Московский государственный университет им. М.В.Ломоносова

Факультет вычислительной математики и кибернетики

Отчёт о выполнении практической работы № 1

Прямые и итерационные методы решения СЛАУ

Выполнил

Маслов Н.С.
группа 205

Москва, 2014

Постановка задачи

Дана система уравнений $Ax = f$ порядка $n \times n$ с невырожденной матрицей A. Требуется написать программу, рещающую систему линейных алгебраических уравнений заданного пользователем размера (n - параметр программы) методом Гаусса и методом Гаусса с выбором главного элемента.

Предусмотреть возможность задания элементов матрицы системы и её правой части как во входном файле данных, так и путём задания специальных формул.

Цели и задачи практической работы

1. Решить заданную СЛАУ методом Гаусса и методом Гаусса с выбором главного элемента;
2. Вычислить определитель матрицы $det(A)$;
3. Вычислить обратную матрицу A^{-1};
4. Исследовать вопрос вычислительной устойчивости метода Гаусса (при больших значениях параметра n);
5. Правильность решения СЛАУ подтвердить системой тестов (например, можно использовать ресурсы on-line системы Wolfram|Alpha, пакета Maple и т.п.)

Описание методов решения

Метод Гаусса

Метод Гаусса построен на идее приведения элементарными преобразованиями матрицы к диагональному виду, при этом параллельно те же преобразования делаются с вектором правой части выражения.

Ниже приведён алгоритм решения и исходный код модуля. (Исходный код базовых операций с векторами и матрицами приведён в приложении 1).

Прямой ход метода Гаусса:

1. Положим $i = 1$;
2. Выберем первую строку матрицы, в которой i-й элемент ненулевой; если это не i-я строка, то поменяем текущую и найденную строку местами (то же преобразование проведём с вектором правой части);
3. Разделим эту строку (и соответствующее значение в векторе правой части) на значение i-го элемента: так на диагонали мы получим единицу;
4. Вычтем из последующих строк, в который i-й элемент ненулевой, текущую, домноженную на значение i-го элемента уменьшаемой строки: так мы получим нулевые элементы в i-м столбце на диагонали. Аналогичное действие проделываем в векторе правой части;
5. Увеличим i на 1 и, если $i \neq n$, перейдём к шагу 2.

В результате прямого хода получается треугольная матрица с единицами на диагонали. При обратном ходе метода Гаусса матрица переходит к диагональному виду. Обратный ход метода Гаусса проводится следующим образом:

1. Положим $i = n$, где n - порядок матрицы (выбираем последнюю строку);

2. Вычтем из всех вышестоящих строк i-ю, домноженную на значение соответствующего элемента строки (задача - оставить нули в i-м столбце над диагональю); аналогичное действие делаем с вектором правой части;

3. Уменьшаем i на 1; если $i \neq 0$, переходим к шагу 2.

Рассмотрим пример решения СЛАУ методом Гаусса.

Дана система уравнений

$$\begin{cases} x + 2y + 3z = 4 \\ 4x + 5z = 8 \\ 2z = 0 \end{cases}$$

Решение методом Гаусса:

$$\left[\begin{array}{ccc|c} 1 & 2 & 3 & 4 \\ 4 & 0 & 5 & 8 \\ 0 & 0 & 2 & 0 \end{array}\right] \to \left[\begin{array}{ccc|c} 1 & 2 & 3 & 4 \\ 0 & -8 & -7 & -8 \\ 0 & 0 & 2 & 0 \end{array}\right] \to \left[\begin{array}{ccc|c} 1 & 2 & 3 & 4 \\ 0 & 1 & 7/8 & 1 \\ 0 & 0 & 1 & 0 \end{array}\right] \to \left[\begin{array}{ccc|c} 1 & 2 & 0 & 4 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{array}\right] \to \left[\begin{array}{ccc|c} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{array}\right]$$

Ответ: $(2, 1, 0)$.

Аналогичным образом метод Гаусса позволяет искать обратную матрицу; в этом случае в правой части записи вместо вектора правой части записывается единичная матрица соответствующего размера.

В коде программы был применён унифицированный подход: к реализации алгоритма метода Гаусса подключается набор функций-операторов правой части, таким образом, можно одним и тем же кодом работать и с вектором для решения СЛАУ, и с матрицей для поиска обратной.

Исходный код модуля на языке Си:

Листинг 1: Исходный код метода Гаусса (файл gauss.c)

```c
#include "gauss.h"
#include "matrix.h"

/** Секция векторных операций для метода Гаусса */

/* Перестановка элементов вектора */
static void gv_swap(int i, int j, void *arg)
{
	vector_t *f = (vector_t *) arg;

	number_t t = f->vector[i];
	f->vector[i] = f->vector[j];
	f->vector[j] = t;
}

/* Деление элемента вектора на заданное число */
static void gv_div(int i, number_t d, void *arg)
{
	if (!NOT_ZERO(d))
		return;

	vector_t *f = (vector_t *) arg;

	f->vector[i] /= d;
```

```
}

/* Линейная комбинация: к элементу вектора dst прибавляется элемент
 * вектора src, умноженный на коэффициент mul */
static void gv_hit(int src, int dst, number_t mul, void *arg)
{
        vector_t *f = (vector_t *) arg;

        f->vector[dst] += mul * f->vector[src];
}

/* Таблица операций для решения СЛАУ методом Гаусса */
static gauss_handlers_t vect_ops = {
        .swap = gv_swap,
        .div = gv_div,
        .hit = gv_hit
};

/* Решение СЛАУ методом Гаусса */
number_t gauss_solve(matrix_t m, vector_t *f)
{
        if (f == NULL) {
                fprintf(stderr, "ERROR in gauss_solve(): No vector\n");
                return 0;
        }

        /* Запуск полного цикла метода Гаусса; оперируем вектором */
        number_t d = gauss_full(m, &vect_ops, f);

        return d;
}


/** Секция матричных операций для метода Гаусса обратная( матрица) */

/* Перестановка строк в матрице */
static void gm_swap(int i, int j, void *arg)
{
        matrix_t *m = (matrix_t *) arg;

        matrix_exchangeRows(*m, i, j);
}

/* Деление элементов строки на заданное число */
static void gm_div(int i, number_t div, void *arg)
{
        if (!NOT_ZERO(div))
                return;

        matrix_t *m = (matrix_t *) arg;

        matrix_divRow(*m, i, div);
}

/* Линейная комбинация строк матрицы: к строке dst прибавляется строка
 * src, умноженная на число mul */
static void gm_hit(int src, int dst, number_t mul, void *arg)
{
        matrix_t *m = (matrix_t *) arg;

        matrix_hitRow(*m, src, dst, mul);
```

```c
}

/* Таблица операций для вычисления обратной матрицы методом Гаусса */
static gauss_handlers_t matr_ops = {
        .swap = gm_swap,
        .div = gm_div,
        .hit = gm_hit
};

/* Вычисление обратной матрицы методом Гаусса */
matrix_t gauss_invert(matrix_t m)
{
        /* Создаём результирующую матрицу единичную() */
        matrix_t result = matrix_create(m.size);
        for (int i=0; i<m.size; i++) {
                result.matrix[i][i] = 1;
        }

        /* запускаем полный цикл метода Гаусса */
        gauss_full(m, &matr_ops, &result);
        return result;
}

/** Секция общих операций метода Гаусса */

/* Полный цикл метода Гаусса: прямой и обратный ход */
number_t gauss_full(matrix_t m, gauss_handlers_t *h, void *arg)
{
        /* Прямой ход; по пути считается определитель матрицы */
        number_t det = gauss_direct(m, h, arg);

        /* Обратный ход */
        gauss_reverse(m, h, arg);
        return det;
}

/* Прямой ход метода Гаусса */
number_t gauss_direct(matrix_t m, gauss_handlers_t *h, void *arg)
{
        number_t det = 1;

        /* Пройдём всю матрицу построчно, на каждом этапе выбирая первую строку,
         * iй- элемент которой не равен 0 */
        for (int i = 0; i < m.size; i++) {
                int base = i;

                while (!NOT_ZERO(matrix_elem(m, base, i)) && base < m.size)
                        base++;

                /* Если мы дошли до конца матрицы, а такой строки нет, то
                 * матрица вырождена и задача смысла не имеет */
                if (base == m.size) {
                        fprintf(stderr, "ERROR: Matrix is singular\n");
                        return 0;
                }

                /* Переставляем найденную строку на требуемую позицию;
                 * делаем перестановку элементов в векторе или строк в матрице */
                matrix_exchangeRows(m, i, base);
                h->swap(i, base, arg);

```

```c
		/* Если поменяли местами строки - меняем знак определителя */
		if (i != base)
			det = -det;

		/* Теперь работаем с iй- строкой матрицы - здесь диагональный
		 * элемент ненулевой. Делим значения строки на значение первого
		 * элемента. На это же значение умножаем значение будущего
		 * определителя матрицы */
		number_t divider = matrix_elem(m, i, i);
		det *= divider;

		/* Делим на это же число и элемент в векторе строку( в матрице) */
		h->div(i, divider, arg);
		matrix_divRow(m, i, divider);

		/* Вычитаем строку из всех тех оставшихся, в которых iй- элемент
		 * ненулевой с( домножением на соответствующий коэффициент */
		for (int j = i + 1; j < m.size; j++) {
			if (NOT_ZERO(matrix_elem(m, j, i))) {
				number_t mul = matrix_elem(m, j, i);
				matrix_hitRow(m, i, j, -mul);
				h->hit(i, j, -mul, arg);
			}
		}
	}

	/* Возвращаем определитель исходной матрицы */
	return det;
}

/* Обратный ход метода Гаусса */
void gauss_reverse(matrix_t m, gauss_handlers_t *h, void *arg)
{
	for (int i = m.size - 1; i >= 0; i--) {
		for (int j = i - 1; j >= 0; j--) {
			if (NOT_ZERO(matrix_elem(m, j, i))) {
				h->hit(i, j, -matrix_elem(m, j, i), arg);
				matrix_elem(m, j, i) = 0;
			}
		}
	}
}
```

Модифицированный метод Гаусса

Отличие модифицированного метода Гаусса от классического заключается в том, что на каждом шаге прямого хода мы ищем наибольший по модулю элемент в строке, а не среди строк в одном столбце. С тем учётом, что найденный элемент будет наибольшим по модулю, после деления строки на это значение мы получим значения в интервале $[0, 1]$, что позволяет несколько сохранить точность вычислений с учётом ошибок округления (из-за конечности длины машинного слова).

Поскольку в этом методе происходит смена мест столбцов, а не строк (что равнозначно переименованию переменных), появляется необходимость переставить элементы в векторе результата (соответственно, строки в матрице, если мы считаем этим методом обратную матрицу).

Исходный код модуля на языке Си:

Листинг 2: Исходный код модифицированного метода Гаусса (файл gauss_mod.c)

```
#include "gauss_mod.h"
#include <math.h>

/* Массив, в котором будет содержаться перестановка переменных при перестановке
 * строк в модифицированном методе Гаусса */
static int *sequence;

/** Секция векторных операций для( решения СЛАУ) */

/* Перестановка переменных в векторе будет( произведена в самом конце) */
static void gv_swap(int i, int j, void *arg)
{
	int t = sequence[i];
	sequence[i] = sequence[j];
	sequence[j] = t;
}

/* Деление элемента вектора на число d */
static void gv_div(int i, number_t d, void *arg)
{
	if (!NOT_ZERO(d))
		return;

	vector_t *f = (vector_t *) arg;

	f->vector[i] /= d;
}


/* Линейная комбинация: к элементу вектора dst прибавляется элемент
 * вектора src, умноженный на коэффициент mul */
static void gv_hit(int src, int dst, number_t mul, void *arg)
{
	vector_t *f = (vector_t *) arg;

	f->vector[dst] += mul * f->vector[src];
}

/* Таблица операций для решения СЛАУ модифицированным методом Гаусса */
static gauss_mod_handlers_t vect_ops = {
	.swap = gv_swap,
	.div = gv_div,
	.hit = gv_hit
};

/* Решение СЛАУ модицифицированным методом Гаусса */
number_t gauss_mod_solve(matrix_t m, vector_t *f)
{
	if (f == NULL) {
		fprintf(stderr, "ERROR in gauss_solve(): No vector\n");
		return 0;
	}

	/* Создаём массив - перестановку переменных */
	sequence = (int *) malloc (m.size * sizeof (int));
	for (int i=0; i<m.size; i++) {
		sequence[i] = i;
	}

```

```
        /* Проводим полный цикл модифицированного метода Гаусса */
        number_t d = gauss_mod_full(m, &vect_ops, f);

        /* Копируем полученный вектор значений; в нём переменные идут в
         * первоначальном порядке. Затем переставляем значения согласно
         * новому порядку переменных */
        vector_t old = vector_copy(*f);
        for (int i=0; i<m.size; i++) {
                f->vector[sequence[i]] = old.vector[i];
        }

        /* Освобождаем память */
        free(sequence);
        vector_free(old);

        return d;
}

/** Секция матричных операций для( вычисления обратной матрицы) */

/* Перестановка строк матрицы откладывается( до конца вычислений) */
static void gm_swap(int i, int j, void *arg)
{
        int t = sequence[i];
        sequence[i] = sequence[j];
        sequence[j] = t;
}

/* Деление строки матрицы на число div */
static void gm_div(int i, number_t div, void *arg)
{
        if (!NOT_ZERO(div))
                return;

        matrix_t *m = (matrix_t *) arg;

        matrix_divRow(*m, i, div);
}

/* Линейная комбинация строк матрицы: к строке dst прибавляется строка
 * src, умноженная на число mul */
static void gm_hit(int src, int dst, number_t mul, void *arg)
{
        matrix_t *m = (matrix_t *) arg;

        matrix_hitRow(*m, src, dst, mul);
}

/* Таблица операций для вычисления обратной матрицы модифицированным
 * методом Гаусса */
static gauss_mod_handlers_t matr_ops = {
        .swap = gm_swap,
        .div = gm_div,
        .hit = gm_hit
};

/* Вычисление обратной матрицы модифицированным методом Гаусса */
matrix_t gauss_mod_invert(matrix_t m)
{
        /* Создаём единичную матрицу для подготовки результата, а также
         * выделяем память для хранения перестановки столбцов исходной матрицы.
```

```
	 * В итоге эта перестановка будет применяться к строкам обратной */
	sequence = (int *) malloc (m.size * sizeof (int));

	matrix_t result = matrix_create(m.size);
	for (int i=0; i<m.size; i++) {
		result.matrix[i][i] = 1;
		sequence[i] = i;
	}

	/* Выполняем полный цикл модиф. метода Гаусса: прямой и обратный ход */
	gauss_mod_full(m, &matr_ops, &result);

	/* Сохраняем предыдущий порядок следования строк обратной матрицы */
	int *old_rows = (int *) malloc (m.size * sizeof (int));
	for (int i=0; i<m.size; i++) {
		old_rows[i] = result.rows[i];
	}

	/* Переставляем строки обратной матрицы согласно перестановке столбцов
	 * исходной матрицы */
	for (int i=0; i<m.size; i++) {
		result.rows[sequence[i]] = old_rows[i];
	}

	/* Освобождаем память */
	free(old_rows);
	free(sequence);

	return result;
}

/** Секция общих операций модифицированного метода Гаусса */

/* Полный цикл модицифированнного метода Гаусса: прямой и обратный ход */
number_t gauss_mod_full(matrix_t m, gauss_mod_handlers_t *h, void *arg)
{
	/* Прямой ход; по пути считаем определитель матрицы */
	number_t det = gauss_mod_direct(m, h, arg);

	/* Обратный ход */
	gauss_mod_reverse(m, h, arg);
	return det;
}

/* Прямой ход модифицированного метода Гаусса */
number_t gauss_mod_direct(matrix_t m, gauss_mod_handlers_t *h, void *arg)
{
	number_t det = 1;

	/* Проходим матрицу построчно; на каждом этапе в строке выбираем столбец с
	 * наибольшим по модулю элементом и перемещаем этот столбец в текущее
	 * положение. Таким образом, на диагонали окажутся наибольшие по модулю
	 * элементы - главные элементы матрицы. */
	for (int i = 0; i < m.size; i++) {
		int base_elem = i;

		/* Выбираем максимальный по модулю элемент в строке */
		number_t max_elem = fabs(matrix_elem(m, i, i));
		int max_elem_ind = i;
		for (int j = i + 1; j < m.size; j++) {
			if (fabs(matrix_elem(m, i, j)) > max_elem) {
```

```
				max_elem = fabs(matrix_elem(m, i, j));
				max_elem_ind = j;
			}
		}

		/* Если наибольший по модулю элемент нулевой - матрица вырождена */
		if (!NOT_ZERO(max_elem)) {
			fprintf(stderr, "ERROR: Matrix is singular\n");
			return 0;
		}

		/* Меняем текущий столбец на столбец с главным элементом */
		matrix_swapCols(m, i, max_elem_ind);
		h->swap(i, max_elem_ind, arg);

		/* Если произошла смена столбцов - меняем знак определителя */
		if (i != max_elem_ind)
			det = -det;

		/* Делим значения строки на значение главного элемента. На это же
		 * значение умножаем значение будущего определителя матрицы */
		number_t divider = matrix_elem(m, i, i);
		det *= divider;
		h->div(i, divider, arg);
		matrix_divRow(m, i, divider);

		/* Вычитаем строку из тех оставшихся, в которых в данном столбце
		 * остались ненулевые элементы */
		for (int j = i + 1; j < m.size; j++) {
			if (NOT_ZERO(matrix_elem(m, j, i))) {
				number_t mul = matrix_elem(m, j, i);
				matrix_hitRow(m, i, j, -mul);
				h->hit(i, j, -mul, arg);
			}
		}
	}

	return det;
}

/* Обратный ход модифицированного метода Гаусса; ничем не отличается от такого
 * для оригинального метода */
void gauss_mod_reverse(matrix_t m, gauss_mod_handlers_t *h, void *arg)
{
	for (int i = m.size - 1; i >= 0; i--) {
		for (int j = i - 1; j >= 0; j--) {
			if (NOT_ZERO(matrix_elem(m, j, i))) {
				h->hit(i, j, -matrix_elem(m, j, i), arg);
				matrix_elem(m, j, i) = 0;
			}
		}
	}
}
```

Метод верхней релаксации, метод Зейделя

Метод верхней релаксации и метод Зейделя (как частный случай метода релаксаци) - итерационные методы решения СЛАУ. Алгоритм работы заключается в следующем.

Рассмотрим квадратную матрицу

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix}$$

Разложим её на сумму трёх матриц

$$A = D + T_H + T_B$$

,

где D - часть матрицы A, содержащая её главную диагональ, T_H - нижняя треугольная часть, T_B - верхняя треугольная часть (без диагонали).

Введём параметр ω и запишем рекуррентное соотношение

$$(D + \omega T_H)\frac{(x_{k+1} - x_k)}{\omega} + Ax_k = f$$

Если матрица A самосопряжённая и положительно определённая, либо является матрицей с диагональным преобладанием (при $\omega = 1$), при итерировании алгоритма по k вектор x_k будет сходиться к решению СЛАУ $Ax = f$.

При значении параметра $\omega = 1$ метод называется *методом Зейделя*.

Для построения алгоритма вычисления очередной итерации нужно разделить в левой части члены, содержащие x_k и x_{k+1}:

$$\left(\frac{1}{\omega}D + T_H\right)x_{k+1} + \left[\left(1 - \frac{1}{\omega}\right)D + T_B\right]x_k = f$$

При переходе от векторной записи к поэлементной, получаем формулу для компонент x_{k+1}:

$$x_i^{k+1} = x_i^k + \frac{\omega}{a_{ii}}\left(f_i - \sum_{j=1}^{i-1} a_{ij}x_j^{k+1} - \sum_{j=i}^{n} a_{ij}x_j^k\right), i = 1, ..., n$$

Эта формула уже довольно просто реализуется в алгоритме.

Условием окончания итерирования можно установить определённое число итераций, либо достижения достаточной точности решения. Последнее можно определить, наблюдая за нормой вектора невязки $\|Ax_k - f\|$. В приведённой ниже реализации применён комбинированный подход: ограничено число итераций, а также ведётся контроль нормы невязки.

Исходный код модуля на языке Си:

Листинг 3: Метод верхней релаксации (файл relax.c)

```
#include "relax.h"

#include "matrix.h"
#include <math.h>

/* Итерация метода верхней релаксации */
static void relax_iteration(matrix_t m, vector_t f, vector_t old,
                                vector_t new, number_t omega)
{
        /* Ниже реализовано вычисление значения по формуле */
        for (int i=0; i<old.size; i++) {
                number_t sum = f.vector[i];

                for (int j=0; j < i; j++) {
                        sum -= m.matrix[i][j] * new.vector[j];
                }
```

```

                for (int j=i+1; j<old.size; j++) {
                        sum -= m.matrix[i][j] * old.vector[j];
                }

                if (!NOT_ZERO(m.matrix[i][i])) {
                        fprintf(stderr, "ERROR: Zero on diagonal\n");
                        return;
                }

                sum /= m.matrix[i][i];

                new.vector[i] = sum;
        }

        /* Подводим параметр w */
        for (int i=0; i<new.size; i++) {
                new.vector[i] = new.vector[i] * omega + old.vector[i] * (1 - omega);
        }
}

/* Решение СЛАУ методом верхней релаксации с заданной точностью и параметром */
void relax_solve(matrix_t m, vector_t *f, number_t omega, number_t eps)
{
        /* Готовим память для векторов итераций и для подсчёта невязки */
        vector_t x1, x2, diff, tmp_vect;
        x1 = vector_create(m.size);
        x2 = vector_create(m.size);
        diff = vector_create(m.size);

        number_t norm = 0;
        int iters = 0;
        do {
                /* Проводим итерацию метода */
                relax_iteration(m, *f, x1, x2, omega);

                /* Вычисление невязки: считаем A*x2 и помещаем в x1 */
                matrix_mulMatVector(m, x2, x1);
                /* Вычитаем f */
                vector_sub(x1, *f, diff);
                /* Считаем норму невязки */
                norm = vector_norm(diff);

                /* Меняем местами старый и новый вектора для следующей итерации */
                tmp_vect = x1;
                x1 = x2;
                x2 = tmp_vect;

                /* Выводим сообщение о текущей итерации анализ( сходимости) */
                fprintf(stderr, "[RELAX] Iteration %d, residual " NUMBER_WRITE_FORMAT
    "\n", iters + 1, norm);

        } while (iters++ < 50 && norm >= eps);

        /* Помещаем результат в f, откуда его заберут снаружи */
        for (int i=0; i<f->size; i++) {
                f->vector[i] = x1.vector[i];
        }

        /* Освобождаем память */
        vector_free(x1);
```

```
        vector_free(x2);
        vector_free(diff);
}
```

Тестирование

Для тестирования в программу был добавлен вывод значения невязки ответа, а для метода релаксации - вывод значения невязки результата на каждой итерации.

Тест 1. Единичная матрица

Отчёт о тестировании каждого алгоритма с единичной матрицей размера 3×3 и вектором правой части $(1, 2, 3)^T$:

Консоль 1 Метод Гаусса, решение СЛАУ

```
 $ ./matrix -m gauss -o solve < tests/solve1.txt
 [INPUT] Type N, than matrix (NxN)
 [INPUT] Type vector f (length 3)
 [OUTPUT] Result
 1 2 3
 [OUTPUT] Residual: 0
```

Консоль 2 Метод Гаусса, подсчёт определителя

```
  $ ./matrix -m gauss -o det < tests/solve1.txt
[INPUT] Type N, than matrix (NxN)
[OUTPUT] Result
1
```

Консоль 3 Метод Гаусса, подсчёт обратной матрицы

```
  $ ./matrix -m gauss -o invert < tests/solve1.txt
[INPUT] Type N, than matrix (NxN)
[OUTPUT] Result
1 0 0
0 1 0
0 0 1
```

Консоль 4 Модифицированный метод Гаусса, решение СЛАУ

```
 $ ./matrix -m gauss_mod -o solve < tests/solve1.txt
[INPUT] Type N, than matrix (NxN)
[INPUT] Type vector f (length 3)
[OUTPUT] Result
1 2 3
[OUTPUT] Residual: 0
```

Консоль 5 Модифицированный метод Гаусса, подсчёт определителя

```
  $ ./matrix -m gauss_mod -o det < tests/solve1.txt
[INPUT] Type N, than matrix (NxN)
[OUTPUT] Result
1
```

Консоль 6 Модифицированный метод Гаусса, подсчёт обратной матрицы

```
  $ ./matrix -m gauss_mod -o invert < tests/solve1.txt
[INPUT] Type N, than matrix (NxN)
[OUTPUT] Result
1 0 0
0 1 0
0 0 1
```

Консоль 7 Метод верхней релаксации, решение СЛАУ, $\omega = 1$, $\varepsilon = 0.00001$

```
$ ./matrix -m relax -o solve < tests/solve1.txt
[INPUT] Type N, than matrix (NxN)
[INPUT] Type vector f (length 3)
[INPUT] Type 'omega' for overrelaxation method
[INPUT] Type precisioncoefficient (eps)
[RELAX] Iteration 1, residual 0
[OUTPUT] Result
1 2 3
[OUTPUT] Residual: 0
```

Как видно, все три алгоритма справляются с задачами с максимальной точностью.

Тест 2. СЛАУ из списка примеров

В списке примеров (вариант 12) предложена следующая СЛАУ с квадратной невырожденной матрицей:

$$\begin{cases} 2x_1 - 2x_2 + x_4 = -3, \\ 2x_1 + 3x_2 + x_3 - 3x_4 = -6, \\ 3x_1 + 4x_2 - x_3 + 2x_4 = 0, \\ x_1 + 3x_2 + x_3 - x_4 = 2 \end{cases}$$

Консоль 8 Метод Гаусса, решение СЛАУ

```
$ ./matrix -m gauss -o solve < tests/solve2.txt
[INPUT] Type N, than matrix (NxN)
[INPUT] Type vector f (length 4)
[OUTPUT] Result
-2 1 4 3
[OUTPUT] Residual: 1.5384e-15
```

Консоль 9 Метод Гаусса, подсчёт определителя

```
$ ./matrix -m gauss -o det < tests/solve2.txt
[INPUT] Type N, than matrix (NxN)
[OUTPUT] Result
-53
```

Консоль 10 Метод Гаусса, подсчёт обратной матрицы

```
$ ./matrix -m gauss -o invert -f latex < tests/solve2.txt
[INPUT] Type N, than matrix (NxN)
[OUTPUT] Result
```

$$\begin{pmatrix} 0.26415 & 0.16981 & 0.075472 & -0.09434 \\ -0.16981 & -0.037736 & 0.09434 & 0.13208 \\ 0.37736 & -0.4717 & -0.32075 & 1.1509 \\ 0.13208 & -0.41509 & 0.037736 & 0.45283 \end{pmatrix}$$

Консоль 11 Модифицированный метод Гаусса, решение СЛАУ

```
 $ ./matrix -m gauss_mod -o solve < tests/solve2.txt
[INPUT] Type N, than matrix (NxN)
[INPUT] Type vector f (length 4)
[OUTPUT] Result
-2 1 4 3
[OUTPUT] Residual: 4.3512e-15
```

Консоль 12 Модифицированный метод Гаусса, подсчёт определителя

```
$ ./matrix -m gauss_mod -o det < tests/solve2.txt
[INPUT] Type N, than matrix (NxN)
[OUTPUT] Result
-53
```

Консоль 13 Модифицированный метод Гаусса, подсчёт обратной матрицы

```
$ ./matrix -m gauss_mod -o invert -f latex < tests/solve2.txt
[INPUT] Type N, than matrix (NxN)
[OUTPUT] Result
```

$$\begin{pmatrix} 0.26415 & 0.16981 & 0.075472 & -0.09434 \\ -0.16981 & -0.037736 & 0.09434 & 0.13208 \\ 0.37736 & -0.4717 & -0.32075 & 1.1509 \\ 0.13208 & -0.41509 & 0.037736 & 0.45283 \end{pmatrix}$$

Консоль 14 Метод верхней релаксации, решение СЛАУ, $\omega = 1$, $\varepsilon = 0.00001$

```
$ ./matrix -m relax -o solve < tests/solve2.txt
[INPUT] Type N, than matrix (NxN)
[INPUT] Type vector f (length 4)
[INPUT] Type 'omega' for overrelaxation method
[INPUT] Type precisioncoefficient (eps)
[RELAX] Iteration 1, residual 49.003
[RELAX] Iteration 2, residual 379.64
...
[RELAX] Iteration 50, residual 1.6483e+45
[RELAX] Iteration 51, residual 1.2745e+46
[OUTPUT] Result
2.1895e+44 -6.1365e+44 -2.991e+45 -4.613e+45
[OUTPUT] Residual: 1.2745e+46
```

Как видно из отчёта, такая система не решается методом верхней релаксации, так как матрица коэффициентов не является положительно определённой или матрицей с диагональным преобладанием. Вектор решения в этом случае должен расходиться, что мы и наблюдаем. Однако, при решении системы методом Гаусса мы получаем достаточно точное решение (порядок нормы невязки 10^{-15}).

Тест 3. Положительно определённая матрица (для демонстрации итерационного метода)

Для демонстрации сходимости метода верхней релаксации, рассмотрим положительно определённую матрицу

$$\begin{pmatrix} 1 & 1 & 0 \\ 1 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

с вектором правой части $(1, 2, 3)$.

Консоль 15 Метод Гаусса, решение СЛАУ

```
$ ./matrix -m gauss -o solve < tests/solve_relax2.txt
[INPUT] Type N, than matrix (NxN)
[INPUT] Type vector f (length 3)
[OUTPUT] Result
0 1 3
[OUTPUT] Residual: 0
```

Консоль 16 Метод верхней релаксации, решение СЛАУ, $\omega = 1$, $\varepsilon = 0.00001$

```
$ ./matrix -m relax -o solve < tests/solve_relax2.txt
[INPUT] Type N, than matrix (NxN)
[INPUT] Type vector f (length 3)
[INPUT] Type 'omega' for overrelaxation method
[INPUT] Type precisioncoefficient (eps)
[RELAX] Iteration 1, residual 0.5
[RELAX] Iteration 2, residual 0.25
[RELAX] Iteration 3, residual 0.125
[RELAX] Iteration 4, residual 0.0625
[RELAX] Iteration 5, residual 0.03125
[RELAX] Iteration 6, residual 0.015625
[RELAX] Iteration 7, residual 0.0078125
[RELAX] Iteration 8, residual 0.0039062
[RELAX] Iteration 9, residual 0.0019531
[RELAX] Iteration 10, residual 0.00097656
[RELAX] Iteration 11, residual 0.00048828
[RELAX] Iteration 12, residual 0.00024414
[RELAX] Iteration 13, residual 0.00012207
[RELAX] Iteration 14, residual 6.1035e-05
[RELAX] Iteration 15, residual 3.0518e-05
[RELAX] Iteration 16, residual 1.5259e-05
[RELAX] Iteration 17, residual 7.6294e-06
[OUTPUT] Result
1.5259e-05 0.99999 3
[OUTPUT] Residual: 7.6294e-06
```

Консоль 17 Метод верхней релаксации, решение СЛАУ, $\omega = 0.8$, $\varepsilon = 0.00001$

```
$ ./matrix -m relax -o solve < tests/solve_relax2.txt
[INPUT] Type N, than matrix (NxN)
[INPUT] Type vector f (length 3)
[INPUT] Type 'omega' for overrelaxation method
[INPUT] Type precisioncoefficient (eps)
[RELAX] Iteration 1, residual 0.74833
[RELAX] Iteration 2, residual 0.31496
[RELAX] Iteration 3, residual 0.20207
[RELAX] Iteration 4, residual 0.12653
[RELAX] Iteration 5, residual 0.077129
[RELAX] Iteration 6, residual 0.046529
[RELAX] Iteration 7, residual 0.027968
[RELAX] Iteration 8, residual 0.016791
[RELAX] Iteration 9, residual 0.010077
[RELAX] Iteration 10, residual 0.0060464
[RELAX] Iteration 11, residual 0.0036279
[RELAX] Iteration 12, residual 0.0021768
[RELAX] Iteration 13, residual 0.0013061
[RELAX] Iteration 14, residual 0.00078364
[RELAX] Iteration 15, residual 0.00047018
[RELAX] Iteration 16, residual 0.00028211
[RELAX] Iteration 17, residual 0.00016927
[RELAX] Iteration 18, residual 0.00010156
[RELAX] Iteration 19, residual 6.0936e-05
[RELAX] Iteration 20, residual 3.6562e-05
[RELAX] Iteration 21, residual 2.1937e-05
[RELAX] Iteration 22, residual 1.3162e-05
[RELAX] Iteration 23, residual 7.8973e-06
[OUTPUT] Result
1.5795e-05 0.99999 3
[OUTPUT] Residual: 7.8973e-06
```

Консоль 18 Метод верхней релаксации, решение СЛАУ, $\omega = 1.3$, $\varepsilon = 0.00001$

```
$ ./matrix -m relax -o solve < tests/solve_relax2.txt
[INPUT] Type N, than matrix (NxN)
[INPUT] Type vector f (length 3)
[INPUT] Type 'omega' for overrelaxation method
[INPUT] Type precisioncoefficient (eps)
[RELAX] Iteration 1, residual 1.4396
[RELAX] Iteration 2, residual 0.32955
[RELAX] Iteration 3, residual 0.13734
[RELAX] Iteration 4, residual 0.029229
[RELAX] Iteration 5, residual 0.01338
[RELAX] Iteration 6, residual 0.0026558
[RELAX] Iteration 7, residual 0.0013379
[RELAX] Iteration 8, residual 0.00025454
[RELAX] Iteration 9, residual 0.00013785
[RELAX] Iteration 10, residual 2.6498e-05
[RELAX] Iteration 11, residual 1.4662e-05
[RELAX] Iteration 12, residual 3.0062e-06
[OUTPUT] Result
5.6956e-06 1 3
[OUTPUT] Residual: 3.0062e-06
```

Консоль 19 Метод верхней релаксации, решение СЛАУ, $\omega = 1.6$, $\varepsilon = 0.00001$

```
$ ./matrix -m relax -o solve < tests/solve_relax2.txt
[INPUT] Type N, than matrix (NxN)
[INPUT] Type vector f (length 3)
[INPUT] Type 'omega' for overrelaxation method
[INPUT] Type precisioncoefficient (eps)
[RELAX] Iteration 1, residual 2.5768
[RELAX] Iteration 2, residual 1.4653
[RELAX] Iteration 3, residual 0.8945
[RELAX] Iteration 4, residual 0.53358
[RELAX] Iteration 5, residual 0.32077
[RELAX] Iteration 6, residual 0.19234
[RELAX] Iteration 7, residual 0.11543
[RELAX] Iteration 8, residual 0.069251
[RELAX] Iteration 9, residual 0.041552
[RELAX] Iteration 10, residual 0.024931
[RELAX] Iteration 11, residual 0.014959
[RELAX] Iteration 12, residual 0.0089751
[RELAX] Iteration 13, residual 0.0053851
[RELAX] Iteration 14, residual 0.003231
[RELAX] Iteration 15, residual 0.0019386
[RELAX] Iteration 16, residual 0.0011632
[RELAX] Iteration 17, residual 0.0006979
[RELAX] Iteration 18, residual 0.00041874
[RELAX] Iteration 19, residual 0.00025125
[RELAX] Iteration 20, residual 0.00015075
[RELAX] Iteration 21, residual 9.0448e-05
[RELAX] Iteration 22, residual 5.4269e-05
[RELAX] Iteration 23, residual 3.2561e-05
[RELAX] Iteration 24, residual 1.9537e-05
[RELAX] Iteration 25, residual 1.1722e-05
[RELAX] Iteration 26, residual 7.0333e-06
[OUTPUT] Result
-3.4116e-06 1 3
[OUTPUT] Residual: 7.0333e-06
```

Как видно из отчёта, метод релаксации справляется с задачей с достаточной точностью, при этом наименьшее число итераций для достижения необходимой точности мы получили при значении параметра $\omega = 1.3$. (Значение было подобрано эмпирически, возможно, есть более оптимальное значение, но данных примеров достаточно для демонстрации особенностей использования метода релаксации).

Тест 4. Матрица, заданная по формуле (1)

Пусть матрица и вектор правой части заданы с помощью формул:

$$A_{ij} = \begin{cases} \frac{i+j}{m+n}, i \neq j, \\ n + m^2 + \frac{j}{m} + \frac{i}{n}, i = j \end{cases}$$

,

$$b_i = 200 + 50i$$

где $i, j = 1, ..., n, n = 20, m = 8$.

Матрицы заполняются внутри программы для того, чтобы не терять точность при переводе чисел из формата с плавающей точкой в текстовый и обратно.

Вычисление обратной матрицы здесь опущено из-за размеров матрицы (20).

Консоль 20 Метод Гаусса, решение СЛАУ

```
$ ./matrix -m gauss -o solve -i formula1
[OUTPUT] Result
2.1322 2.6631 3.1927 3.721 4.248 4.7737 5.2981 5.8213 6.3431 6.8637 7.3831 7.9012
8.418 8.9335 9.4478 9.9609 10.473 10.983 11.493 12.001
[OUTPUT] Residual: 1.033e-12
```

Консоль 21 Метод Гаусса, подсчёт определителя

```
$ ./matrix -m gauss -o det -i formula1
[OUTPUT] Result
4.6398e+38
```

Консоль 22 Модифицированный метод Гаусса, решение СЛАУ

```
$ ./matrix -m gauss_mod -o solve -i formula1
[OUTPUT] Result
2.1322 2.6631 3.1927 3.721 4.248 4.7737 5.2981 5.8213 6.3431 6.8637 7.3831 7.9012
8.418 8.9335 9.4478 9.9609 10.473 10.983 11.493 12.001
[OUTPUT] Residual: 1.033e-12
```

Консоль 23 Модифицированный метод Гаусса, подсчёт определителя

```
$ ./matrix -m gauss_mod -o det -i formula1
[OUTPUT] Result
4.6398e+38
```

Консоль 24 Метод верхней релаксации, решение СЛАУ, $\omega = 1$, $\varepsilon = 0.00001$

```
$ ./matrix -m relax -o solve -i formula1
[INPUT] Type 'omega' for overrelaxation method
1
[INPUT] Type precisioncoefficient (eps)
0.00001
[RELAX] Iteration 1, residual 345.83
[RELAX] Iteration 2, residual 17.645
[RELAX] Iteration 3, residual 0.45965
[RELAX] Iteration 4, residual 0.0084147
[RELAX] Iteration 5, residual 0.00053034
[RELAX] Iteration 6, residual 1.4704e-05
[RELAX] Iteration 7, residual 2.7193e-07
[OUTPUT] Result
2.1322 2.6631 3.1927 3.721 4.248 4.7737 5.2981 5.8213 6.3431 6.8637 7.3831 7.9012
8.418 8.9335 9.4478 9.9609 10.473 10.983 11.493 12.001
[OUTPUT] Residual: 2.7193e-07
```

Консоль 25 Метод верхней релаксации, решение СЛАУ, $\omega = 0.8$, $\varepsilon = 0.00001$

```
$ ./matrix -m relax -o solve -i formula1
[INPUT] Type 'omega' for overrelaxation method
0.8
[INPUT] Type precisioncoefficient (eps)
0.00001
[RELAX] Iteration 1, residual 516.51
[RELAX] Iteration 2, residual 86.73
[RELAX] Iteration 3, residual 17.223
[RELAX] Iteration 4, residual 3.7086
[RELAX] Iteration 5, residual 0.79942
[RELAX] Iteration 6, residual 0.16788
[RELAX] Iteration 7, residual 0.034243
[RELAX] Iteration 8, residual 0.00681
[RELAX] Iteration 9, residual 0.0013281
[RELAX] Iteration 10, residual 0.00025572
[RELAX] Iteration 11, residual 4.8997e-05
[RELAX] Iteration 12, residual 9.424e-06
[OUTPUT] Result
2.1322 2.6631 3.1927 3.721 4.248 4.7737 5.2981 5.8213 6.3431 6.8637 7.3831 7.9012
8.418 8.9335 9.4478 9.9609 10.473 10.983 11.493 12.001
[OUTPUT] Residual: 9.424e-06
```

Консоль 26 Метод верхней релаксации, решение СЛАУ, $\omega = 0.8$, $\varepsilon = 0.00001$

```
$ ./matrix -m relax -o solve -i formula1
[INPUT] Type 'omega' for overrelaxation method
1.3
[INPUT] Type precisioncoefficient (eps)
0.00001
[RELAX] Iteration 1, residual 1421.7
[RELAX] Iteration 2, residual 572.57
[RELAX] Iteration 3, residual 225.95
[RELAX] Iteration 4, residual 87.296
[RELAX] Iteration 5, residual 33.075
[RELAX] Iteration 6, residual 12.316
[RELAX] Iteration 7, residual 4.5156
[RELAX] Iteration 8, residual 1.6332
[RELAX] Iteration 9, residual 0.58347
[RELAX] Iteration 10, residual 0.20615
[RELAX] Iteration 11, residual 0.072096
[RELAX] Iteration 12, residual 0.024978
[RELAX] Iteration 13, residual 0.0085782
[RELAX] Iteration 14, residual 0.0029218
[RELAX] Iteration 15, residual 0.00098738
[RELAX] Iteration 16, residual 0.00033118
[RELAX] Iteration 17, residual 0.00011028
[RELAX] Iteration 18, residual 3.6462e-05
[RELAX] Iteration 19, residual 1.1973e-05
[RELAX] Iteration 20, residual 3.9048e-06
[OUTPUT] Result
2.1322 2.6631 3.1927 3.721 4.248 4.7737 5.2981 5.8213 6.3431 6.8637 7.3831 7.9012
8.418 8.9335 9.4478 9.9609 10.473 10.983 11.493 12.001
[OUTPUT] Residual: 3.9048e-06
```

Все три метода справились с задачей с достаточной точностью.

Тест 5. Матрица, заданная по формуле (2)

Пусть матрица и вектор правой части заданы по формуле:

$$A_{ij} = \begin{cases} q_M^{i+j} + 0.1(j-i), i \neq j, \\ (q_M - 1)^{i+j}, i = j \end{cases}$$

,

$$q_M = 1.001 - 2 \cdot 10^{-3} \cdot M$$

$$b_i = n \cdot \exp\left(\frac{x}{i}\right) \cdot \cos(x)$$

где $i, j = 1, ..., n, n = 100, M = 4$.

Матрицы заполняются внутри программы для того, чтобы не терять точность при переводе чисел из формата с плавающей точкой в текстовый и обратно.

Вычисление обратной матрицы здесь опущено из-за размеров матрицы (100).

Консоль 27 Метод Гаусса, подсчёт определителя

```
$ ./matrix -m gauss -o det -i formula2
[OUTPUT] Result
5.4745e-26
```

Консоль 28 Модифицированный метод Гаусса, подсчёт определителя

```
$ ./matrix -m gauss_mod -o det -i formula2
[OUTPUT] Result
5.4745e-26
```

Положим $x = 0$. Вывод результата будет опущен, для оценки работы алгоритмов достаточно значения нормы невязки.

Консоль 29 Метод Гаусса, решение СЛАУ ($x = 0$)

```
$ ./matrix -m gauss -o solve -i formula2
[INPUT] Type X:
0
[OUTPUT] Result: ... (опущено, очень длинный вывод)
[OUTPUT] Residual: 1.1188e-08
```

Консоль 30 Модифицированный метод Гаусса, решение СЛАУ ($x = 0$)

```
$ ./matrix -o solve -m gauss_mod -i formula2
[INPUT] Type X:
0
[OUTPUT] Result: ... (опущено, очень длинный вывод)
[OUTPUT] Residual: 2.67e-13
```

Как мы видим, модифицированный метод Гаусса дал куда лучшее решение, чем традиционный.

Метод верхней релаксации не подходит для решения этой задачи, так как не будет сходимости вектора решения.

Рассмотрим также решения при других значениях x.

Положим $x = -10$.

Консоль 31 Метод Гаусса, решение СЛАУ ($x = -10$)

```
$ ./matrix -m gauss -o solve -i formula2
[INPUT] Type X:
-10
[OUTPUT] Result: ... (опущено, очень длинный вывод)
[OUTPUT] Residual: 1.0741e-06
```

Консоль 32 Модифицированный метод Гаусса, решение СЛАУ ($x = -10$)

```
$ ./matrix -m gauss_mod -o solve -i formula2
[INPUT] Type X:
-10
[OUTPUT] Result: ... (опущено, очень длинный вывод)
[OUTPUT] Residual: 7.4998e-12
```

Положим $x = 10$.

Консоль 33 Метод Гаусса, решение СЛАУ ($x = 10$)

```
$ ./matrix -m gauss -o solve -i formula2
[INPUT] Type X:
10
[OUTPUT] Result: ... (опущено, очень длинный вывод)
[OUTPUT] Residual: 0.007408
```

Консоль 34 Модифицированный метод Гаусса, решение СЛАУ ($x = 10$)

```
$ ./matrix -m gauss_mod -o solve -i formula2
[INPUT] Type X:
10
[OUTPUT] Result: ... (опущено, очень длинный вывод)
[OUTPUT] Residual: 6.242e-08
```

Как видно, при увеличении по модулю значения x, точность теряется у обоих методов решения, но при этом модифицированный метод Гаусса даёт гораздо более высокую точность решения.

Выводы

Из проделанной работы можно сделать следующие основные выводы.

У каждого из описанных выше методов есть свои достоинства и недостатки в своих сферах применения.

Метод Гаусса является достаточно универсальным методом решения СЛАУ, при этом предоставляя возможность вычисления определителя матрицы и подсчёта обратной матрицы. Модификация метода Гаусса с выбором главного элемента в строке в определённом круге задач может значительно уточнить решение.

Итерационные методы позволяют достаточно быстро получить результат требуемой точности, но требуют определённой подготовки матрицы для сходимости. Итерационные методы, как правило, более устойчивы к особенностям хранения действительных чисел в машинном слове и к соответствующим ошибкам при вычислениях и округлении.

Приложение 1. Исходный код проекта

Исходные коды проекта доступны на Github: https://github.com/webconn/cmc_MatrixMethods

Ниже приведены исходные коды модулей, не описанных выше.

Листинг 4: Исходный код интерфейса программы (файл main.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#define _GNU_SOURCE
#include <getopt.h>

#include "matrix.h"

#include "gauss.h"
#include "gauss_mod.h"
#include "relax.h"

#include "input.h"

static struct option longopts[] = {
        { .name = "help", .has_arg = 0, .flag = NULL, .val = ’h’ },
        { .name = "method", .has_arg = 1, .flag = NULL, .val = ’m’ },
        { .name = "operation", .has_arg = 1, .flag = NULL, .val = ’o’ },
        { .name = "format", .has_arg = 1, .flag = NULL, .val = ’f’ },
        { .name = "input", .has_arg =1, .flag = NULL, .val = ’i’ }
};

static char *input_names[] = {
        "text", "formula1", "formula2"
};

static enum m_inputs {
        INPUT_STDIN = 0,
        INPUT_FORM1 = 1,
        INPUT_FORM2 = 2,
        INPUT_END
} input;

static enum m_methods {
        METHOD_GAUSS = 0,
        METHOD_GAUSS_MOD = 1,
        METHOD_RELAXATION = 2,
        METHOD_END
} method;

static char *method_names[] = {
        "gauss", "gauss_mod", "relax"
};

static char *ops_names[] = {
        "det", "solve", "invert"
};

static enum m_ops {
        OPERATION_DET = 0,
        OPERATION_SOLVE = 1,
        OPERATION_INVERT = 2,
        OPERATION_END
```

```
} operation;

static char *format_names[] = {
        "text", "latex"
};

static format_t format;

static void input_stdin(matrix_t *m, vector_t *v)
{
        fprintf(stderr, "[INPUT] Type N, than matrix (NxN)\n");
        *m = matrix_read(stdin);

        fprintf(stderr, "[INPUT] Type vector f (length %d)\n", m->size);
        *v = vector_readN(stdin, m->size);
}

static void input_stdin_m(matrix_t *m)
{
        fprintf(stderr, "[INPUT] Type N, than matrix (NxN)\n");
        *m = matrix_read(stdin);
}

void op_solve(enum m_methods met, format_t format)
{
        if (met < 0 || met >= METHOD_END) {
                fprintf(stderr, "[ERROR] Unknown method: %d\n", met);
                return;
        }

        matrix_t m;
        vector_t f;

        switch (input) {
                case INPUT_STDIN:
                        input_stdin(&m, &f);
                        break;
                case INPUT_FORM1:
                        input_form1(&m, &f);
                        break;
                case INPUT_FORM2:
                        input_form2(&m, &f);
                        break;
                default:
                        return;
        }

        vector_t f_orig = vector_copy(f);
        matrix_t m_orig = matrix_copy(m);

        number_t omega = 1;
        number_t eps = 0.0001;
        switch (met) {
                case METHOD_GAUSS:
                        gauss_solve(m, &f);
                        break;
                case METHOD_GAUSS_MOD:
                        gauss_mod_solve(m, &f);
                        break;
                case METHOD_RELAXATION:
                        fprintf(stderr, "[INPUT] Type 'omega' for over"
```

```
                                        "relaxation method\n");
                        scanf(NUMBER_READ_FORMAT, &omega);
                        fprintf(stderr, "[INPUT] Type precision"
                                        "coefficient (eps)\n");
                        scanf(NUMBER_READ_FORMAT, &eps);
                        relax_solve(m, &f, omega, eps);
                        break;
        }

        fprintf(stderr, "[OUTPUT] Result\n");
        vector_print(stdout, f, format);

        /* Calculate residual */
        vector_t diff_orig = vector_create(m_orig.size);
        matrix_mulMatVector(m_orig, f, diff_orig);
        vector_sub(diff_orig, f_orig, diff_orig);

        fprintf(stderr, "[OUTPUT] Residual: " NUMBER_WRITE_FORMAT "\n", vector_norm(
    diff_orig));

        matrix_free(m_orig);
        vector_free(f_orig);
        vector_free(diff_orig);
        matrix_free(m);
        vector_free(f);
}

void op_det(enum m_methods met, format_t format)
{
        if (met != METHOD_GAUSS && met != METHOD_GAUSS_MOD) {
                fprintf(stderr, "[INPUT] Determinant calculation methods: "
                                "gauss and gauss_mod\n");
                return;
        }

        matrix_t m;
        switch (input) {
                case INPUT_STDIN:
                        input_stdin_m(&m);
                        break;
                case INPUT_FORM1:
                        input_form1_m(&m);
                        break;
                case INPUT_FORM2:
                        input_form2_m(&m);
                        break;
                default:
                        return;
        }
        vector_t f = vector_create(m.size);

        number_t det = 0;

        switch (met) {
                case METHOD_GAUSS:
                        det = gauss_solve(m, &f);
                        break;
                case METHOD_GAUSS_MOD:
                        det = gauss_mod_solve(m, &f);
                        break;
        }
```

```
	fprintf(stderr, "[OUTPUT] Result\n");
	printf(NUMBER_WRITE_FORMAT "\n", det);

	matrix_free(m);
	vector_free(f);
}

void op_invert(enum m_methods met, format_t format)
{
	if (met != METHOD_GAUSS && met != METHOD_GAUSS_MOD) {
		fprintf(stderr, "[INPUT] Invertion methods: gauss and gauss_mod\n");
		return;
	}

	matrix_t m;
	switch (input) {
		case INPUT_STDIN:
			input_stdin_m(&m);
			break;
		case INPUT_FORM1:
			input_form1_m(&m);
			break;
		case INPUT_FORM2:
			input_form2_m(&m);
			break;
		default:
			return;
	}
	matrix_t inv;

	switch (met) {
		case METHOD_GAUSS:
			inv = gauss_invert(m);
			break;
		case METHOD_GAUSS_MOD:
			inv = gauss_mod_invert(m);
			break;
	}

	fprintf(stderr, "[OUTPUT] Result\n");
	matrix_print(stdout, inv, format);

	matrix_free(m);
	matrix_free(inv);
}

char *argv0;

void print_help()
{
	fprintf(stderr, "Usage: %s -o <operation> -m <method> [-f <format>]\n\n",
			argv0);
	fprintf(stderr, " -o, --operation=<operation>\tOperation name: det, "
			"solve, invert\n");
	fprintf(stderr, " -m, --method=<method>\t\tMethod: gauss, gauss_mod, "
			"relax (only for solve operation)\n");
	fprintf(stderr, " -i, --input=<source>\t\tInput source: text, formula1, "
			"formula2\n");
	fprintf(stderr, " -f, --format=<format>\t\tOutput format: text, latex\n");
	fprintf(stderr, " -h, --help\t\t\tPrint this message\n\n");
```

```
}


int main(int argc, char *argv[])
{
        int c;
        int flag_gotmet = 0, flag_gotop = 0, flag_gotformat = 0, flag_gotinput = 0;
        argv0 = argv[0];

        while ((c = getopt_long(argc, argv, "hm:o:f:i:", longopts, NULL)) > 0) {
                switch (c) {
                        case 'h':
                                print_help();
                                exit(0);
                        case 'i':
                                flag_gotinput = 1;
                                while (input != INPUT_END &&
                                        strcmp(optarg, input_names[input]))
                                        input++;

                                if (input == INPUT_END) {
                                        fprintf(stderr, "ERROR: Unknown input: "
                                                        "%s\n\n", optarg);
                                        print_help();
                                        exit(1);
                                }
                                break;
                        case 'm':
                                flag_gotmet = 1;
                                method = 0;

                                while (method != METHOD_END &&
                                        strcmp(optarg, method_names[method]))
                                        method++;

                                if (method == METHOD_END) {
                                        fprintf(stderr, "ERROR: Unknown method: "
                                                        "%s\n\n", optarg);
                                        print_help();
                                        exit(1);
                                }
                                break;
                        case 'o':
                                flag_gotop = 1;
                                operation = 0;

                                while (operation != OPERATION_END &&
                                       strcmp(optarg, ops_names[operation]))
                                        operation++;

                                if (operation == OPERATION_END) {
                                        fprintf(stderr, "ERROR: Unknown operation: "
                                                        "%s\n\n", optarg);
                                        print_help();
                                        exit(1);
                                }
                                break;
                        case 'f':
                                flag_gotformat = 1;
                                format = 0;

```

```
                                while (format != FORMAT_END &&
                                       strcmp(optarg, format_names[format]))
                                        format++;

                                if (format == FORMAT_END) {
                                        fprintf(stderr, "ERROR: Unknown format: "
                                                        "%s\n\n", optarg);
                                        print_help();
                                        exit(1);
                                }
                                break;
                }
        }

        if (!flag_gotmet || !flag_gotop) {
                print_help();
                exit(0);
        }

        if (!flag_gotformat) {
                format = FORMAT_TEXT;
        }

        if (!flag_gotinput) {
                input = INPUT_STDIN;
        }

        /* select operation and method */
        switch (operation) {
                case OPERATION_SOLVE:
                        op_solve(method, format);
                        break;
                case OPERATION_DET:
                        op_det(method, format);
                        break;
                case OPERATION_INVERT:
                        op_invert(method, format);
                        break;
        }

        return 0;
}
```

Листинг 5: Исходный код библиотеки векторных операций (файл matrix.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "matrix.h"

matrix_t matrix_read(FILE *stream)
{
        /* 1. read matrix size */
        int N;
        fscanf(stream, "%d", &N);
        return matrix_readN(stream, N);
}

matrix_t matrix_create(int n)
{
```

```
	return matrix_readN(NULL, n);
}

matrix_t matrix_readN(FILE *stream, int N)
{
	/* 2. allocate memory for sequences */
	unsigned int *rows, *cols;
	rows = (unsigned int *) malloc(N * sizeof (unsigned int));
	cols = (unsigned int *) malloc(N * sizeof (unsigned int));

	/* 2. allocate memory for matrix */
	number_t **matrix = (number_t **) malloc(N * sizeof (number_t *));

	for (unsigned int i=0; i<N; i++) {
		matrix[i] = (number_t *) malloc(N * sizeof (number_t));
	}

	/* 3. read matrix data */
	for (unsigned int i=0; i<N; i++) {
		for (unsigned int j=0; j<N; j++) {
			if (stream != NULL && fscanf(stream, NUMBER_READ_FORMAT,
							&matrix[i][j]) == 0) {
				fprintf(stderr, "ERROR: Wrong input stream "
						"(unexpected EOF)\n");
				exit(1);
			} else if (stream == NULL) {
				matrix[i][j] = 0;
			}
		}
		rows[i] = i;
		cols[i] = i;
	}

	matrix_t ret = {
		.matrix = matrix,
		.size = N,
		.cols = cols,
		.rows = rows
	};

	return ret;
}

void matrix_print(FILE *stream, matrix_t m, format_t f)
{
	if (f == FORMAT_LATEX)
		fprintf(stream, "$$\\begin{pmatrix}\n");

	for (int i=0; i<m.size; i++) {
		for (int j=0; j<m.size; j++) {
			fprintf(stream, NUMBER_WRITE_FORMAT " ",
					m.matrix[m.rows[i]][m.cols[j]]);
			if (f == FORMAT_LATEX && j != m.size - 1)
				fprintf(stream, " & ");
		}

		if (f == FORMAT_LATEX && i != m.size - 1)
			fprintf(stream, " \\\\");

		fputc('\n', stream);
	}
```

```
	if (f == FORMAT_LATEX)
		fprintf(stream, "\\end{pmatrix}$$\n");
}

matrix_t matrix_copy(matrix_t source)
{
	number_t **m = (number_t **) malloc(source.size * sizeof (number_t *));
	unsigned int *rows = (unsigned int *) malloc(source.size * sizeof (int));
	unsigned int *cols = (unsigned int *) malloc(source.size * sizeof (int));

	for (int i=0; i<source.size; i++) {
		m[i] = (number_t *) malloc(source.size * sizeof (number_t));

		for (int j=0; j<source.size; j++) {
			m[i][j] = source.matrix[i][j];
		}

		rows[i] = source.rows[i];
		cols[i] = source.cols[i];
	}

	matrix_t ret = {
		.matrix = m,
		.size = source.size,
		.rows = rows,
		.cols = cols
	};

	return ret;
}

void matrix_divRow(matrix_t m, int j, number_t div)
{
	if (!NOT_ZERO(div))
		return;

	for (int i = 0; i < m.size; i++) {
		m.matrix[m.rows[j]][i] /= div;
	}
}

void matrix_hitRow(matrix_t m, int src, int dst, number_t mul)
{
	for (int i = 0; i < m.size; i++) {
		m.matrix[m.rows[dst]][i] += mul * m.matrix[m.rows[src]][i];
	}
}

vector_t vector_create(int N)
{
	return vector_readN(NULL, N);
}

vector_t vector_read(FILE *stream)
{
	int n;
	fscanf(stream, "%d", &n);
	return vector_readN(stream, n);
}

```

```
vector_t vector_readN(FILE *stream, int n)
{
	number_t *vect = (number_t *) malloc(n * sizeof (number_t));

	for (int i = 0; i < n; i++) {
		if (stream) {
			fscanf(stream, NUMBER_READ_FORMAT, &vect[i]);
		} else {
			vect[i] = 0;
		}
	}

	vector_t v = {
		.size = n,
		.vector = vect
	};

	return v;
}

void vector_print(FILE *stream, vector_t v, format_t f)
{
	if (f == FORMAT_LATEX)
		fprintf(stream, "$$\\begin{pmatrix}\n");

	for (int i = 0; i < v.size; i++) {
		fprintf(stream, NUMBER_WRITE_FORMAT " ", v.vector[i]);

		if (f == FORMAT_LATEX)
			fprintf(stream, " \\\\\n");
	}

	if (f == FORMAT_LATEX)
		fprintf(stream, "\\end{pmatrix}$$");

	fputc('\n', stream);
}

vector_t vector_copy(vector_t source)
{
	vector_t ret = vector_create(source.size);

	for (int i=0; i<ret.size; i++) {
		ret.vector[i] = source.vector[i];
	}

	return ret;
}

void vector_exchangeElems(vector_t v, int a, int b)
{
	number_t tmp = v.vector[a];
	v.vector[a] = v.vector[b];
	v.vector[b] = tmp;
}

void matrix_exchangeRows(matrix_t m, int a, int b)
{
	unsigned int tmp = m.rows[a];
	m.rows[a] = m.rows[b];
	m.rows[b] = tmp;
```

```
}

void matrix_swapCols(matrix_t m, int a, int b)
{
	unsigned int tmp = m.cols[a];
	m.cols[a] = m.cols[b];
	m.cols[b] = tmp;
}

void matrix_mulMatVector(matrix_t m, vector_t f, vector_t result)
{
	for (int i=0; i<result.size; i++) {
		number_t sum = 0;

		for (int j=0; j<result.size; j++) {
			sum += m.matrix[m.rows[i]][m.cols[j]] * f.vector[j];
		}

		result.vector[i] = sum;
	}
}

void vector_free(vector_t v)
{
	free(v.vector);
}

void vector_sub(vector_t a, vector_t b, vector_t result)
{
	for (int i=0; i<result.size; i++) {
		result.vector[i] = a.vector[i] - b.vector[i];
	}
}

number_t vector_norm(vector_t a)
{
	number_t sum = 0;

	for (int i=0; i<a.size; i++) {
		sum += a.vector[i] * a.vector[i];
	}

	return sqrt(sum);
}

void matrix_free(matrix_t m)
{
	for (unsigned int i=0; i<m.size; i++)
		free(m.matrix[i]);
	free(m.matrix);
	free(m.rows);
	free(m.cols);
}
```

Листинг 6: Исходный код для генерации матрицы по формуле 1 (файл input1.c)

```
#include "input.h"

#include <stdio.h>
#include <stdlib.h>

```

```
#define M 8
#define N 20

void input_form1(matrix_t *m, vector_t *f)
{
	input_form1_m(m);

	*f = vector_create(N);
	for (int i=0; i<N; i++) {
		f->vector[i] = 200 + 50 * (i + 1);
	}

	if (N <= 10) {
		fprintf(stderr, "[INPUT] Vector:\n");
		vector_print(stderr, *f, FORMAT_TEXT);
	}
}

void input_form1_m(matrix_t *m)
{
	*m = matrix_create(N);

	for (int i=0; i<N; i++) {
		for (int j=0; j<N; j++) {
			if (i == j) {
				m->matrix[i][j] = (number_t) N + (number_t) M*M + (
    number_t) (j + 1) / M + (number_t) (i + 1) / N;
			} else {
				m->matrix[i][j] = (number_t) (i + j + 2) / (M + N);
			}
		}
	}

	if (N <= 10) {
		fprintf(stderr, "[INPUT] Matrix:\n");
		matrix_print(stderr, *m, FORMAT_TEXT);
	}


}
```

Листинг 7: Исходный код для генерации матрицы по формуле 2 (файл input2.c)

```
#include "input.h"

#include <stdio.h>
#include <stdlib.h>

#include <math.h>

#define N 100
#define M 4

void input_form2(matrix_t *m, vector_t *f)
{
	input_form2_m(m);

	*f = vector_create(N);

	number_t x;
	fprintf(stderr, "[INPUT] Type X:\n");
```

```
        fscanf(stdin, NUMBER_READ_FORMAT, &x);

        for (int i=0; i<N; i++) {
                f->vector[i] = (number_t) N * exp(x / (i + 1)) * cos(x);
        }

        if (N <= 20) {
                fprintf(stderr, "[INPUT] Vector:\n");
                vector_print(stderr, *f, FORMAT_TEXT);
        }
}

void input_form2_m(matrix_t *m)
{
        *m = matrix_create(N);

        number_t qm = 1.001 - 2 * M * 0.001;

        for (int i=0; i<N; i++) {
                for (int j=0; j<N; j++) {
                        if (i == j) {
                                m->matrix[i][j] = pow(qm - 1, i + j + 2);
                        } else {
                                m->matrix[i][j] = pow(qm, i + j + 2) + 0.1 * (j - i);
                        }
                }
        }

        if (N <= 20) {
                fprintf(stderr, "[INPUT] Matrix:\n");
                matrix_print(stderr, *m, FORMAT_TEXT);
        }
}
```