

Отчет по заданию
"Решение системы линейных уравнений
методом вращений"

Сибгатуллин Артур, 310 учебная группа

27 сентября, 2020

Содержание

1	Введение	3
1.1	Алгоритм метода вращений	3
2	Точечный метод вращения	5
2.1	Реализация(Псевдокод)	5
2.2	Применимость	5
2.3	Обратный ход метода Гаусса	5
2.4	Оценка числа арифметических операций	5
2.5	Организация хранения в памяти	6
3	Блочный метод вращения	7
3.1	Обратный ход метода Гаусса	7
3.2	Оценка числа арифметических операций	8
3.3	Организация хранения в памяти	8

1 Введение

В курсе предмета "Работа на ЭВМ и программирование" перед нами была поставлена задача: решить систему линейных уравнений методом "**Вращений (Гивенса)**". Пусть система уравнений $Ax = b$ задана следующими элементами:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}, x = \begin{pmatrix} x_{11} \\ x_{21} \\ \vdots \\ x_{n1} \end{pmatrix}, b = \begin{pmatrix} b_{11} \\ b_{21} \\ \vdots \\ b_{n1} \end{pmatrix}$$

Где матрица A —невырожденная, x —вектор столбец искоемых неизвестных, b —вектор столбец свободных членов. Необходимо найти x .

1.1 Алгоритм метода вращений

Лемма 1: $\forall r = (x_1, x_2)^T \in \mathbb{R}^2$, где $r \neq 0$ существует матрица T :

$$T = \begin{pmatrix} \cos(\varphi) & -\sin(\varphi) \\ \sin(\varphi) & \cos(\varphi) \end{pmatrix} \text{ и } Tr = \|r\|(1, 0)^T \text{ где } \|r\| = \sqrt{x_1^2 + x_2^2}$$

□ Просто предъявим такую матрицу: $\cos(\varphi) = \frac{x_1}{\sqrt{x_1^2 + x_2^2}}$, $\sin(\varphi) = -\frac{x_2}{\sqrt{x_1^2 + x_2^2}}$ ■

Как можно заметить матрица T это матрица вращения на угол φ в \mathbb{R} относительно оси x_1 , поэтому метод называется методом вращения.

Таким же образом мы можем привести вектор-столбцы любого размера к виду $(\|x_1\|, \dots, 0)^T$, например $(x_1, x_2, x_3)^T$ приводим к виду $(\|x_1\|, 0, 0)^T$, сначала преобразовав к виду $(\|x_1\|, 0, x_3)^T$, а затем к $(\|x_1\|, 0, 0)^T$. Формальное доказательство этого факта приведено в учебнике К.Ю.Богачева «Практикум на ЭВМ. Методы решения линейных систем и нахождения собственных значений», стр.43 Лемма 3.

Сам алгоритм заключается в приведении матрицы A к верхнетреугольному виду следующим образом. Обозначим первый столбец матрицы A как

$$a^1 = \begin{pmatrix} a_{1,1} \\ a_{2,1} \\ \vdots \\ a_{n,1} \end{pmatrix}$$

Тогда \exists набор матриц $T_{1,2}, \dots, T_{1,n}$, таких что $T_{1,2} * \dots * T_{1,n} * a^1 = \|a^1\| * e_1$, (e_1, \dots, e_n) - стандартный базис в \mathbb{R}^n . Умножим систему $Ax = b$ на $T_{1,2} * \dots * T_{1,n}$

слева. Получим новую систему $A^{(1)}x = b^{(1)}$ эквивалентную исходной, где

$$A^{(1)} = T_{1,2} * \dots * T_{1,n} * A = \begin{pmatrix} \|a_1\| & c_{1,2} & \dots & c_{1,n} \\ 0 & a_{2,2}^{(1)} & \dots & a_{2,n}^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & a_{n,2}^{(1)} & \dots & a_{n,n}^{(1)} \end{pmatrix}, b^{(1)} = T_{1,2} * \dots * T_{1,n} * \begin{pmatrix} b_{1,1} \\ b_{2,1} \\ \vdots \\ b_{n,1} \end{pmatrix}$$

Далее применим процесс для подматрицы $(a_{i,j}^{(1)})_{i,j=2\dots n}$ (1). После применения данного процесса ко всем подматрицам вида (1) мы получим матрицу

$$R = \begin{pmatrix} \|a_1\| & c_{1,2} & c_{1,3} & \dots & c_{1,n-2} & c_{1,n-1} & c_{1,n} \\ & \|a_1^{(1)}\| & c_{2,3} & \dots & c_{2,n-2} & c_{2,n-1} & c_{2,n} \\ & & \|a_1^{(2)}\| & \dots & c_{3,n-2} & c_{3,n-1} & c_{3,n} \\ & & & \ddots & \vdots & \vdots & \vdots \\ & & & & \|a_1^{(n-3)}\| & c_{n-2,n-1} & c_{n-2,n} \\ & & & & & \|a_1^{(n-2)}\| & c_{n-1,n} \\ & & & & & & a_{n,n}^{(n-1)} \end{pmatrix},$$

и столбец свободных коэффициентов $y = b^{(n-1)} = \prod_{i=n-1}^{n-1} \prod_{j=n}^{i+1} T_{ij} b$, где $\prod_{j=n}^{i+1}$ означает, что сомножители берутся в порядке $n, \dots, i+1$

Система $Rx = y$, где R - верхнетреугольная матрица, решается обратным ходом метода Гаусса.

2 Точечный метод вращения

2.1 Реализация(Псевдокод)

```
for i=1 to n do
  for j = i+1 to n do
    cos(phi) = A[i][i] / norm(j)
    sin(phi) = -A[j][i] / norm(j)
    for k = i to n do
      A[i][k] = A[k][j] * cos(phi) - sin(phi) * A[j][k]
      A[j][k] = A[k][j] * cos(phi) + sin(phi) * A[j][k]
    end for
    b[i] = b[i] * cos(phi) - sin(phi) * b[j]
    b[j] = b[i] * sin(phi) + b[j] * cos(phi)
  if A[i][i] < EPS
    return ERROR: System is inconsistent
  end for
end for
Back-Substitution_procedure()
```

2.2 Применимость

Матрица A должна быть не вырожденной.

2.3 Обратный ход метода Гаусса

Обратный ход метода Гаусса заключается в нахождении x из верхнетреугольной матрицы, следующим образом:

$$x_n = y_n, x_i = y_i - \sum_{j=i+1}^n c_{i,j} x_j \text{ где } i = n-1, \dots, 1$$

2.4 Оценка числа арифметических операций

Посчитаем сложность алгоритма для k -го шага а затем просуммируем по $k = 1, \dots, n-1$

- Для вычисления матриц $T_{k,k+1}, \dots, T_{k,n}$ необходимо $n-k$ аддитивных, $4*(n-k)$ мультипликативных и $n-k$ операций извлечения корня.
- Для вычисления компонент k, \dots, n k -го столбца матрицы $A^{(k)}$, которые мы считаем по формуле $\|a_1^{(k-1)}\|e_1^{(n-k+1)}$ требуется $(n-k+1)$ мультипликативных операций, $(n-k)$ аддитивных операций и 1 операция извлечения корня

- Так как в алгоритме мы применяем умножение $n - k$ матриц вращения на подматрицу $(a_{i,j}^{(k-1)})_{i=1,\dots,n,j=k+1\dots n}$, то на это требуется $4(n - k)^2$ мультипликативных и $2(n - k)^2$ аддитивных операций
- Для вычисления столбца y требуется $4(n - k)$ мультипликативных и $n - k$ аддитивных операций
- Для вычисления обратного хода метода Гаусса требуется $\frac{n(n-1)}{n}$ аддитивных и $\frac{n(n-1)}{n}$ операций

Итого:

- Аддитивных операций $\sum_{k=1}^{n-1}(4(n - k)^2 + 9(n - k) + 1) = \frac{4n^3}{3} + O(n^2)$ при $n \rightarrow \inf$
- Мультипликативных операций $\sum_{k=1}^{n-1}(2(n - k)^2 + 3(n - k)) = \frac{2n^3}{3} + O(n^2)$ при $n \rightarrow \inf$
- Операций извлечения корня $\sum_{k=1}^{n-1}(n - k + 1) = O(n^2)$ при $n \rightarrow \inf$

Как можно заметить, метод вращений требует в 2 раза больше мультипликативных и в 4 раза больше аддитивных операций, чем метод Гаусса.

2.5 Организация хранения в памяти

Матрица A хранится в памяти в виде массива размером n^2 , решение x и столбец свободных членов b хранятся в виде массивов длины n . Таким потреблением памяти $n^2 + 2n + O(1)$. Так как совершается $O(n^3)$ операций с $O(n^2)$ элементами, то возникает необходимость в кэшировании часто используемых данных, для оптимизации работы с памятью.

3 Блочный метод вращения

Представим нашу матрицу A в блочном виде:

$$A = \begin{pmatrix} A_{11} & A_{12} & \dots & A_{1k} \\ A_{21} & A_{22} & \dots & A_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ A_{k1} & A_{k2} & \dots & A_{kk} \end{pmatrix}$$

Где размер каждого блока m , тогда без ограничения общности $k = n/m$ - на-цело.

На первом шаге алгоритма преобразуем блок A_{11} к верхнетреугольному виду с помощью обыкновенного метода вращений. Как побочный продукт алгоритма у нас останутся вычисленными матрицы поворота: $T_{11} = \prod_{i=n-1}^{n-1} \prod_{j=n}^{i+1} T_{ij}$. Применим их к строке блоков $A_{12} \dots A_{1k}$ и к блоку B_1 - вектор столбец B в блочном виде. Получим матрицу вида:

$$A = \begin{pmatrix} A_{11}^{(1)} & A_{12}^{(1)} & \dots & A_{1k}^{(1)} \\ A_{21} & A_{22} & \dots & A_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ A_{k1} & A_{k2} & \dots & A_{kk} \end{pmatrix}$$

Затем обнуляем столбец блоков $A_{21} \dots A_{k1}$, матрицами вращений, строящихся из диагональных элементов $a_{ii} \in A_{11}^{(1)}$ и элементов из столбца $a_{2i}, \dots, a_{ji} \in A_{21}^{(1)}$. Эти же матрицы поворота используем для преобразования строк блоков $A_{22}, \dots, A_{2k}, \dots, A_{k2}, \dots, A_{kk}$. Имеем:

$$A = \begin{pmatrix} A_{11}^{(1,2)} & A_{12}^{(1,2)} & \dots & A_{1k}^{(1,2)} \\ 0 & A_{22}^{(2)} & \dots & A_{2k}^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & A_{k2}^{(2)} & \dots & A_{kk}^{(2)} \end{pmatrix}$$

Теперь применяем тот же алгоритм для подматрицы с начальным элементом $A_{22}^{(2)}$

В конечном итоге получаем верхнетреугольную матрицу, которую можно решить обратным ходом метода Гаусса.

3.1 Обратный ход метода Гаусса

Обратный ход метода Гаусса заключается в нахождении x из верхнетреугольной матрицы, следующим образом:

$$x_n = y_n, x_i = y_i - \sum_{j=i+1}^n c_{i,j} x_j \text{ где } i = n-1, \dots, 1$$

3.2 Оценка числа арифметических операций

m - размер блока, k - количество блоков в строке (столбце).

- Для приведения блока A_{ii} к верхнетреугольному виду необходимо $2m^3$ арифметических операций. Таких преобразований будет k штук.
- Для умножения строки блоков $A_{i2} \dots A_{ik}$ на матрицы поворота необходимо $(k - t)m(m - 1)6m$ арифметических операций.
- Для обнуления столбца блоков необходимо $6m^2(k - t) \sum_{i=1}^m (m - i)$

Итого: $2nm^2 + \sum_{t=1}^{n/m} (6m^2(m - 1)(k - t) + 6m^2(k - t) \sum_{i=1}^m (m - i)) =$

- $m = 1$:
- $m = n$:

3.3 Организация хранения в памяти

Матрица A хранится в памяти в виде массива размером n^2 , решение x и столбец свободных членов b хранятся в виде массивов длины n . Также необходимо хранить 3 массива размером m^2 в кэш-памяти.

Подпрограмма загрузки (выгрузки) блока матрицы в кэш-память:

```
void get_block (double *A, int matrix_size, double *Block,
               int block_size, int i_, int j_)
{
    int dev = matrix_size / block_size;
    int rem_of_dev = matrix_size % block_size;
    int block_m_size = (i_ < dev ? block_size : rem_of_dev);
    int block_n_size = (j_ < dev ? block_size : rem_of_dev);

    double *Block_i = Block;
    double *Ai = A + (i_ * matrix_size + j_) * block_size;

    for (int i = 0; i < block_m_size; i++)
    {
        for (int j = 0; j < block_n_size; j++)
            Block_i[j] = Ai[j];

        Ai += matrix_size;
        Block_i += block_n_size;
    }
}

void set_block (double *A, int matrix_size, double *Block,
```



```

        int block_size, int i_, int j_)
{
    int dev = matrix_size / block_size;
    int rem_of_dev = matrix_size % block_size;
    int block_m_size = (i_ < dev ? block_size : rem_of_dev);
    int block_n_size = (j_ < dev ? block_size : rem_of_dev);

    double *Block_i = Block;
    double *Ai = A + (i_ * matrix_size + j_) * block_size;

    for (int i = 0; i < block_m_size; i++)
    {
        for (int j = 0; j < block_n_size; j++)
            Ai[j] = Block_i[j];

        Ai += matrix_size;
        Block_i += block_n_size;
    }
}

```