании матриц H_k квазиньютоновское направление находится из матрично-векторного произведения $\mathbf{p}_k = -H_k\mathbf{g}_k$, что требует порядка n^2 операций. Переход от матрицы H_k к H_{k+1} задается соотношениями

$$H_{k+1} = V_k^T H_k V_k + \rho_k \mathbf{s}_k \mathbf{s}_k^T,$$

где

$$V_k = I - \rho_k \mathbf{y}_k \mathbf{s}_k^T, \quad \rho_k = 1/(\mathbf{y}_k, \mathbf{s}_k),$$

а квазиньютоновское условие в этом случае представляется соотношением $H_{k+1}\mathbf{y}_k = \mathbf{s}_k$.

В работе [3] обсуждаются трудности, связанные с использованием метода Ньютона при высокой стоимости вычисления вторых частных производных. Вместо метода Ньютона в [3] предлагается использовать квазиньютоновский метод, который должен удовлетворять следующим требованиям:

1) матрица B_k должна удовлетворять квазиньютоновскому условию;
2) если B_{k-1} симметрична, то и B_k должна быть симметричной;
3) если B_{k-1} положительно определена, то и B_k должна быть положительно определенной;
4) определение нового направления поиска не должно быть слишком трудоемким. Для корректировки матрицы должны использоваться матрицы малого ранга.

Таким образом, в [3] предложен квазиньютоновский алгоритм с ограниченной памятью для оценки матриц B_k. По аналогии с [3] в работе [4] предложен квазиньютоновский алгоритм с ограниченной памятью, в котором вместо построения матриц H_k предлагалось запоминать только некоторое количество векторов, определяющих квазиньютоновские поправки. Эти алгоритмы, получившие наименование L-BFGS, стали интенсивно использоваться для решения больших задач.

Большой статистический материал по применению алгоритмов L-BFGS, в которых оцениваются приближения к H_k, приведен в [5]. Представлены результаты решения большого числа тестовых задач безусловной минимизации и задач с двусторонними ограничениями на переменные. Размерность в некоторых задачах превышает 15 000. По результатам численных экспериментов в [5] делается вывод, что с ростом числа поправочных векторов возрастает эффективность L-BFGS алгоритмов.

В работе [6] рассматриваются вопросы распараллеливания для решения задач безусловной минимизации небольшой размерности, в которых вычисление $f(\mathbf{x})$ трудоемко, а первые и вторые производные недоступны. Предлагается для некоторых i, j оценивать вторые производные, например, следующим образом:

$$\nabla^2 f(\mathbf{x})_{ij} \cong \frac{f(\mathbf{x} + \mu_i\mathbf{e}_i + \alpha_j\mathbf{e}_j) - f(\mathbf{x} + \mu_i\mathbf{e}_i) - f(\mathbf{x} + \alpha_j\mathbf{e}_j) + f(\mathbf{x})}{\mu_i a_j},$$

где $\mu_i = \text{macheps}^{1/2}|\mathbf{x}_i|$ и $\alpha_j = \text{macheps}^{1/4}|\mathbf{x}_j|$. Здесь требуется вычисление множественных оценок $f(\mathbf{x})$ и ее производных одновременно. Отсюда формируются требования к компьютеру, который должен выполнять различные вычисления на различных данных одновременно.

В [7] рассматривается адаптация алгоритма L-BFGS для решения очень больших (миллиарды переменных) задач с использованием распределенных систем вычислительной техники на основе технологии MapReduce.

Квазиньютоновские алгоритмы с ограниченной памятью стали использоваться для решения больших задач стохастической оптимизации. В работе [8] предлагается версия квазиньютоновского алгоритма с ограниченной памятью для стохастической оптимизации и дается обоснование сходимости алгоритма для выпуклой стохастической функции. В общем виде требуется минимизировать $F(w) = E(f(\mathbf{w};\boldsymbol{\xi}))$, где $\boldsymbol{\xi}$ — случайная переменная. Как специальный случай рассматривается $f(\mathbf{w};\boldsymbol{\xi}) = f(\mathbf{w};\mathbf{x}_i,\mathbf{z}_i)$, где переменные $\mathbf{x}_i$ представляются на вход некоторой системы, а в качестве отклика системы служат переменные $\mathbf{z}_i$. В этом случае целевая функция имеет вид

$$F(w) = \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{w};\mathbf{x}_i,\mathbf{z}_i),$$

а градиент

$$\tilde{\nabla}F(w) = \frac{1}{k}\sum_{i\in S} f(\mathbf{w};\mathbf{x}_i,\mathbf{z}_i),$$

где k — некоторый параметр (объем выборки) и $S \subset \{1,2,\ldots,N\}$ — индексы выборочных данных для переменных $\mathbf{x}_i$ и $\mathbf{z}_i$. Итерационный процесс кратко можно представить в следующем виде: $\mathbf{w}^{k+1} = \mathbf{w}^k - \alpha^k H_k \tilde{\nabla} F(\mathbf{w}^k)$.

В работе проводится сопоставление предложенного алгоритма с алгоритмами других авторов. Предшествующая [8] работа [9] посвящена экспериментам по поиску способов оценки кривизны вдоль направлений в стохастическом случае и определению достаточных объемов выборок.

Далее рассмотрены программная реализация и применение алгоритмов BFGS и L-BFGS для решения задач средней размерности. В алгоритме BFGS делаются приближения к гессиану, а в L-BFGS — к матрице, обратной гессиану. Получены экспериментальные результаты, позволяющие сопоставить эти алгоритмы по точности и трудоемкости. Для решения задач большой размерности реализован параллельный алгоритм L-BFGS с использованием технологии OpenMP [10]. На тестовых задачах большой размерности проверена эффективность параллельного алгоритма.

1. Алгоритмы

Рассмотрим программную реализацию квазиньютоновских алгоритмов для решения задач безусловной минимизации или минимизации с учетом двусторонних ограничений на переменные. Приводится два вида алгоритмов, одни из которых строят приближения B_k к гессиану, другие используют квазиньютоновский алгоритм с ограниченной памятью для построения приближения H_k к матрице, обратной гессиану. Авторы первых квазиньютоноских алгоритмов отдавали предпочтение алгоритмам, в которых строились приближения H_k к матрицам, обратным матрицам Гессе. В дальнейшем выяснилось, что построение приближений B_k к гессиану имеет некоторые преимущества [1].

1.1. Алгоритм BFGS

Рассмотрим задачу минимизации функции $f(\mathbf{x})$ для $\mathbf{x} \in R^n$ в области $\alpha_j \le \mathbf{x}_j \le \beta_j$, $j = 1, \ldots, n$, с помощью квазиньютоновского алгоритма BFGS. Алгоритм BFGS с поддержанием на итерациях матриц B_k в факторизованной форме был реализован автором данной работы для решения задач нелинейного программирования с помощью методов модифицированных функций Лагранжа [11]. Матрица B_k сохраняется в двух вещественных массивах с двойной точностью, для L_k отводится $n(n-1)/2$ элементов, а для D_k, соответственно, n. Для перехода на итерациях метода от $L_k D_k L_k^T$ к $L_{k+1} D_{k+1} L_{k+1}^T$ реализована предложенная в [2] экономичная процедура, основанная на методе отражения. В результате переход от одной факторизованной формы к следующей занимает всего $n^2 + O(n)$ операций.

Для определения параметра α_k можно воспользоваться процедурой квадратичной или кубической интерполяции. В качестве основного критерия завершения процесса решения применяется неравенство $||\nabla f(\mathbf{x}_k)||_\infty < \varepsilon$.

Решение задачи с двусторонними ограничениями на переменные получается в результате решения последовательности задач того же вида, что и исходная задача, в которых некоторые переменные принимают постоянные значения $\mathbf{x}_i = \alpha_i$ или $\mathbf{x}_i = \beta_i$ для индексов i, принадлежащих множеству J. Пересмотр набора J производится после решения каждой вспомогательной задачи. Считается, что вспомогательная задача решена, если $||\nabla f(\mathbf{x}_k)||_\infty < \varepsilon_k$, где ε_k, убывая, стремится к заданному значению ε. Во время решения вспомогательной задачи набор J пополняется по мере того, как некоторые переменные достигают своих граничных значений.

1.2. Алгоритм L-BFGS

Рассмотрим реализацию квазиньютоновского алгоритма L-BFGS с ограниченной памятью, в котором на итерациях строятся неявным образом приближения к матрице, обратной гессиану. Здесь также можно учесть двусторонние ограничения на переменные по правилам, рассмотренным выше. Алгоритм определяется заданием на итерациях векторов $\mathbf{s}_k$, $\mathbf{y}_k$ и параметра m, который указывает, какое максимальное число этих векторов может использоваться для определения направления $\mathbf{p}_k$. Для хранения m векторов $\mathbf{s}_k$ и $\mathbf{y}_k$ определяются матрицы $S(n,m)$ и $Y(n,m)$ и вектор $\mathbf{a}(m)$ для запоминания величин $a_i = \dfrac{(\mathbf{s}_i, \mathbf{p}_k)}{(\mathbf{s}_i, \mathbf{y}_i)}$.

Последовательная и параллельная версии алгоритмов задаются следующими шагами:

1. Выбрать начальную точку $\mathbf{x}_0$ и параметр m, положить $m_0 = 0$ и $k = 0$.
2. Вычислить градиент $\mathbf{g}_k$. Проверить, выполняется ли неравенство $||\mathbf{g}_k||_\infty < \varepsilon$; если оно выполнено, то завершить вычисления, иначе положить $\mathbf{p}_k = -\mathbf{g}_k$.
3. Если $m_0 > 0$, то определить направление $\mathbf{p}_k = -H_k \mathbf{g}_k$.
4. Найти $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$. Вычислить градиент $\mathbf{g}_{k+1} = \nabla f(\mathbf{x}_{k+1})$, проверить, выполнено ли неравенство $||\mathbf{g}_{k+1}||_\infty < \varepsilon$; если выполнено, то завершить вычисления.
5. Определить $\mathbf{y}_{k+1} = \mathbf{g}_{k+1} - \mathbf{g}_k$ и $\mathbf{s}_{k+1} = \mathbf{x}_{k+1} - \mathbf{x}_k$.
6. Если $k \ge m$, то в матрицах S и Y заменить векторы-столбцы $\mathbf{s}_{k-m}$ и $\mathbf{y}_{k-m}$ на $\mathbf{s}_{k+1}$ и $\mathbf{y}_{k+1}$; если $m_0 < k$, то положить $m_0 = m_0 + 1$ и дополнить матрицы S и Y новыми столбцами $\mathbf{s}_{k+1}$ и $\mathbf{y}_{k+1}$.
7. Положить $k = k + 1$ и перейти на п. 3.

Приближения к матрице H_k задаются с помощью следующих двух циклов [4]:
$for\ i = k-1, k-2, \ldots, k-m$
$a_i = \dfrac{(\mathbf{s}_i, \mathbf{p}_k)}{(\mathbf{s}_i, \mathbf{y}_i)}$
$\mathbf{p}_k = \mathbf{p}_k - \alpha_i \mathbf{y}_i$
$end(for)$
$\mathbf{p}_k = H_k^0 \mathbf{p}_k$
$for\ i = k-m, k-m+1, \ldots, k-1$
$\beta = \dfrac{(\mathbf{y}_i, \mathbf{p}_k)}{(\mathbf{s}_i, \mathbf{y}_i)}$
$\mathbf{p}_k = \mathbf{p}_k + (\alpha_i - \beta)\mathbf{s}_i$
$end(for)$

Начальное значение вектора $\mathbf{p}_k$ полагается равным $-\nabla f(\mathbf{x}_k)$. Диагональная матрица $H_k^0 = \dfrac{(\mathbf{s}_{k1}, \mathbf{s}_{k-1})}{(\mathbf{y}_{k-1}, \mathbf{y}_{k-1})} I$. После выполнения этих двух циклов имеем некоторое приближение к матрице H_k, обратной гессиану.

В параллельной версии программы были выделены следующие векторные операции: $(\mathbf{x}, \mathbf{x})$; $(\mathbf{x}, \mathbf{y})$; $\mathbf{x} = c\mathbf{x}$ (c — скаляр); $\mathbf{x} = c\mathbf{y}$; $\mathbf{x} = \mathbf{x} + c\mathbf{y}$; $\mathbf{x} = \mathbf{y} + cz$. Каждой из перечисленных векторных операций соответствует процедура, в директиве которой **parallel** необходимо задать опцию **schedule**, указывающую способ распределения итераций цикла между нитями (ядрами) [10]. Некоторые из типов распределения могут зависеть от реализации системы распараллеливания. Проведен эксперимент для выбора типа распределения итераций цикла. В эксперименте вычислялись скалярные произведения двух векторов размерности $n = 3 \cdot 10^7$. Получена статистика времени исполнения, состоящая из десяти наблюдений, для каждого типа распределения итераций в цикле (**static, dynamic, guided, auto**). Оценки математического ожидания $M(t)$ и стандартного отклонения $S(t)$ для разных типов оказались следующими:

static :	$M(t) \cong 0.134$	$S(t) \cong 0.112$
dynamic :	$M(t) \cong 1.195$	$S(t) \cong 0.008$
guided :	$M(t) \cong 0.058$	$S(t) \cong 0.056$
auto :	$M(t) \cong 0.058$	$S(t) \cong 0.053$

Исходя из полученных значений $M(t)$ и $S(t)$ можно предположить, что в качестве типа **auto** в системе принято **guided**. Большие затраты времени для типа **dynamic** объясняются тем, что величина блока по умолчанию принимается равной единице. При использовании типа **dynamic** в процедурах нужно было бы подбирать начальные значения блока в зависимости от размерности задачи. В нашем случае в процедурах вычисления векторных операций принят тип **auto**. Процедуры вычисления $f(\mathbf{x})$ и $\nabla f(\mathbf{x})$ выполняются в последовательном режиме независимо от того, содержат ли они векторные операции.

2. Тестирование

Все расчеты производились в Сибирском суперкомпьютерном центре на сервере HP BL2 × 220c G7 с процессором Intel Xeon X5670 2.93GHz. Сервер содержит 96 вычислительных модулей. Вычислительный модуль (узел) состоит из 12 ядер с общей памятью 24 Гбайта. Программы написаны на языке Фортран, для трансляции использовался

транслятор Intel fortran 14.0.4. Распараллеливание осуществлялось с помощью технологии OpenMP, для вычислений использовался один узел.

Далее приведем результаты решения тестовых задач с помощью алгоритма BFGS. Для линейного поиска вдоль выбранного направления использована процедура кубической интерполяции, в которой поиск α_k завершается после выполнения двух следующих условий:

1) $f(\mathbf{x}+\alpha\mathbf{p}) < f(\mathbf{x}) + 0.0001\cdot\alpha\cdot(\mathbf{p},\nabla f(\mathbf{x}))$,

2) $|(\mathbf{p},\nabla f(\mathbf{x}+\alpha\mathbf{p}))|/|(\mathbf{p},\nabla f(\mathbf{x}))| < 0.5$.

Такого вида приближенное оценивание α_k вошло в практику применения квазиньютоновских алгоритмов достаточно давно [12, 13]. В [12] установлена глобальная сходимость метода с приближенным вычислением α_k для $f(\mathbf{x})$, равномерно выпуклых в допустимой области. Кроме того, для таких $f(\mathbf{x})$ при приближенных вычислениях α_k установлена сверхлинейная скорость сходимости. Второе неравенство можно ослабить, используя вместо коэффициента 0.5 коэффициент 0.9. В случае, когда вычисления $f(\mathbf{x})$ и $\nabla f(\mathbf{x})$ трудоемки, использование коэффициента 0.9 позволяет уменьшить число вычислений $f(\mathbf{x})$ и $\nabla f(\mathbf{x})$, при этом немного увеличивается число итераций, необходимых для получения оптимума.

В табл. 1 приведены результаты решения некоторых тестовых задач [14] с помощью алгоритма BFGS. Во всех задачах использовался критерий окончания решения $||\nabla f(\mathbf{x}_k)||_\infty < 1\cdot 10^{-6}$. Здесь и в последующих таблицах приняты обозначения:

- it — число итераций;
- nfg — число обращений к вычислению $f(\mathbf{x})$ и $\nabla f(\mathbf{x})$;
- t_{sec} — время решения задач в секундах;
- $\delta\mathbf{x}$ — $\max\limits_i |\mathbf{x}_i - \mathbf{x}_i^*|$, где $\mathbf{x}_i^*$ — оптимальное значение i-й переменной, а $\mathbf{x}_i$ — полученное;
- δf — $|f - f^*|/|f^*|$, где f — полученное значение целевой функции, а f^* — ее оптимальное значение;
- $d_{\max}/d_{\min}$ — полученная оценка обусловленности гессиана;
- c* означает, что точные значение $\mathbf{x}_i^*$ неизвестны.

Для задачи 12 получено существенное отклонение по $\mathbf{x}$ от оптимума. В тестовых задачах 5–12 моделируются трудности получения оптимума, связанные с разномасштабностью переменных. Рассмотрим это подробнее на примере задачи 12.

$$f(\mathbf{x}) = 1 + \sum_{i=1}^{n}(i/n)^2\mathbf{x}_i^2 + 0.26\sum_{i=1}^{n-1}\mathbf{x}_i^2(\mathbf{x}_{i+1}+\mathbf{x}_{i+1}^2)^2 +$$

$$+0.26\sum_{i=1}^{2m}\mathbf{x}_i^2\mathbf{x}_{i+m}^4 + 0.26\sum_{i=1}^{m}(i/n)^2\mathbf{x}_i\mathbf{x}_{i+2m}, \quad m = n/3.$$

В результате в решении для одной переменной имеем отклонение $\delta\mathbf{x}_i > 0.1$, для двух переменных — $\delta\mathbf{x}_i > 0.01$, для трех — $\delta\mathbf{x}_i > 0.001$. Аналогично в задачах 9–11 используются множители $(i/n)^2$, а в задачах 5–8 — множители i/n.

В табл. 2 приведены результаты решения тестовых задач с помощью квазиньютоновского алгоритма с ограниченной памятью L-BFGS, в котором строятся приближения H_k к матрицам, обратным гессиану. Расчеты проведены для достаточно больших значений $m = 20$ и $m = 30$.

Из табл. 2 видно, что в задачах 9–12 большие уклонения от решения по $\mathbf{x}$. Были предприняты попытки улучшить решение задачи 12, ужесточив требования к критерию завершения решения. Получены следующие результаты (для $m = 30$):

$\varepsilon = 10^{-7}, \quad it = 3757, \quad nfg = 4413, \quad \delta\mathbf{x} = 0.12;$
$\varepsilon = 10^{-8}, \quad it = 7026, \quad nfg = 8218, \quad \delta\mathbf{x} = 5.4 \cdot 10^{-3};$
$\varepsilon = 10^{-9}, \quad it = 7755, \quad nfg = 9195, \quad \delta\mathbf{x} = 2.6 \cdot 10^{-3}.$

Т а б л и ц а 1. Результаты решения задач алгоритмом BFGS для $n = 3000$

задачи	Название	it	nfg	t_{sec}	$\delta\mathbf{x}$	δf	$d_{\max}/d_{\min}$
1	DIXMAANA	11	13	0.92	$8.7 \cdot 10^{-9}$	$1.1 \cdot 10^{-13}$	1.0
2	DIXMAANB	45	88	4.05	$9.3 \cdot 10^{-8}$	$1.2 \cdot 10^{-13}$	2.3
3	DIXMAANC	66	120	6.02	$1.69 \cdot 10^{-7}$	$1.08 \cdot 10^{-13}$	2.3
4	DIXMAAND	113	261	10.29	$2.9 \cdot 10^{-7}$	$4.0 \cdot 10^{-13}$	2.3
5	DIXMAANE	289	301	26.47	$2.78 \cdot 10^{-4}$	$7.0 \cdot 10^{-10}$	$3.2 \cdot 10^{5}$
6	DIXMAANF	287	304	31.50	$3.3 \cdot 10^{-4}$	$5.1 \cdot 10^{-10}$	$2.3 \cdot 10^{5}$
7	DIXMAANG	476	545	42.90	$2.8 \cdot 10^{-5}$	$1.1 \cdot 10^{-11}$	$1.5 \cdot 10^{5}$
8	DIXMAANH	544	632	49.40	$9.86 \cdot 10^{-5}$	$3.15 \cdot 10^{-10}$	$3.1 \cdot 10^{5}$
9	DIXMAANI	5554	5577	531.90	$1.35 \cdot 10^{-3}$	$2.3 \cdot 10^{-10}$	$6.5 \cdot 10^{8}$
10	DIXMAANJ	2951	2969	271.59	$4.05 \cdot 10^{-2}$	$2.99 \cdot 10^{-8}$	$2.3 \cdot 10^{7}$
11	DIXMAANK	3324	3405	301.30	$3.58 \cdot 10^{-2}$	$9.27 \cdot 10^{-7}$	$1.0 \cdot 10^{7}$
12	DIXMAANL	1840	1927	200.13	0.89	$3.8 \cdot 10^{-7}$	$3.7 \cdot 10^{8}$
13	LIARWHD	12	69	0.60	$6.6 \cdot 10^{-10}$	$2.6 \cdot 10^{-14}$	$2.4 \cdot 10^{6}$
14	EROSEN	12727	25581	1225.14	$8.0 \cdot 10^{-10}$	$2.4 \cdot 10^{-13}$	$3.2 \cdot 10^{4}$
15	TRIDIA	903	3634	86.17	c*	$7.0 \cdot 10^{-16}$	$1.7 \cdot 10^{6}$
16	WOOD	3663	11270	352.29	$8.9 \cdot 10^{-7}$	$1.3 \cdot 10^{-10}$	$9.0 \cdot 10^{4}$

Т а б л и ц а 2. Результаты решения задач алгоритмом L-BFGS для $n = 3000$

	$m = 20$					$m = 30$				
задачи	it	nfg	t_{sec}	$\delta\mathbf{x}$	δf	it	nfg	t_{sec}	$\delta\mathbf{x}$	δf
1	8	11	0.00	$5.7 \cdot 10^{-8}$	$4.9 \cdot 10^{-12}$	8	11	0.0	$5.7 \cdot 10^{-8}$	$4.9 \cdot 10^{-12}$
2	8	12	0.00	$1.4 \cdot 10^{-8}$	$8.4 \cdot 10^{-14}$	8	12	0.0	$1.4 \cdot 10^{-8}$	$8.4 \cdot 10^{-14}$
3	9	13	0.00	$3.0 \cdot 10^{-7}$	$3.6 \cdot 10^{-12}$	9	13	0.0	$3.0 \cdot 10^{-7}$	$3.6 \cdot 10^{-12}$
4	11	16	0.00	$2.5 \cdot 10^{-7}$	$1.7 \cdot 10^{-13}$	11	16	0.1	$2.5 \cdot 10^{-7}$	$1.7 \cdot 10^{-13}$
5	238	272	0.05	$1.5 \cdot 10^{-3}$	$1.3 \cdot 10^{-9}$	243	276	0.09	$2.7 \cdot 10^{-4}$	$3.6 \cdot 10^{-10}$
6	217	252	0.05	$4.8 \cdot 10^{-4}$	$8.9 \cdot 10^{-7}$	194	224	0.04	$1.5 \cdot 10^{-3}$	$1.3 \cdot 10^{-9}$
7	197	237	0.03	$8.6 \cdot 10^{-4}$	$8.6 \cdot 10^{-7}$	168	197	0.08	$4.3 \cdot 10^{-4}$	$5.3 \cdot 10^{-10}$
8	220	259	0.05	$3.8 \cdot 10^{-4}$	$7.6 \cdot 10^{-7}$	172	212	0.07	$1.5 \cdot 10^{-3}$	$1.0 \cdot 10^{-9}$
9	3110	3671	0.58	1.1	$9.7 \cdot 10^{-7}$	3159	3678	0.87	1.1	$1.7 \cdot 10^{-7}$
10	725	664	0.15	1.3	$9.7 \cdot 10^{-7}$	756	905	0.21	1.3	$2.9 \cdot 10^{-7}$
11	686	823	0.12	1.3	$8.0 \cdot 10^{-7}$	731	854	0.25	1.3	$2.9 \cdot 10^{-7}$
12	643	762	0.11	1.3	$9.1 \cdot 10^{-7}$	644	751	0.16	1.3	$3.4 \cdot 10^{-7}$
13	20	33	0.00	$7.6 \cdot 10^{-13}$	$4.2 \cdot 10^{-20}$	20	33	0.00	$9.1 \cdot 10^{-13}$	$4.2 \cdot 10^{-20}$
14	31527	61880	5.82	$1.9 \cdot 10^{-9}$	$6.4 \cdot 10^{-15}$	40044	79490	10.02	$1.0 \cdot 10^{-8}$	$2.3 \cdot 10^{-11}$
15	1185	1408	0.20	c*	$4.8 \cdot 10^{-14}$	1142	1361	0.28	c*	$2.3 \cdot 10^{-13}$
16	326	678	0.05	$4.3 \cdot 10^{-7}$	$1.0 \cdot 10^{-10}$	337	682	0.08	$3.9 \cdot 10^{-7}$	$6.1 \cdot 10^{-9}$

В последнем случае ($\varepsilon = 10^{-9}$) в программе выработался признак аварийного завершения решения задачи — несоответствие в задании функции и ее производных. Для того чтобы в плохо масштабированных задачах получать решение с высокой точностью, необходимо разрабатывать программное обеспечение с учетверенной точностью.

Из сопоставления результатов, приведенных в табл. 1 и 2, можно сделать вывод, что при решении задач небольшой и средней размерности алгоритм BFGS по точности превосходит L-BFGS (где приближения строятся к матрице, обратной гессиану), несмотря на то, что используются достаточно большие значения m. При решении с помощью алгоритма BFGS всех задач, указанных в табл. 1, не требовалось никаких манипуляций с оценками d_i. В программе принято, что если $d_i < 1 \cdot 10^{-5}$, то $d_i := 1 \cdot 10^{-5}$. Аналогично контролируются большие значения d_i: если $d_i > 1 \cdot 10^{9}$, то $d_i := 1 \cdot 10^{9}$. В то же время следует отметить, что в задачах с умеренной обусловленностью гессиана алгоритмы L-BFGS обеспечивают существенную экономию времени.

В табл. 3 и 4 приведены результаты решения тестовых задач большой размерности. Здесь t_1 — время решения задачи с использованием одного ядра, а t_{12} — решение задачи с распараллеливанием с помощью технологии OpenMP на 12 ядрах.

Таблица 3. Решение больших задач в параллельном режиме (12 нитей)

задачи	n	it	nfg	t_{12}	$\delta\mathbf{x}$	δf
5	$3 \cdot 10^7$	1342	1542	8232.81	1.6	$6.4 \cdot 10^{-5}$
6	$3 \cdot 10^7$	236	270	1576.60	1.4	$1.1 \cdot 10^{-5}$
7	$3 \cdot 10^7$	206	246	1471.52	1.4	$1.2 \cdot 10^{-5}$
8	$3 \cdot 10^7$	172	198	1131.44	1.4	$2.0 \cdot 10^{-5}$
9	$3 \cdot 10^7$	2042	2400	11802.73	1.6	$6.6 \cdot 10^{-3}$
10	$3 \cdot 10^7$	423	480	2870.47	1.4	$2.6 \cdot 10^{-5}$
11	$3 \cdot 10^7$	264	327	1955.07	1.4	$9.5 \cdot 10^{-5}$
12	$3 \cdot 10^7$	228	260	1672.56	1.4	$2.8 \cdot 10^{-4}$
14	$3 \cdot 10^4$	400596	795163	2501.47	$1.5 \cdot 10^{-9}$	$1.2 \cdot 10^{-14}$
15	$3 \cdot 10^5$	14552	17015	594.84	c*	$1.6 \cdot 10^{-13}$
16	$3 \cdot 10^7$	304	608	1686.48	$4.3 \cdot 10^{-7}$	$1.2 \cdot 10^{-6}$

Таблица 4. Решение больших задач в последовательном режиме

задачи	n	it	nfg	t_1	$\delta\mathbf{x}$	δf	t_1/t_{12}
5	$3 \cdot 10^7$	1344	1564	30948.64	1.6	$6.3 \cdot 10^{-5}$	3.76
6	$3 \cdot 10^7$	247	286	5718.50	1.4	$8.0 \cdot 10^{-6}$	3.63
7	$3 \cdot 10^7$	202	246	4681.28	1.4	$1.4 \cdot 10^{-5}$	3.18
8	$3 \cdot 10^7$	185	213	4201.06	1.4	$1.2 \cdot 10^{-6}$	3.71
9	$3 \cdot 10^7$	2419	2844	56374.25	1.6	$4.0 \cdot 10^{-3}$	4.78
10	$3 \cdot 10^7$	458	532	10870.17	1.4	$1.9 \cdot 10^{-5}$	3.79
11	$3 \cdot 10^7$	265	314	6077.31	1.4	$1.5 \cdot 10^{-6}$	3.11
12	$3 \cdot 10^7$	265	314	6177.31	1.4	$1.3 \cdot 10^{-4}$	3.69
14	$3 \cdot 10^4$	400636	795254	9319.42	$1.0 \cdot 10^{-9}$	$1.0 \cdot 10^{-14}$	3.72
15	$3 \cdot 10^5$	12067	14015	2529.45	c*	$9.8 \cdot 10^{-14}$	4.25
16	$3 \cdot 10^7$	326	640	7026.08	$2.3 \cdot 10^{-7}$	$4.7 \cdot 10^{-7}$	4.17

Т а б л и ц а 5. Распределение значений t_1/t_{12} для разных n в задаче WOOD

n	t_1/t_{12}	n	t_1/t_{12}	n	t_1/t_{12}	n	t_1/t_{12}	n	t_1/t_{12}	n	t_1/t_{12}
$4 \cdot 10^7$	4.55	$3 \cdot 10^6$	4.41	$3 \cdot 10^5$	4.46	$3 \cdot 10^4$	3.48	$3 \cdot 10^3$	1.05	$1.5 \cdot 10^3$	0.74

Из экспериментов исключены простые задачи 1–4, 13. С другой стороны, решения задач 14 и 15 с увеличением размерности требуют все больших затрат времени, по этой причине в нашем случае размерность уменьшена.

Система распараллеливания требует затрат времени на согласование выполнения операций. Для иллюстрации используем пример учебной программы умножения двух матриц из работы [10]. В результате при расчетах, проведенных на упомянутом сервере, для $n = 4096$ получено более чем семикратное ускорение: $t_1/t_{12} \cong 88.41/12.12 \cong 7.29$. Для матрицы размером $n = 512$ проведено по десять испытаний для оценки среднего и стандартного отклонений времени исполнения программы в параллельном и последовательном режимах. Получены следующие оценки: $M(t_1) \cong 0.048$, $S(t_1) \cong 0.032$ для последовательного и $M(t_{12}) \cong 0.134$, $S(t_{12}) \cong 0.002$ для параллельного исполнения программы. Отсюда следует, что при $n = 512$ в параллельном режиме требуется больше времени по сравнению с последовательным исполнением программы. Этот пример показывает, что и в очень благоприятном случае для применения технологии OpenMP издержки на распределение работ между ядрами существенны.

В табл. 5 приведены значения отношения t_1/t_{12}, полученные при решении тестовой задачи WOOD для разных значений n в параллельном и последовательном режимах.

Анализ результатов, приведенных в табл. 5, позволяет заключить, что распараллеливание позволяет сократить затраты времени в широком диапазоне изменения размерностей. Как видно из данных табл. 2, при $n = 3000$ время решения задачи в последовательном режиме составляет сотые доли секунды, и при этом распараллеливание для таких размерностей неактуально.

Таким образом, в настоящей работе рассмотрены алгоритмы решения задач безусловной минимизации на основе квазиньютоновских алгоритмов. Проведен детальный сравнительный анализ алгоритмов BFGS и L-BFGS с использованием достаточно большого числа поправочных векторов ($m = 20$ и $m = 30$). Для задач большой размерности разработан параллельный алгоритм с использованием технологии OpenMP, проверена эффективность алгоритма на задачах большой размерности.

Список литературы / References

[1] **Гилл Ф., Мюррей У., Райт М.** Практическая оптимизация. М.: Мир, 1985. 509 с.
Gill, P., Murray, W., Wright, M. Practical optimization. Moscow: Mir, 1985. 509 p. (In Russ.)

[2] **Gill, P.E., Murray, W.** Quasi-Newton methods for unconstrained optimization // J. of Math. Anal. and Applications. 1972. Vol. 9, No. 1. P. 91–108.

[3] **Matthies, H., Strang, G.** The solution of nonlinear finite element equations // Intern. J. for Numer. Methods in Eng. 1979. Vol. 14. P. 1613–1626.

[4] **Nocedal, J.** Updating quasi-Newton matrices with limited storage // Math. of Comput. 1980. Vol. 35, No. 151. P. 773–782.

[5] **Zhu, C., Byrd, R.H., Lu, P., Nocedal, J.** Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization // ACM Trans. Math. Softw. 1997. Vol. 23, No. 4. P. 550–560.

[6] **Byrd, R.H., Schnabel, R.B., Shultz, G.A.** Parallel quasi-Newton methods for uncinstrained optimization // Mathematical Programming. 1988. Vol. 42. P. 273–306.

[7] **Chen, W., Wang, Z., Zhou, J.** Large-scale L-BFGS using MapReduce // Advances in Neural Inform. Proc. Sys. 2014. Vol. 27. P. 1332–1340.

[8] **Byrd, R.H., Hansen, S.L., Nocedal, J., Singer, Y.** A stochastic quasi-Newton method for large-scale optimization // SIAM J. on Optimization. 2016. Vol. 26, No. 2. P. 1008–1031.

[9] **Byrd, R.H., Chin, G.M., Nocedal, J., Wu, Y.** Sample size selection in optimization methods for machine learning // Math. Programming. Ser. B. 2012. Vol. 134. P. 127–155.

[10] **Антонов А.С.** Параллельное программирование с использованием технологии OpenMP. М.: Изд-во МГУ, 2009. 76 с.
Antonov, A.S. Parallel programming using OpenMP. Moscow: Izd-vo MGU, 2009. 76 p. (In Russ.)

[11] **Забиняко Г.И.** Программы по нелинейному программированию на основе квазиньютоновского метода. Отчет ВЦ СО АН СССР; № ГР.0186.0125752; Новосибирск: ВЦ СО АН СССР, 1987. 123 с.
Zabinyako, G.I. Nonlinear programming codes based on quasi-Newton method. Otchet VTs SO AN SSSR; No. 0186.0125752. Novosibirsk: VTs SO AN SSSR, 1987. 123 p. (In Russ.)

[12] **Powell, M.J.D.** Some global convergence properties of a variable metric algorithm for minimization without exact line searches // Noulinear Programming / R.W. Cottle, C.E. Lemke (Eds). American Math. Soc., 1976. P. 53–72.

[13] **Shanno, D.F., Phua, K.H.** Minimization of unconstrained multivariate functions // ACM Trans. Math. Softw. 1976. Vol. 2, No. 1. P. 87–94.

[14] **Andrei N.** An unconstrained optimization test functions collection // Advanced Modeling and Optimization. 2008. Vol. 10, No. 1. P. 147–161.

Поступила в редакцию 18 мая 2016 г.,
с доработки — 10 мая 2017 г.

Applications of quasi-Newton algorithms for solving large scale problems

Zabinyako, Gerard I.

Institute of Computational Mathematics and Mathematical Geophysics SB RAS, Novosibirsk, 630090, Russia

Corresponding author: Zabinyako, Gerard I., e-mail: zabin@rav.sscc.ru

In this paper, numerical stability of quasi-Newton algorithms is studied, and their efficiency is estimated. Two types of quasi-Newton algorithms are considered. In the BFGS algorithm, iterative approximations to the Hessian matrix B_k, are constructed. The matrix B_k is maintained in a factored form, $B = L_k D_k L_k^T$, by a procedure based on the reflection method.

In the limited — memory quasi-Newton algorithm, L-BFGS, approximations to the inverse Hessian matrix are constructed. Instead of the inverse Hessian H_k,

L-BFGS stores a few vectors that represent the quasi-Newton updates. The accuracy and efficiency of the BFGS algorithms are compared by solving some test problems. A parallel L-BFGS algorithm based on OpenMP programming interface is developed for solving large scale problems.

The algorithm is tested on problems with a large number of variables. The use of the parallel algorithm makes it possible to significantly reduce the execution time in a wide range of the problem dimensions.

Keywords: quasi-Newton algorithms, quasi-Newton algorithms with limited memory, unconstrained minimization, OpenMP technology.

Received 18 May 2016
Received in revised form 10 May 2017