CFD COURSE 2024

18 января 2024 г.

Содержание

1 Что происходит?

Цель курса - освоить метод сеток для решения дифференциальных уравнений в частных производных эллиптического типа.

2 Рекомендации

Для того, чтобы сдать этот предмет на 4 и выше необходимо просто посещать все пары. Даже, если вы не будете ничего понимать, но будете переписывать код, раз за разом будет становиться понятно, что происходит.

3 План решения

1. Составить конечно-разностную схему для дифференциального уравнения (СЛАУ $A\vec{u} = \vec{b}$);
2. Определиться с тем, как эффективно хранить ненулевые элементы матрицы A;
3. Выбрать метод, для решение СЛАУ $A\vec{u} = \vec{b}$ для нахождения $\vec{u}$;

4 Дифференциальное уравнение Пуассона 1D

Решим методом сеток задачу Дирихле для уравнения Пуассона на одномерной сетке. Эта задача ставиться следующим образом.

Найти непрерывную функцию $u(x)$, удовлетворяющую внутри прямоугольной области $\Omega = \{(x)|0 \leq x \leq a\}$ уравнению Пуассона:

Дифференциальное уравнение Пуассона 1D

$$\frac{d^2u}{dx^2} = f(x)$$

и принимающую на границе области Ω заданные значения (условия Дирихле), т. е.

Зададим на отрезке $[a, b]$ равномерную координатную сетку с шагом δ:

$$x_i = i \cdot h_x,$$

Граничные условия первого рода (условия Дирихле):

$$u(0) = g_1, u(a) = g_2, 0 \leq x \leq a,$$

Граничные условия второго рода (условия Неймана) для рассматриваемой задачи могут быть представлены в виде:

$$\frac{du}{dx}|_0 = g_1,$$
$$\frac{du}{dx}|_a = g_2$$

Проводя дискретизацию граничных условий Неймана на сетке, получим:

$$\frac{u_2 - u_1}{h_x} = g_1,$$
$$\frac{u_n - u_{n-1}}{h_x} = g_2.$$

Проводя дискретизацию уравнения для внутренних точек сетки, получим:

$$\frac{u_{i+1} - 2u_i + u_{i-1}}{h_x^2} = f_i,\ i = 2 \ldots (N-1),$$

5 Дифференциальное уравнение Пуассона 2D

Решим методом сеток задачу Дирихле для уравнения Пуассона в прямоугольной области. Эта задача ставиться следующим образом.

Найти непрерывную функцию $u(x, y)$, удовлетворяющую внутри прямоугольной области $\Omega = \{(x, y)|0 \leq x \leq a, 0 \leq y \leq b\}$ уравнению Пуассона:

Дифференциальное уравнение Пуассона

$$\frac{\partial^2u}{\partial x^2} + \frac{\partial^2u}{\partial y^2} = f(x, y)$$

и принимающую на границе области Ω заданные значения, т. е.

$$u(0, y) = \Gamma_1(y), u(a, y) = \Gamma_3(y), 0 \leq y \leq b,$$
$$u(x, 0) = \Gamma_4(x), u(x, b) = \Gamma_3(x), 0 \leq x \leq a$$

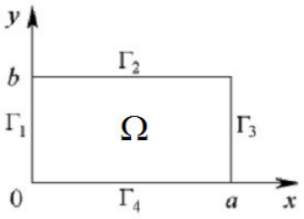
где $\Gamma_1, \Gamma_2, \Gamma_3, \Gamma_4$ - это заданные функции.

Считаем, что $u(x, y)$ - непрерывна на границе области Ω, т. е. $\Gamma_1(0) = \Gamma_4(0)$, $\Gamma_1(b) = \Gamma_2(0)$, $\Gamma_3(0) = \Gamma_4(a)$, $\Gamma_3(b) = \Gamma_2(a)$. Выбрав шаги h_x, h_y по x и y соответственно строим сетку

$$x_i = i \cdot h_x, i = \overline{0, N}, y_j = j \cdot h_y, i = \overline{0, M}$$

где N, M - это количество узлов.

Уравнение является уравнением эллиптического типа. Решение таких уравнений можно получить, используя явную разностную схему:



Конечно-разностная схема

$$\begin{cases} u_{i,j} = \frac{1}{4}\left(u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - h^2 \cdot f_{i,j}\right), f_{i,j} = f(x_i, y_j), \\ u_{i,m} = \Gamma_2(x_i), u_{i,0} = \Gamma_4(x_i), u_{0,j} = \Gamma_1(y_j), u_{n,j} = \Gamma_3(y_j), \\ i = \overline{1, N-1}, j = \overline{1, M-1} \end{cases}$$

Численное решение задачи Дирихле для уравнения Пуассона в прямоугольнике состоит в нахождении приближенных значений $u_{i,j}$ функции $u(x, y)$ во внутренних узлах сетки. Для определения величин $u_{i,j}$ требуется решить СЛАУ (1)

Эту систему будем решать итерационным методом Гаусса-Зейделя, который состоит в последовательности итераций вида

$$\begin{cases} u_{i,j}^{(k+1)} = \frac{1}{4}\left(u_{i+1,j}^{(k+1)} + u_{i-1,j}^{(k+1)} + u_{i,j+1}^{k+1} + u_{i,j-1}^{(k+1)} - h^2 \cdot f_{i,j}\right), \\ f_{i,j} = f(x_i, y_j) \end{cases}$$

где верхним индексом k обозначен номер итерации. При $k \to \infty$ последовательность $u_{i,j}^k$ сходится к точному решению системы (33). В качестве условия окончания итерационного процесса можно принять

Критерий останова:

$$\max \left| u_{i,j}^{(k+1)} - u_{i,j}^{(k)} \right| < \epsilon, \ i \in [1; N-1], j \in [1; M-1]$$

Однако этот критерий недостаточно надежен, поскольку итерационный процесс сходится медленно. На практике применяют более надежный критерий

$$\max_{i,j} \left| u_{i,j}^{(k+1)} - u_{i,j}^{(k)} \right| < \epsilon(1 - \nu)$$

где $\nu = \frac{\max_{ij} |u_{i,j}^{(k+1)} - u_{i,j}^{(k)}|}{\max_{ij} |u_{i,j}^{(k)} - u_{i,j}^{(k-1)}|}$.

Таким образом, погрешность приближенного решения, полученного методом сеток, складывается из двух погрешностей:

- погрешности аппроксимации дифференциального уравнения разностными уравнениями;
- погрешности, возникающей в результате приближенного решения системы разностных уравнений (33).

Известно, описанная здесь разностная схема обладает свойством устойчивости и сходимости. Устойчивость схемы означает, что малые изменения в начальных данных приводят к малым изменениям решения разностной задачи. Только такие схемы имеет смысл применять в реальных вычислениях. Сходимость схемы означает, при стремлении шага сетки к нулю (т. е. при $h \to 0$) решение разностной задачи стремится в некотором смысле к решению исходной задачи. Таким образом, выбрав достаточно малый шаг h, можно как угодно точно решить исходную задачу.

6 Форматы хранения матрицы

Перед тем, как решать СЛАУ, полученной в конечно-разностной схеме, есть возможность сократить сложность по времени и по памяти за счет того, что в матрице будет находится множество нулей и нет смысла хранить их в памяти, вместо этого можно сосредоточиться на ненулевых элементах. Вот тут и возникают разные методы записи ненулевых элементов.

Все форматы делятся на те, которые легко итерировать (1 группа) и те, которые легко собирать (2 группа). Ниже, для каждого метода хранения матрицы будет показана работа основных функций:

1. LinearIndex - конвертация индексов матрицы A в индекс data;
2. SetValue - установить значение элемента матрицы;
3. Mult - произведение матрицы на вектор;
4. MultRow - произведение строки матрицы на вектор;
5. Diagonal - получение диагональных элементов матрицы;

6.1 DenseMatrix (итерация по строкам)

$$A = \begin{bmatrix} 2 & 0 & 1 \\ 6 & 3 & 0 \\ 4 & 7 & 9 \\ 0 & 1 & 6 \\ 0 & 3 & 5 \end{bmatrix}, data = [2, 0, 1, 6, 3, 0, 4, 7, 9, 0, 1, 6, 0, 3, 5]$$

```
// Пример: linear_index[2, 1] -> 7
size_t linear_index(size_t irow, size_t icol) const{
    return irow * _n_cols + icol;
};


// Пример: linear_index[2, 1] -> 7, data[linear_index[2, 1]] = 7
void set_value(size_t irow, size_t icol, double value) override{
    size_t k = linear_index(irow, icol);
    _data[k] = value;
}


// Пример: хотим умножить irow = 2 строчку матрицы A на x = {2, 5, 1}
double mult_row(size_t irow, const std::vector<double>& x) const override{
    double ret = 0;
    const double* it = &_data[irow * _n_rows];
    for (size_t irow = 0; irow < _n_rows; ++irow){
        ret += (*it) * x[irow];
        ++it;
    }
    return ret;
}


// Пример: хотим перемножить все строчки матрицы A на x = {2, 5, 1}
std::vector<double> mult(const std::vector<double>& x) const override{
    // Инициализация нулевого N - мерного вектора
    std::vector<double> ret(_n_rows, 0);
    for (size_t i = 0; i < _n_rows; ++i){
        ret[i] += mult_row(i, x);
    }
    return ret;
}


std::vector<double> diagonal() const override{
    // Инициализация нулевого N - мерного вектора
    std::vector<double> ret(_n_rows, 0);
    // Для каждой строки цепляем элемент, в котором i = j
    for (size_t i =0; i<_n_rows; ++i){
        size_t k = linear_index(i, i);
        ret[i] = _data[k];
    }
    return ret;
}
```

6.2 DenseMatrix (итерация по столбцам)

$$A = \begin{bmatrix} 2 & 0 & 1 \\ 6 & 3 & 0 \\ 4 & 7 & 9 \\ 0 & 1 & 6 \\ 0 & 3 & 5 \end{bmatrix}, data = [2, 6, 4, 0, 0, 0, 3, 7, 1, 3, 1, 0, 9, 6, 5]$$

```
// Пример: linear_index[2, 1] -> 1 * 5 + 2 = 7
size_t linear_index(size_t icol, size_t irow) const{
    return icol * _n_rows + irow;
};


// Пример: linear_index[2, 1] -> 7, data[linear_index[2, 1]] = 7
void set_value(size_t icol, size_t irow, double value) override{
    size_t k = linear_index(icol, irow);
    _data[k] = value;
}


double mult_row(size_t i_row, const std::vector<double>& x) const override{
    double ret = 0;
    // В каждой колонке должны взять по 1 элементу и сложить к ret
    for (size_t icol = 0; icol < _n_cols; ++icol){
        // Каждый раз совершаем скачок
        ret += _data[i_row + icol * _n_rows] * x[icol];
    }
    return ret;
}


std::vector<double> mult(const std::vector<double>& x) const override{
    std::vector<double> ret(_n_cols, 0);
    for (size_t i = 0; i < _n_rows; ++i){
        ret[i] += mult_row(i, x);
    }
    return ret;
}


std::vector<double> diagonal() const override{
    std::vector<double> ret(_n_cols, 0);
    for (size_t i = 0; i < _n_cols; ++i){
        size_t k = linear_index(i, i);
        ret[i] = _data[k];
    }
    return ret;
}
```

Возникает вопрос: какой из DenseMatrix лучше подходит? Который берет все элементы строки разом или тот, который осуществляет скачок, чтобы дойти до нужного элемента.

Лучше использовать DenseMatrix по строкам. И, конечно, при обоих способах количество итераций не изменится.

6.3 CsrMatrix (Compressed Sparse Row)

Compressed sparse row - сжатая разреженная матрица.

Он похож на метод *COO(Coordinate list)*, но сжимает индексы строк, отсюда и название. Этот формат обеспечивает быстрый доступ к строкам и матрично-векторное умножение.

Вместо того, чтобы заполнять *vals, cols, rows* явно:

$$\begin{bmatrix} 1 & 0 & 2 \\ 3 & 0 & 4 \\ 0 & 7 & 0 \end{bmatrix} = \begin{array}{c} vals = \{1, 2, 3, 4, 7\}, \\ cols = \{0, 2, 0, 2, 1\}, \\ rows = \{0, 0, 1, 1, 2\} \end{array}$$

Можем немного схитрить и найти индексы первых элементов в *vals* в каждой строке и таким образом заполнить массив *rows*. Но есть одно уточнение, последний элемент *rows* - это *length(Vals)*. Это нужно для умножения матрицы на вектор.

$$\begin{bmatrix} 1 & 0 & 2 \\ 3 & 0 & 4 \\ 0 & 7 & 0 \end{bmatrix} = \begin{array}{c} vals = \{1, 2, 3, 4, 7\}, \\ cols = \{0, 2, 0, 2, 1\}, \\ rows = \{0, 2, 4, 5\} \end{array}$$

Тут могут возникнуть вопросы: "А есть ли метод CSC - Compressed Sparse Column? Есть ли в нем отличая от CSR? Если существует, то можно одновременно и по строкам и по стролбцам сократить количество элементов?".

Ответ: Да, такой метод есть, можно в вики посмотреть, отличий в нем нет, кроме того, что аналогичным способом заполняются не строки, а колонки, и обход идет сверху-вниз слева-направо.

$$\begin{bmatrix} 1 & 0 & 2 \\ 3 & 0 & 4 \\ 0 & 7 & 0 \end{bmatrix} = \begin{array}{c} vals = \{1, 2, 3, 4, 7\}, \\ cols = \{0, 2, 0, 2, 1\}, \\ rows = \{0, 2, 4, 5\} \end{array}$$

Насчет последнего вопроса, покажу на примере, что произойдет, если взять первые значения по столбцам.

$$\begin{bmatrix} 1 & 0 & 2 \\ 3 & 0 & 4 \\ 0 & 7 & 0 \end{bmatrix} = \begin{array}{c} vals = \{1, 2, 3, 4, 7\}, \\ cols = \{0, 2, 1, ?\}, \\ rows = \{?\} \end{array}$$

Вот тут становится понятно, что из-за привычного обхода матрицы слева-направо сверху-вниз для заполнения массива *vals* не получится вместе соединить оба метода, потому что в таком случае сбивается монотонное возрастание индексов при заполнение *cols*.

Поэтому нужно выбирать что-то одно.

```
// Массивы для реализации методов
private:
std::vector<double> _vals;  //  []
std::vector<double> _cols;  //  []
std::vector<double> _addr;  //  []


// + дополнительный метод value
double value(size_t irow, size_t icol) const{
    // Мы будем вызывать метод value по всему rows,
    // поэтому нужно вставить последний элемент
    size_t ibegin = _addr[irow];
    size_t iend = _addr[irow+1];
    auto it = std::lower_bound(_cols.begin() + ibegin,
            _cols.begin() + iend, icol);
    if (it != _cols.begin() + iend && *it == icol){
        size_t a = it - _cols.begin();
        return _vals[a];
    } else {
        return 0;
    }
}


// Установить значение элемента матрицы
void set_value(size_t irow, size_t icol, double value) override{
    size_t ibegin = _addr[irow];
    size_t iend = _addr[irow + 1];
    auto cols_begin = _cols.begin() + ibegin;
    auto cols_end = _cols.begin() + iend;
    auto it = std::lower_bound(cols_begin, cols_end, icol);
    size_t a = it - _cols.begin();
    if (it != cols_end && *it == icol){
        _vals[a] = value;
    } else {
        for (size_t i=irow+1; i<_addr.size(); ++i) _addr[i] += 1;
        _cols.insert(_cols.begin() + a, icol);
        _vals.insert(_vals.begin() + a, value);
    }
}
```

6.4 TripletSparseMatrix

Пример:

$$\begin{bmatrix} 0 & 0 & 5 & 2 \\ 0 & 0 & 0 & 6 \\ 9 & 0 & 0 & 0 \\ 7 & 0 & 0 & 0 \end{bmatrix} \begin{array}{c} vals = \{5, 2, 6, 9, 7\}, \\ = rows = \{0, 0, 1, 2, 3\} \\ cols = \{2, 3, 3, 0, 0\} \end{array}$$

```
    const size_t _n_rows;
std::vector<int> _rows;
std::vector<int> _cols;
std::vector<double> _vals;

double value(size_t irow, size_t icol) const{
int a = find_by_row_col(irow, icol);
if (a >= 0){
return _vals[a];
} else {
return 0.0;
}
}

    // returns >= 0 if [irow, icol] found, else -1
    int find_by_row_col(size_t irow, size_t icol) const{
        for (size_t a=0; a<_rows.size(); ++a){
            if (_rows[a] == (int)irow
                && _cols[a] == (int)icol){
                return (int)a;
            }
        }
        return -1;
    }

    // Установить значение элемента матрицы
    void set_value(size_t irow, size_t icol, double value) override{
        int addr = find_by_row_col(irow, icol);
        if (addr >= 0){
            _vals[addr] = value;
        } else {
            _rows.push_back(irow);
            _cols.push_back(icol);
            _vals.push_back(value);
        }
    }
```

Кроме вышеперечисленных существуют куча других методах, которые пригодятся при определенных ситуациях, полезно знать об их существовании, чтобы вовремя воспользоваться. тык

6.5 Сравнение методов хранения матриц

	COO	DOK	LIL	CSR	CSC	BSR	DIA	Dense
Indexing	-	+	+	+	+	-	-	+
Write-only	+	+	+	-	-	-	-	+
Read-only	-	-	-	+	+	+	+	+
Low memory	+	-	-	+	+	+	+	-

7 Методы решения СЛАУ

7.1 Jacobi–Gauss

Формулировка

Дана квадратная система из N линейных уравнений с неизвестным $\mathbf{u}$:

$$A\mathbf{u} = \mathbf{f} \Leftrightarrow \sum_{i=0} A_{ij}u_j = f_i, i = \overline{0, N-1}$$

где:

$$A = \begin{bmatrix} a_{11} & a_{12}\dots & a_{1n} \\ \dots & \dots & \dots \\ a_{n1} & a_{n2}\dots & a_{nn} \end{bmatrix}, \mathbf{u} = \begin{bmatrix} u_1 \\ \dots \\ u_n \end{bmatrix}, \mathbf{f} = \begin{bmatrix} f_1 \\ \dots \\ f_n \end{bmatrix}$$

Причем матрица A представима в виде суммы: $A = D+L+U$ (LU - разложение), где D - матрица с диагональными элементами, L, U - матрицы с нижнее и верхне треугольными элементами соответственно.

$$D = \begin{bmatrix} A_{11} & 0 & \dots & 0 \\ 0 & A_{22} & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & A_{nn} \end{bmatrix}, L+U = \begin{bmatrix} 0 & A_{12} & \dots & A_{1n} \\ A_{21} & 0 & \dots & A_{2n} \\ \dots & \dots & \dots & \dots \\ A_{n1} & A_{n2} & \dots & 0 \end{bmatrix},$$

$$(D+L+U)\mathbf{u} = D\mathbf{u} + L\mathbf{u} + U\mathbf{u} = \mathbf{f},$$

$$D\mathbf{u} = \mathbf{f} - (L+U)\mathbf{u},$$

$$\mathbf{u} = D^{-1}\left(\mathbf{f} - (L+U)\mathbf{u}\right)$$

итеративно схема выглядит сдедующим образом:

$$\mathbf{u}^{(k+1)} = D^{-1}\left(\mathbf{f} - (L+U)\mathbf{u}^{(k)}\right)$$

Алгоритм Якоби-Гаусса

- Задать начальное приближение;
- Совершить итерацию

$$u_i^{(k+1)} = \frac{1}{A_{i,i}}\left(f_i - \sum_{j\neq i} A_{i,j}u_j^{(k)}\right)$$

Необходимое условие сходимости: $\Rightarrow$
Спектральный радиус ρ итерационной матрицы меньше 1:

$$\rho\left(D^{-1}(L+U)\right) < 1,$$

$$\rho(A) = \max\{|\lambda_1|, \dots |\lambda_n|\},$$

где λ_i - собственные числа матрицы A.
Достаточное условие сходимости: $\Leftarrow$
Матрица A строго или неприводимо доминирует по диагонали. Строгое доминирование по диагонали строки означает:

$$|A_{i,i}| > \sum_{j\neq i} |A_{i,j}|.$$

Теорема. Метод Якоби-Гаусса сходится тогда и только тогда $\Leftrightarrow$, когда все значения λ_i, определяемые уравнением

$$\det \begin{bmatrix} \lambda A_{1,1} & A_{1,2} & A_{1,3} & \dots & A_{1,N} \\ A_{2,1} & \lambda A_{2,2} & A_{2,3} & \dots & A_{2,N} \\ \dots & \dots & \dots & \dots & \dots \\ A_{N,1} & A_{N,2} & A_{N,3} & \dots & \lambda A_{N,N} \end{bmatrix} = 0,$$

были по модулю меньше единицы: $|\lambda_i| < 1 \ \ \forall i$.
Метод Якоби иногда сходится, даже если эти условия не выполняются.

Доказательство. Рассмотрим матрицу перехода:

$$R = -D^{-1}(L + U).$$

По теореме о необходимом и достаточном условии сходимости (МПИ) метод сходиться в том случае, если

$$\rho(R) < 1.$$

Вычислим спектральный радиус матрицы R. Для этого найдем собственные значения матрицы R.

$$R\vec{e} = \lambda\vec{e} \Rightarrow \det(R - \lambda E) = 0,$$

$$\det(-D^{-1}(L + U) - \lambda E) = \det(-D^{-1}[(L + U)\lambda D])$$

$$\det(-D^{-1}[(L + U) + \lambda D]) = \det(-D^{-1})\det(L + U + \lambda D) = 0,$$

$$\det(L + U + \lambda D) = 0.$$

Таким образом, если все значения λ_i, определяемые уравнением $\det(L+U+\lambda D) = 0$ по модулю меньше единицы, то $\forall i|\lambda_i| < 1 \leftrightarrow \rho(R) < 1$. □

Недостатки метода:

- Медленная сходимость;
- Зависимость от начального приближения;

7.2 Gauss–Seidel

Формулировка

Дана квадратная система из N линейных уравнений с неизвестным $\mathbf{u}$:

$$A\mathbf{u} = \mathbf{f} \Leftrightarrow \sum_{i=0} A_{ij}u_j = f_i, i = \overline{0, N-1}$$

где:

$$A = \begin{bmatrix} a_{11} & a_{12}\dots & a_{1n} \\ \dots & \dots & \dots \\ a_{n1} & a_{n2}\dots & a_{nn} \end{bmatrix}, \mathbf{u} = \begin{bmatrix} u_1 \\ \dots \\ u_n \end{bmatrix}, \mathbf{f} = \begin{bmatrix} f_1 \\ \dots \\ f_n \end{bmatrix}$$

Причем матрица A представима в виде суммы: $A = L_* + U$.

$$A = \underbrace{\begin{bmatrix} A_{11} & 0 & \dots & 0 \\ A_{21} & A_{22} & \dots & 0 \\ \dots & \dots & \dots & \dots \\ A_{n1} & A_{n2} & \dots & A_{nn} \end{bmatrix}}_{L_*} + \underbrace{\begin{bmatrix} 0 & A_{12} & \dots & A_{1n} \\ 0 & 0 & \dots & A_{2n} \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 0 \end{bmatrix}}_{U},$$

$$A\mathbf{u} = (L_* + U)\mathbf{u} = \mathbf{f},$$

$$L_*\mathbf{u} = \mathbf{f} - U\mathbf{u}.$$

Метод Гаусса–Зайделя теперь решает левую часть этого выражения для $\mathbf{u}$, используя предыдущее значение в правой части. Итеративно схема выглядит сдедующим образом:

$$\mathbf{u}^{(k+1)} = L_*^{-1}\left(\mathbf{f} - U\mathbf{u}^{(k)}\right)$$

Алгоритм Gauss–Seidel (Частный случай метода SOR при $\omega = 1$)

- Задать начальное приближение;
- Совершить итерацию

$$u_i^{(k+1)} = \frac{1}{A_{i,i}} \left(f_i - \sum_{j=1}^{i-1} A_{i,j} u_j^{(k+1)} - \sum_{j=i+1}^{N} A_{i,j} u_j^{(k)}, \right)$$

Теорема. Необходимое и достаточное условие сходимости
Неоходимым и достаточным условием сходимости метода Гаусса-Зейделя является требование, чтобы все значения λ_i, определяемые уравнением:

$$\det \begin{bmatrix} \lambda A_{1,1} & A_{1,2} & A_{1,3} & \dots & A_{1,N} \\ \lambda A_{2,1} & \lambda A_{2,2} & A_{2,3} & \dots & A_{2,N} \\ \dots & \dots & \dots & \dots & \dots \\ \lambda A_{N,1} & \lambda A_{N,2} & \lambda A_{N,3} & \dots & \lambda A_{N,N} \end{bmatrix} = 0,$$

были по модулю меньше единицы: $|\lambda_i| < 1 \ \ \forall i$.
Метод Гаусса–Зайделя иногда сходится, даже если эти условия не выполняются.

Доказательство. Матрица перехода равна:

$$R = -(L + D)^{-1} U.$$

$$\det(R - \lambda E) = 0,$$
$$\det(-(L + D)^{-1} U - \lambda E) = 0,$$
$$\det(-(L + D)^{-1}) \det(U + \lambda(L + D)) = 0.$$

Получим, что

$$\det(U + \lambda(L + D)) = 0$$

□

Недостатки метода:

- Медленная сходимость для больших систем уравнений;
- Чувствителен к выбору начального приближения. Если начальное приближение далеко от точного решения, то метод может сходиться медленно или расходиться;
- Не гарантирует сходимость для некоторых систем уравнений. В таких случаях может потребоваться использование других методов решения систем уравнений.

7.3 SOR (Successive over-relaxation)

Формулировка
Дана квадратная система из N линейных уравнений с неизвестным $\mathbf{u}$:

$$A\mathbf{u} = \mathbf{f} \Leftrightarrow \sum_{i=0} A_{ij} u_j = f_i, i = \overline{0, N-1}$$

где:

$$A = \begin{bmatrix} a_{11} & a_{12} \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{n1} & a_{n2} \dots & a_{nn} \end{bmatrix}, \mathbf{u} = \begin{bmatrix} u_1 \\ \dots \\ u_n \end{bmatrix}, \mathbf{f} = \begin{bmatrix} f_1 \\ \dots \\ f_n \end{bmatrix}$$

Причем матрица A представима в виде суммы: $A = D + L + U$ (LU - разложение), где D - матрица с диагональными элементами, L, U - матрицы с нижнее и верхне треугольными элементами соответственно.

Тогда можно переписать СЛАУ:

$$A\mathbf{u} = \mathbf{f},$$

$$\omega L\mathbf{u} + \omega U\mathbf{u} + \omega D\mathbf{u} = \omega\mathbf{f},$$

$$D\mathbf{u} + \omega L\mathbf{u} + \omega U\mathbf{u} + \omega D\mathbf{u} = \omega\mathbf{f} + D\mathbf{u},$$

$$D\mathbf{u} + \omega L\mathbf{u} = \omega\mathbf{f} + D\mathbf{u} - \omega U\mathbf{u} - \omega D\mathbf{u},$$

$$(D + \omega L)\mathbf{u} = \omega\mathbf{f} - [\omega U + (\omega - 1)D]\,\mathbf{u},\ (\omega > 1)$$

Итеративно метод может быть записан, как:

$$\mathbf{u}^{(k+1)} = (D + \omega L)^{-1}\left(\omega\mathbf{f} - [\omega U + (\omega - 1)D]\,\mathbf{u}^{(k)}\right) = L_\omega\mathbf{u}^{(k)} + c,$$

Однако, используя преимущество треугольной формы $(D + \omega L)$, элементы $u^{(k+1)}$ могут быть вычислены последовательно с использованием прямой подстановки:

$$u_i^{(k+1)} = (1-\omega)u_i^{(k)} + \frac{\omega}{A_{ii}}\left(f_i - \sum_{j<i} A_{ij}u_j^{(k+1)} - \sum_{j>i} A_{ij}u_j^{(k)}\right), i = \overline{0, N-1}$$

Вот тут не хватает доказательства сходимости метода и обоснования того, что при:

1. $\omega = 1$ получаем метод Зейделя (сказать словами);
2. $1 < \omega < 2$ метод последовательной верхней релаксации;
3. $0 < \omega < 1$ метод последовательной нижней релаксации;
4. $\omega < 0 \vee \omega > 2$ метод расходится;

Алгоритм SOR

- Задать начальное приближение;
- Совершить итерацию

$$u_i^{n+1} = (1-\omega)u_i^n + \frac{\omega}{A_{i,i}}\left[f_i - \sum_{j=0}^{i-1} A_{i,j}u_j^{n+1} - \sum_{j=i+1}^{N-1} A_{i,j}u_j^n\right],$$

где ω - параметр релаксации. Требование к устойчивости $\omega \in (1; 2)$.

Хочется, чтобы было проще итерироваться по циклу, поэтому упростим выражение в скобках, путем вынесения нулевого члена за скобку.

$$-\sum_{j=0}^{i-1} A_{i,j}u_j^{n+1} - \sum_{j=i+1}^{N-1} A_{i,j}u_j^n = -A_{i,0}u_0^{n+1} + A_{i,i}u_i^n - \sum_{j=1}^{i-1} A_{i,j}u_j^{n+1} - \sum_{j=i}^{N-1} A_{i,j}u_j^n$$

$$u_j = \begin{cases} u_i^{n+1}, j < i \\ u_j^n, j \geq i \end{cases} \Rightarrow A_{i,j}u_j = \begin{cases} A_{i,j}u_j^{n+1}, j < i \\ A_{i,j}u_j^n, j \geq i \end{cases} \tag{1}$$

$$\sum_{j=0}^{N-1} A_{i,j}u_j = \sum A_{i,j}u_j^{n+1}$$

В результате упрощений придем к упрощенному обновлению $\{u\}$

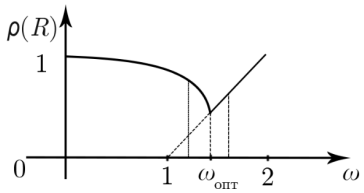
$$u_i+ = \frac{\omega}{A_{i,i}}\left[f_i - \sum_{j=0}^{N-1} A_{i,j}u_j\right]$$

Теорема (Островского – Рейча) Условие сходимости
Пусть матрица симметрична $A = A^T > 0$. Тогда метод релаксации сходится для любого $\omega \in (0; 2)$.

Возникает вопрос, при каких значениях ω_{opt} метод сходится быстрее всего. В общем случае ответа на этот вопрос нет. Однако значение ω_{opt} известно для специального класса задач. Однако для класса уравнений (уравнения Лапласа и Пуассона), в которых существует ω_{opt}.

Оптимальное значение итерационного параметра равно:

$$\omega_{opt} = \frac{2}{1 + \sqrt{1 - \rho^2(R_{Jacobi})}}, R = -(L + D)^{-1}U$$



На парах Калинин показывал, как найти ω_{opt}, но я хз, что там было.

7.4 CG (Conjugate Gradient Method)

Формулировка

Рассмотрим задачу оптимизации:

$$F(u) = \frac{1}{2}\langle Au, u\rangle - \langle f, u\rangle \to inf, u \in \mathbb{R}^N$$

Заметим, что $F'(u) = Au - f$. Условие экстремума функции $F'(u) = 0$ эквивалентно системе $Au - f = 0$. Функция F достигает своей нижней грани в единственной точке u_*, определяемой уравнением $Au_* = f$. Таким образом, данная задача оптимизации сводится к решению СЛАУ $Au = f$.

Идея метода сопряженных градиентов состоит в следующем:

Пусть $\{p_k\}_{k=1}^N$ - базис в $\mathbb{R}^N$. Тогда для любой точки $u_0 \in \mathbb{R}^N$ вектор $u_* - u_0$ раскладывается по базису $u_* - u_0 = \alpha_1 p_1 + \dots \alpha_N p_N$. Таким образом u_* представимо в виде:

$$u_* = u_0 + \alpha_1 p_1 + \dots \alpha_N p_N$$

каждое следущее приближение вычисляется по формуле:

$$u_k = u_0 + \alpha_1 p_1 + \dots \alpha_N p_N$$

Определение
Два вектора p и q называются сопряженными относительно симметричной матрицы B, если $\langle Bp, q\rangle = 0$.

Как построить базис $\{p_k\}_{k=1}^N$?

В качестве начального приближения u_0 выбираем произвольных вектор. На каждой итерации α_k выбираются по правилу:

$$\alpha_k = \arg\min_{\alpha_k} F(u_{k-1} + \alpha_k p_k)$$

Базисные векторы $\{p_k\}$ вычисляются по формулам:

$$p_1 = -F'(u_0)$$

$$p_{k+1} = -F'(u_k) + \beta_k p_k$$

Коэффициенты β_k выбираются так, чтобы векторы p_k и p_{k+1} были сопряженными относительно A.

$$\beta_k = \frac{\langle F'(u_k), Ap_k\rangle}{\langle Ap_k, p_k\rangle}$$

.

Если обозначить $r_k = f - Au_k = -F'(u_k)$, то получим окончательные формулы, используемые при применении метода сопряженных градиентов на практике.

Алгоритм сопряженных градиентов

1. Задать начальное приближение u^0;
2. $r^0 = f - Au^0$
3. $z^0 = r^0$

k - я итерация метода

1. $\alpha_k = \frac{(r^{k-1}, r^{k-1})}{(Az^{k-1}, z^{k-1})}$
2. $u^k = u^{k-1} + \alpha_k z^{k-1}$
3. $r^k = r^{k-1} - \alpha_k A z^{k-1}$
4. $\beta_k = \frac{(r^k, r^k)}{(r^{k-1}, r^{k-1})}$
5. $z^k = r^k + \beta_k z^{k-1}$

Сходимость метода
Если все вычисления точные, и исходные данные точны то метод сходится к решению системы не более чем за N итераций. Более тонкий анализ показывает, что число итераций не превышает M - количество собственных значений матрицы A.
В Matlab существует готовая функция pcg (preconditioned conjugate gradients method).

7.5 BiCGStab (Biconjugate Gradient Stabilized Method)

Формулировка

Дана квадратная система из N линейных уравнений с неизвестным $\mathbf{u}$:

$$A\mathbf{u} = \mathbf{f} \Leftrightarrow \sum_{i=0} A_{ij} u_j = f_i, i = \overline{0, N-1}$$

где:

$$A = \begin{bmatrix} a_{11} & a_{12} \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{n1} & a_{n2} \dots & a_{nn} \end{bmatrix}, \mathbf{u} = \begin{bmatrix} u_1 \\ \dots \\ u_n \end{bmatrix}, \mathbf{f} = \begin{bmatrix} f_1 \\ \dots \\ f_n \end{bmatrix}$$

Алгоритм BiCGStab
Подготовка перед итерационным процессом

1. Выберем начальное приближение u^0
2. $r^0 = f - Au^0$
3. $\widetilde{r} = r^0$
4. $\rho^0 = \alpha^0 = \omega^0 = 1$
5. $v^0 = p^0 = 0$

k - я итерация метода

1. $\rho^k = (\widetilde{r}, r^{k-1})$
2. $\beta^k = \frac{\rho^k}{\rho^{k-1}} \frac{\alpha^{k-1}}{\omega^{k-1}}$
3. $p^k = r^{k-1} + \beta^k (p^{k-1} - \omega^{k-1} v^{k-1})$
4. $v^k = Ap^k$
5. $\alpha^k = \frac{\rho^k}{(\widetilde{r}, v^k)}$
6. $s^k = r^{k-1} - \alpha^k v^k$
7. $t^k = As^k$
8. $\omega^k = \frac{[t^k, s^k]}{[t^k, t^k]}$
9. $u^k = u^{k-1} + \omega^k s^k + \alpha^k p^k$
10. $r^k = s^k - \omega^k t^k$

Обозначения

$$(u, v) = \sum_{i=1}^{N} u_i v_i$$

Критерии останова

- Заданная невязка $\frac{||r^k||}{||f||} < \epsilon$;
- Число итераций $k \leq k_{max}$;
- $|\omega^k| < \epsilon_\omega$, где ϵ_ω заранее заданно

В Matlab существует готовая функция bicgstab (stabilized biconjugate gradients method).

8 Выводы

Если у вы дошли до этой части и спокойно ориентируетесь в коде (можете добавлять свои методы), который был написан на парах Калинина, то вы не потеряетесь на экзамене и сможете:

1. Объяснить любую строчку кода;
2. Добавить новый метод хранения матрицы;
3. Объяснить теорию;

Удачи!