

Решение системы линейных уравнений методом Холецкого

Лобанова Валерия, группа 310

Содержание

| | | |
|----------|---|-----------|
| 1 | Введение | 3 |
| 1.1 | Постановка задачи. Разложение Холецкого | 3 |
| 1.2 | Оценка сложности алгоритма построения верхнетреугольной матрицы в разложении Холецкого | 4 |
| 2 | Блочный метод Холецкого | 5 |
| 2.1 | Описание блочного разложения Холецкого | 5 |
| 2.2 | Описание решения систем $R^T y = b$ и $DRx = y$ | 5 |
| 2.3 | Хранение матриц | 6 |
| 2.4 | Оценка сложности в алгоритме построения верхнетреугольной матрицы в блочном разложении Холецкого | 7 |
| 3 | Параллельный блочный метод Холецкого | 9 |
| 3.1 | Описание параллельного блочного разложения Холецкого | 9 |
| 3.2 | Описание параллельного решения систем $R^T y = b$ и $DRx = y$ | 10 |
| 3.3 | Оценка числа точек синхронизаций | 12 |
| 3.4 | Оценка сложности в алгоритме построения верхнетреугольной матрицы в параллельном блочном разложении Холецкого | 13 |
| 4 | MPI блочный метод Холецкого | 14 |
| 4.1 | Описание MPI блочного разложения Холецкого | 14 |
| 4.2 | Описание MPI-решения систем $R^T y = b$ и $DRx = y$ | 15 |
| 4.3 | Оценка сложности MPI-реализации решения СЛУ методом Холецкого | 16 |

1 Введение

1.1 Постановка задачи. Разложение Холецкого

Задача. Найти решение системы линейных уравнений $Ax = b$, где A — симметричная вещественнозначная матрицы размера $n \times n$, b — известный вектор размера n , x — неизвестный вектор.

Идея решения. Поиск решения будет осуществляться с помощью разложения Холецкого матрицы $A = R^T DR$, где

R — верхнетреугольная матрица,

D — диагональная матрица с 1 или -1 на диагонали.

Найдем такое y , что $R^T y = b$ и затем из условия $DRx = y$ найдем x . \square

Теорема. Пусть матрица A — самосопряженная и все ее угловые миноры отличны от нуля. Тогда существует матрица $R = (r_{ij}) \in RT(n)$ с вещественными положительными элементами на главной диагонали и диагональная матрица D с вещественными равными по модулю единице диагональными элементами такие, что $A = R^T DR$.

Решение задачи. Применим точечный метод Холецкого для поиска матрицы R . Элементы d_{ii}, r_{ii}, r_{ij} могут быть вычислены по следующим формулам:

$$d_{ii} = \operatorname{sgn}\left(a_{ii} - \sum_{k=1}^{i-1} |r_{ki}|^2 d_{kk}\right), \quad i = 1, \dots, n, \quad (1)$$

$$r_{ii} = \sqrt{\left|a_{ii} - \sum_{k=1}^{i-1} |r_{ki}|^2 d_{kk}\right|}, \quad i = 1, \dots, n,$$

$$r_{ij} = (r_{ii} d_{ii})^{-1} \left(a_{ij} - \sum_{k=1}^{i-1} r_{ki} d_{kk} r_{kj}\right), \quad i < j, \quad i, j = 1, \dots, n,$$

\square

1.2 Оценка сложности алгоритма построения верхне-треугольной матрицы в разложении Холецкого

Из формул (1) следует, что для вычисления элемента d_{ii} , $i = 1, \dots, n$ требуется $2(i - 1)$ операций (умножение на d_{ii} за операцию не считаем). Следовательно, вычисление всех элементов матрицы D требует

$$\sum_{i=1}^n 2(i - 1) = n(n - 1) = O(n^2), \quad n \rightarrow \infty \quad \text{операций.}$$

Для вычисления элемента r_{ii} требуется $2(i - 1) + 1 = 2i - 1$ операций (учитываем 1 операцию извлечения корня).

При фиксированном $i = 1, \dots, n$ вычисление элементов r_{ij} для всех $j = i + 1, \dots, n$ по формулам (1) требует

$$\sum_{j=i+1}^n (2i - 1) = (n - i)(2i - 1) \quad \text{операций.}$$

Таким образом нахождение матрицы R требует

$$\sum_{i=1}^n (n - i)(2i - 1) + (2i - 1) = \frac{2n^2 + 3n + 1}{6} = \frac{n^3}{3} + O(n^2), \quad n \rightarrow \infty \quad \text{операций.}$$

2 Блочный метод Холецкого

2.1 Описание блочного разложения Холецкого

Разобьем матрицу A на блоки (A_{ij}) размера $m \times m$, где $m < n$ и в случае когда $m \nmid n \Rightarrow n = m * k + l, l \neq 0$, крайние блоки могут иметь размеры $m \times l, l \times m, l \times l$. Матрицы R и D можно также искать в виде блочных матриц.

Из формул $A = R^T D R$ и (1) ясно, что формулы для нахождения блоков матрицы R имеют вид:

$$R_{ii}^T D_i R_{ii} = A_{ii} - \sum_{j=1}^{i-1} R_{ji}^T D_j R_{ji}, \quad i = 1, \dots, k, \quad (2)$$

$$R_{is}^T D_i R_{is} = A_{is} - \sum_{j=1}^{i-1} R_{ji}^T D_j R_{js}, \quad i, s = 1, \dots, k, \quad i < s$$

$$R_{is} = D_i (R_{ii}^T)^{-1} (A_{is} - \sum_{j=1}^{i-1} R_{ji}^T D_j R_{js}), \quad i, s = 1, \dots, k, \quad i < s \quad (3)$$

Тем самым сначала получаются блоки R_{ii} и D_i точечным разложением Холецкого из формулы (2), а затем используя (3) вычисляются R_{is} для $s = i + 1, \dots, n$.

2.2 Описание решения систем $R^T y = b$ и $DRx = y$

Для решения $R^T y = b$ представляем, что R^T на самом деле не транспонированная и лежит в памяти как R , но работаем с ней как с транспонированной. Тогда получаем следующий алгоритм:

```
for (j = 0; j < block_lim; j++)
{
    sum = 0;
    for (i = 0; i < j; i++)
        sum += Y[i] * R_{ij};

    Y[j] = (B[j] - sum) / R_{jj};
}
```

Для решения $DRx = y$ важно учитывать, что умножение на матрицу D можно производить после подсчета суммы.

```

for (i = block_lim - 1; i >= 0; i--)
{
    sum = 0;
    for (j = block_lim - 1; j > i; j--)
        sum += X[j] * R_{ij};

    X[i] = D[i] * (Y[i] - sum * D[i]) / R_{ii};
}

```

2.3 Хранение матриц

Так как матрица A симметричная, то логично хранить не всю матрицу, а только верхнюю ее часть над главной диагональю и саму диагональ.

$$a_{00} = a[0]; \quad a_{11} = a[n]; \quad a_{22} = a[n + (n - 1)]; \dots$$

$$a_{ii} = a\left[\sum_{j=0}^{i-1} (n - j)\right] = a[i * (2 * n - i + 1) / 2];$$

$$a_{is} = a\left[\sum_{j=0}^{i-1} (n - j)\right] = a[i * (2 * n - i + 1) / 2 + (s - i)], \quad i \leq s$$

У матрицы D хранить нужно только диагональ в массиве длины n .

При вычислении матрицы R элементы R_{is} можно записывать сразу на место A_{is} , так как A_{is} больше не будет использоваться.

2.4 Оценка сложности в алгоритме построения верхнетреугольной матрицы в блочном разложении Холецкого

Если известно количество операций в случае $l = 0$, то количество операций при $l \neq 0$ можно оценить сверху, заменив в оценке k на $k + 1$.

Начнём с оценки количества операций для $R_{ji}^T D_j R_{js}$, чтобы не путаться в индексах рассмотрим это произведение как $R^T D R$, тогда

$$(R^T D R)_{ij} = \sum_{k=1}^{\min(i,j)} r_{ki} d_k r_{kj}$$

здесь $\min(i, j) - 1$ аддитивных и $\min(i, j)$ мультипликативных операций, то есть всего $2\min(i, j) - 1$ операций для одного элемента (по аналогии с неблочным методом умножение на d_i за операцию не считаем).

Тогда для вычисления $R_{ji}^T D_j R_{js}$ требуется

$$\begin{aligned} \sum_{i=1}^m \sum_{j=1}^m (2\min(i, j) - 1) &= \sum_{i=1}^m \sum_{j=1}^i (2j - 1) + \sum_{i=1}^m \sum_{j=i+1}^m (2i - 1) = \\ &= \frac{m(m+1)(2m+1)}{6} + \frac{m(2m^2 - 3m + 1)}{6} = \frac{m(2m^2 + 1)}{3} \end{aligned}$$

Обозначим как $Mult(m) = (2m^3 + m)/3$.

Для вычисления $A_{is} - \sum_{k=1}^{i-1} R_{ji}^T D_j R_{js}$ требуется:

$$H(i, m) = (i - 1)(Mult(m) + m^2) = (i - 1)(2m^3 + 3m^2 + m)/3 \quad \text{операций.}$$

Сложность разложения Холецкого $Chol(m) = m^3/3$.

Сложность вычисления блока R_{ii} и D_i : $H(i, m) + Chol(m)$

Сложность вычисления всех диагональных блоков:

$$S_1(n, m, k) = \sum_{i=1}^k (H(i, m) + Chol(m)) = n(2mn + 3n + k - 3m - 1)/6$$

Умножение на треугольную матрицу требует $Y(m) = m^3$ операций. Здесь имеется ввиду умножение на $(R_{ii}^T)^{-1}$ в формуле (3). Подсчёт обратной к R_{ii}^T учтём позже, так как это вычисление выполняется 1 раз при подсчете всей строки.

Итак, сложность вычисления недиагонального блока R_{ij} :

$$R(i, m) = H(i, m) + Y(m) = (i - 1)(2m^3 + 3m^2 + m)/3 + m^3$$

Сложность вычисления всех недиагональных блоков R :

$$S_2(n, m, k) = \sum_{i=1}^k \sum_{s=i+1}^k R(i, m) = n(k - 1)(2mn + 3n + k + 5m^2 - 6m - 2)/18$$

Для вычисления строки - R_{ij} при фиксированном i требуется $(R_{ii}^T)^{-1}$, следовательно нужно $(k - 1)$ раз найти обратную матрицу за $S_3(n, m) = (k - 1)Chol(m) = (k - 1)m^3/3$ операций.

Итак, нахождение всех блоков R_{is} требует

$$\begin{aligned} S(n, m) &= S_1 + S_2 + S_3 = \frac{n(2mn + 3n + k - 3m - 1)}{6} + \\ &+ \frac{n(k - 1)(2mn + 3n + k + 5m^2 - 6m - 2)}{18} + \frac{(k - 1)m^3}{3} = \\ &= \frac{n(2n^2 + m^2 + 9mn - 3m - 1 + 3nk + k^2)}{18} - \frac{m^3}{3} = \\ &= \boxed{\frac{n^3}{9} + \frac{nm^2}{18} + \frac{n^2m}{2} - \frac{nm}{6} - \frac{n}{18} + \frac{n^3}{6m} + \frac{n^3}{18m^2} - \frac{m^3}{3}} \\ S(n, n) &= \frac{n^3}{9} + \frac{n^3}{18} + \frac{n^3}{2} - \frac{n^2}{6} - \frac{n}{18} + \frac{n^2}{6} + \frac{n}{18} - \frac{n^3}{3} = \frac{n^3}{3} \\ S(n, 1) &= \frac{n^3}{9} + \frac{n}{18} + \frac{n^2}{3} - \frac{n}{6} - \frac{n}{18} + \frac{n^3}{6} + \frac{n^3}{18} - \frac{1}{3} = \frac{n^3}{3} + \frac{n^2}{3} - \frac{n}{6} \\ &\boxed{S(n, n) = \frac{n^3}{3} \quad S(n, 1) = \frac{n^3}{3} + O(n^2), \quad n \rightarrow \infty} \end{aligned}$$

3 Параллельный блочный метод Холецкого

3.1 Описание параллельного блочного разложения Холецкого

Из формул (2) и (3) видно, что после вычисления диагонального блока и его обращения все блоки вне диагонали можно искать в любом порядке в контексте текущей строки.

Пусть p — количество потоков. Принадлежность столбца потоку определяется так: s -ый поток обрабатывает столбцы с номерами $s + zp$, где $z \in \mathbb{Z}$.

Для вычисления R_{is} требуются блоки из столбцов i и s , находящиеся строго в предыдущих строках. При чем s -ый столбец считается «своим», а i -ый принадлежит другому потоку, поэтому предлагается каждому потоку иметь копию столбца i .

Барьеры используются для корректного копирования данных из общей памяти (матрицы A), соответственно до и после этой операции.

Таким образом, имеем следующий параллельный алгоритм:

```
for (i = 0; i < block_lim; i++)
{
    Barrier
    get diag block R_{ii} and i-th column
    Barrier

    calculate diag block R_{ii}
    inverse diag block R_{ii}

    if (i % th_p == th_i)
        put diag block R_{ii} and block D_{i}

    if (i % th_p < th_i)
        j = i - (i % p) + th_i;
    else
        j = i - (i % p) + p + th_i;

    for (; j < block_lim; j += th_p)
```

```

    {
        get block R_{ij}
        calculate block R_{ij}
        put block R_{ij}
    }
}

```

3.2 Описание параллельного решения систем $R^T y = b$ и $DRx = y$

Описанный выше линейный алгоритм решения системы $R^T y = b$ имеет особенность - для вычисления $y[i]$ требуются готовые значения всех предыдущих компонент, то есть $y[j]$, $j = 0 \dots i - 1$, что для параллельной реализации недопустимо.

По аналогии с параллельным блочным разложением определим принадлежность данных потоку. Пусть p — количество потоков, i -ый блок вектора b принадлежит потоку с номером $i \% p$.

Заметим, что в СЛУ матрица R транспонирована, поэтому при умножении на блоки R всегда подразумевается операция с транспонированным блоком. К тому же, обратный ход Гаусса выполняется с вычитанием не строк, а столбцов.

Получается следующий параллельный алгоритм:

```

for (i = 0; i < block_lim; i++)
{
    Barrier
    get diag block R_{ii} and block B_{i}
    Barrier

    inverse diag block R_{ii}
    calculate inv_R_{ii}^T * B_{i} -> B_diff

    if (i % th_p < th_i)
        s = i - (i % th_p) + th_i;
    else
        s = i - (i % th_p) + th_p + th_i;

    for (; s < block_lim; s += th_p)

```

```

{
    get block R_{is}
    get block B_{s}
    calculate B_{s} - R_{is}^T * (B_diff)
    put block B_{s}
}

if (i % th_p == th_i)
    put changed block B_{i}
}

```

Описанный выше линейный алгоритм решения системы $DRx = y$ по той же причине не можем использовать.

Перед началом решения параллельно умножит вектор y на $D^{-1} = D$. На i -ой итерации обратного хода потоки будут вычислять изменение текущего блока $Y[i]$ (за счет условного вычитания строк) и складывать результат в разделяемый буферный блочный вектор S . После поток с номером $i \% p$ соберет эти изменения и применит к блоку $Y[i]$.

Таким образом, получаем следующий параллельный алгоритм:

```

for (i = 0; i < block_lim; i++)
{
    if (i % th_p < th_i)
        s = i + (i % th_p) - th_i;
    else
        s = i + (i % th_p) - th_p - th_i;

    bzero B_diff
    for (; s >= 0; s -= th_p)
    {
        get block R_{is}
        get block B_{s}
        calculate - R_{is} * B_{s} -> B_diff
    }
    put_block B_diff -> S
    Barrier

    if (i % th_p == th_i)
    {

```

```

    get diag block R_{ii}
    get block B_{i} -> B_diff
    inverse diag block R_{ii}

    /* get total B_diff */
    for (j = 0; j < block_size; j ++)
    {
        sum = 0;
        for (s = 0; s < th_p; s++)
            sum += S[j + s * block_size];
        B_diff[j] += sum;
    }

    calculate inv_R_{ii} * B_{i}
    put changed block B_{i}
}
Barrier
}

```

3.3 Оценка числа точек синхронизаций

Считаем, что $n = mk + l$, $l = 0$.

Разложение матрицы методом Холецкого требует $2k$ барьеров. При решении систем $R^T y = b$ и $DRx = y$ используется по $2k$ точек синхронизации. Причем каждый поток использует только свою память.

Таким образом, в параллельном алгоритме $3k$ точек синхронизаций.

3.4 Оценка сложности в алгоритме построения верхнетреугольной матрицы в параллельном блочном разложении Холецкого

Из приведенного выше алгоритма видно, что все потоки вычисляют совпадающие диагональные блоки и обратные к ним, поэтому

$$S_1(n, m, p) = S_1(n, m) \quad \text{и} \quad S_3(n, m, p) = S_3(n, m)$$

Эти формулы из оценки блочного алгоритма.

Вычисление внедиагональных блоков происходит полностью параллельно, поэтому $S_2(n, m, p) = S_2(n, m)/p$

Итак,

$$\begin{aligned} S(n, m, p) &= S_1 + \frac{S_2}{p} + S_3 = \frac{n(2mn + 3n + k - 3m - 1)}{6} + \\ &+ \frac{n(k-1)(2mn + 3n + k + 5m^2 - 6m - 2)}{18p} + \frac{(k-1)m^3}{3} = \\ &= \frac{n^3}{9} + \frac{n^2m}{6p} + \frac{n^2m}{3} + \frac{nm^2}{3} - \frac{5nm^2}{18p} - \frac{m^3}{3} + \frac{n^2}{2} - \frac{n^2}{2p} - \frac{nm}{2} + \frac{nm}{3p} - \\ &\quad - \frac{n}{6} + \frac{n}{9p} + \frac{n^3}{6pm} + \frac{n^3}{18pm^2} + \frac{n^2}{6m} - \frac{n^2}{6pm} \end{aligned}$$

Причем $S(n, m, 1) = S(n, m)$

4 MPI блочный метод Холецкого

4.1 Описание MPI блочного разложения Холецкого

Отличие MPI-реализации от параллельной блочной, использующей потоки, заключается в том, что из одного процесса невозможно иметь доступ ко всей матрице.

Процесс с номером my_rank считает «своими» блок-столбцы с номером i_glob , для которых выполнено $my_rank = i_glob \% comm_size$, и выделяет для них память.

Для вычисления R_{is} (для матрицы R всегда имеется ввиду глобальная нумерация) требуются блоки из столбцов i и s , находящиеся строго в предыдущих строках. Причем s -ый столбец считается «своим», а i -ый принадлежит другому процессу, поэтому «хозяин» рассылает каждому процессу столбец i .

Таким образом, имеем следующий алгоритм:

```
for (i_glob = 0; i_glob < block_lim; i_glob++)
{
    if (i_glob % comm_size == my_rank)
        copy i_glob-th column to buf_column

    MPI_Bcast (buf_column, m * m * i_glob + m * (m + 1) / 2,
               MPI_DOUBLE, i_glob % comm_size, MPI_COMM_WORLD);

    diag block R_{ii} := buf_column + m * m * i_glob
    calculate diag block R_{ii}
    inverse diag block R_{ii}

    if (i_glob % comm_size == my_rank)
        put diag block R_{ii} and block D_{ii}

    for (j_glob = i_glob + 1; j < block_lim; j++)
        if (j_glob % comm_size == my_rank)
            calculate block R_{ij}
}
```

4.2 Описание MPI-решения систем $R^T y = b$ и $DRx = y$

По аналогии с MPI блочным разложением определим принадлежность данных процессу: i -ый блок вектора b принадлежит потоку с номером $i \% comm_size$.

Если рассматривать MPI-модернизацию параллельного алгоритма решения системы $R^T y = b$, то она требует большого количества папарных пересылок изменного вектора b . Чтобы минимизировать количество пересылок используем следующий MPI-алгоритм:

```
for (i_glob = 0; i_glob < block_lim; i_glob++)
{
    column_lim := m or n%m for last block
    if (i_glob % comm_size == my_rank)
    {
        sum = 0;
        for (j_glob = 0; j_glob < i_glob; j_glob++)
            sum += Y[j_glob] * R_{ji}

        Y[i_glob] = (B[i_glob] - sum) / R_{ii}
    }
    MPI_Bcast (Y[i_glob], column_lim, MPI_DOUBLE,
               i_glob % commSize, MPI_COMM_WORLD);
}
```

MPI-алгоритм решения системы $DRx = y$:

```
for (i_glob = block_lim - 1; i_glob >= 0; i_glob--)
{
    copy i_glob-th string to buf_str (MPI_Gatherv)
    if (my_rank == main_rank)
    {
        sum = 0;
        for (j_glob = block_lim - 1; j_glob > i_glob; j_glob--)
            sum += X[j_glob] * buf_string[j_glob];

        X[i_glob] = D[i_glob]
                    * (Y[i_glob] - sum * D[i_glob]) / buf_str[i_glob];
    }
}
```

4.3 Оценка сложности MPI-реализации решения СЛУ методом Холецкого

Оценим число пересылок между процессами и объем пересылаемых данных. Будем считать, что $l = 0$, где $n = m * k + l$.

- В разложении:
 - k пересылок - каждая блочная строка будет разослана всем,
 - общий объем пересылаемых данных будет $n * (n + 1) / 2$, то есть равен размеру матрицы.
- При вычислении вектора y :
 - k пересылок - каждый новый посчитанный блок будет разослан всем,
 - общий объем пересылаемых данных будет n , то есть равен размеру вектора.
- При вычислении вектора x :
 - n пересылок - на каждой итерации вычисления компоненты вектора будет собираться строка матрицы,
 - общий объем пересылаемых данных будет $n * (n + 1) / 2$, то есть равен размеру матрицы.
- Всего:
 - $2k + n$ пересылок,
 - общий объем пересылаемых данных равен $n * (n + 2)$.

Оценим объем памяти расходуемый на все процессы:

- хранение матрицы A — $n * (n + 1) / 2$,
- хранение матрицы D — n ,
- хранение векторов b, x, y — $3 * n * comm_size$,
причем вектор y используется как буфер при вычислении y ,
- память для буферной блочной строки — $n * m * comm_size$.

Итого, $0,5 * n * (n + 3) + comm_size * n * (3 + m)$.