Блочный метод Холецкого решения системы линейных уравнений

Лапин Александр
lapinra@gmail.com
2011

Теоретическая часть:

Метод Холецкого применяется для решения уравнений вида $Ax = b$, где A - самосопряженная матрица. В дальнейшем рассматривается случай, когда все числа вещественные.

Метод основан на представлении матрицы A в виде $A = R^T DR$, где R - верхнетреугольная матрица с положительными элементами на главной диагонали, а D - диагональная матрица с равными по модулю единице элементами.

Пусть $A = (a_{ij})$, $R = (r_{ij})$, $D = (d_{ij})$, $i, j = 1, \ldots, n$. Тогда, из равенства $A = R^T DR$, имеем:

$$a_{ij} = \sum_{k=1}^{i} r_{ki} d_{kk} r_{kj},$$

Отсюда:

$$r_{ii} d_{ii} r_{ij} = a_{ij} - \sum_{k=1}^{i-1} r_{ki} d_{kk} r_{kj}$$

$$r_{ij} = d_{ii}^{-1} r_{ii}^{-1} (a_{ij} - \sum_{k=1}^{i-1} r_{ki} d_{kk} r_{kj}) \tag{1}$$

$$r_{ii} d_{ii} r_{ii} = a_{ii} - \sum_{k=1}^{i-1} r_{ki} d_{kk} r_{ki} \tag{2}$$

$$r_{ii} = \sqrt{|a_{ii} - \sum_{k=1}^{i-1} r_{ki} d_{kk} r_{ki}|}$$

$$d_{ii} = \operatorname{sgn}(a_{ii} - \sum_{k=1}^{i-1} r_{ki} d_{kk} r_{ki})$$

Построение матрицы R происходит последовательным вычислением её элементов r_{ij} в порядке возрастания i.

Аналогичным образом работает блочный метод Холецкого. Пусть $A = (A_{ij})$, $R = (R_{ij})$, $D = (D_i)$. Тогда аналогами формул (1) и (2) будут являться:

$$R_{ij} = D_i^{-1} (R_{ii}^T)^{-1} \cdot (A_{ij} - \sum_{k=1}^{i-1} R_{ki}^T D_k R_{kj}) \tag{1'}$$

$$R_{ii}^T D_i R_{ii} = A_{ii} - \sum_{k=1}^{i-1} R_{ki}^T D_k R_{ki} \tag{2'}$$

Из (2′) получаем, что R_{ii} и D_i можно найти, применяя метод Холецкого к матрице $(A_{ii} - \sum_{k=1}^{i-1} R_{ki}^T D_k R_{ki})$

Элементы блочной матрицы $R = (R_{ij})$, так же, как и в неблочном методе, вычисляются в порядке возрастания индексов i.

После разложения A в произведение $A = R^T DR$, решаются две системы уравнений $DRy = b$ и $R^T x = y$.

Необходимые операции:

1.Решение системы $DRx = b$ и $R^T x = b$, где R - верхнетреугольная матрица

Из $(1')$:

2. Умножение матриц вида AB, AD, ADB, где D - диагональная матрица
3. Вычитание одной матрицы из другой
4. Нахождение обратной матрицы для нижнетреугольной матрицы

Из $(2')$:

5. Применение неблочного метода Холецкого к матрице

Организация хранения матриц:

Исходную матрицу A можно хранить в виде верхнетреугольной матрицы (хранить только элементы a_{ij}, $i \leqslant j$). Элементы матрицы R можно записывать на соответсвующие места матрицы A. Матрицу D можно хранить в виде вектора $D = (d_{11}, \ldots, d_{nn})$.

Особенности реализации:

Для эффективной работы алгоритма необходимо организовать наиболее подходящим образом хранение блоков матрицы, порядок выполнения операций в вышеуказанных функциях и сам порядок выполнения этих функций. Рассмотрим особенности реализаций каждой из них:

1. Решение системы $DRx = b$ и $R^T x = b$, где R - верхнетреугольная матрица.
   а) При решении таких систем нет необходимости изменять матрицу R^T (R)
   б) Матрицы стоит хранить так, чтобы при решении системы внутренний цикл пробегал по строке
2. Умножение матриц вида AB, AD, ADB, где D - диагональная матрица
   Переумножать матрицы надо так, чтобы внутренний цикл пробегал по строке
   Например:

```
for k = 1, .., m
  for i = 1, .., n
    for j = 1, .., m
      c[i, j] += a[i, k] * b[k, j]
```

3. Вычитание одной матрицы из другой
   Операции надо производить по строкам
4. Нахождение обратной матрицы для нижнетреугольной матрицы
   Чтобы внутренний цикл бежал по строке, можно вычитать из i-ой строки вышестоящие строки
5. Применение неблочного метода Холецкого к матрице
   Так как i-ая строка R зависит только от вышестоящих, то внутренний цикл в формуле (1) можно сделать по индексу j

Алгоритм блочного алгоритма Холецкого:

1. Пусть $i-1$ строка блочной матриц R и D уже посчитаны.
2. Переберем k от 1 до $(i-1)$, j от i до количества блоков в строке.
   Для фиксированных k, j:

$$A' = R_{ki}^T,$$

$$B' = R_{kj},$$

$$C' = R_{ki}^T D R_{kj},$$

$$A_{ij} = A_{ij} - C'$$

3. Вычисляем R_{ii} и D_i неблочным методом Холецкого от A_{ii}.
4. Ищем обратную матрицу к $R_{ii} D_i$:

$$A' = R_{ii} D_i$$

$$B' = (A')^{-1}$$

5. Умножаем i-ую строку матрицы A, начиная с $(i+1)$-го элеметна, на B'.

6. i-ые строки блочных матриц R и D посчитаны, переходим к $(i+1)$-ой.
7. После вычисления R и D, решаем $DRy = b$.
8. Решаем $R^T x = y$.

Параллельный алгоритм:

Последовательный блочный алгоритм выглядит следующим образом:

```
for i = 1, .., n
{
..for j = i, .., n
....for k = 1, .., i-1
......Вычесть из блока A[i][j] произведение A[k][i]*A[k][j]

..Посчитать A[i][i], D[i]
..Посчитать обратный блок к A[i][i]

..for j = i+1, .. n
....Умножить блок A[i][j] на обратный к A[i][i]
}
```

Пусть теперь есть p потоков. На i-ом шаге внешнего цикла, разобьем множество всех блоков, которые на этом шагу должны быть посчитаны, на p частей. И пусть в кадом потоке считаются только соответственные блоки A_{ij}. Тогда, единственным местом, к которому обращаются все потоки, будет i-ый столбец блоков **(A)**, включая блок A_{ii} **(B)**.

Коллизии в случае **(A)** можно значительно уменьшить путем запуска цикла по k в каждом потоке с разными начальными значениями. В случае **(B)** можно для каждого потока скопировать обратную к A_{ii} в отдельный блок (для каждого потока свой).

Чтобы перейти к умножению на обратный блок, необходимо что бы к этому моменту был посчитан блок A_{ii}. Для этого можно сделать перед вторым циклом по j точку синхронизации. Понятно, что чтобы перейти к следующей строке блоков (к $i+1$-ому шагу внешнего цикла), также необходима точка синхронизации.

Пусть t - номер текущего потока, p - количество потоков, тогда получаем следующую схему параллельного алгоритма для одного потока:

```
for i = 1, .., n
{
..for j = i + t, .., n, на каждом шаге j += p
....for k = (i + p - 1) / p * t, .., (i + p - 1) / p * t - 1, на каждом шаге k = k%i + 1
......Вычесть из блока A[i][j] произведение A[k][i]*A[k][j]

..Если текущий поток тот, в котором считалось A[i][i]
..{
....Посчитать A[i][i], D[i]
....Посчитать обратный блок к A[i][i]
..}

..Точка синхронизации

..Скопировать обратную к A[i][i]

..for j = i + 1 + t, .. n, на каждом шаге j += p
....Умножить блок A[i][j] на обратный к A[i][i]

..Точка синхронизации
```

```
}
```