Дисциплина: Численные методы
Лабораторное задание №2

Отчет
Тема: Применение точных методов решения систем линейных алгебраических уравнений

Выполнили:
студенты 3 курса 8 группы
Крутько А.С.
Сикарев Р.О.

Проверила:
старший преподаватель
Фролова О.А.

Оглавление

Постановка задачи

Вариант 3. Метод Халецкого для решения СЛАУ с симметричными ленточными матрицами.

Входные параметры основной процедуры:

N, L – размерность системы и половина ширины ленты матрицы;

A – массив размерности, N L × содержащий нижнюю часть ленты матрицы исходной системы уравнений;

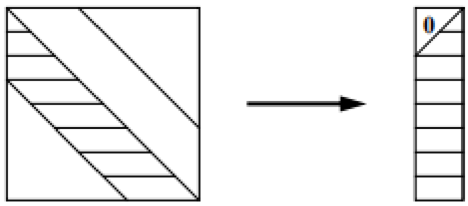
f – вектор правой части системы размерности N.

Выходные параметры основной процедуры:

IER – код завершения;

x – вектор решения размерности N.

Символическое изображение схемы хранения ленточной матрицы:

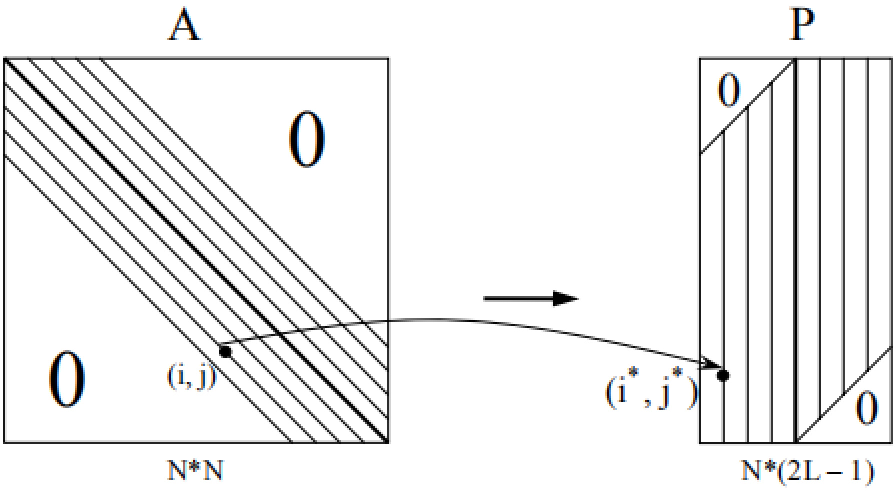


N*N N*L

При численной реализации недопустимо использование матриц размерности. NxN

Методы решения

Для хранения ленточной матрицы размерности $N * N$ с шириной ленты L была использована половина прямоугольной матрицы размерности $N * (2L - 1)$



Где связь между $a(i,j) \in A$ и $p(i,j) \in P$ определяется следующим образом:

$$\begin{cases} i^* = i \\ j^* = j - i + L \end{cases}$$

Поскольку мы работаем с симметричными матрицами, мы храним только половину ленты и используем прямоугольные массивы размера NxL (где L - половина ширины ленты + главная диагональ исходной матрицы А)

Для ленточной матрицы, удовлетворяющей условиям LU-теоремы, использовались следующие формулы:

$$b_{ij} = a_{ij} - \sum_{k=k0(i)}^{j-1} \frac{b_{ik} b_{jk}}{b_{kk}}, i = 1:N,\ j = i:kN(i)$$

$$y_i = \left(f_i - \sum_{k=k0(i)}^{j-1} b_{ik} y_k\right) \Big/ b_{ii}, i = 1:N$$

$$x_i = y_i - \frac{\left(\sum_{k=i+1}^{kN(i)} b_{ki} x_k\right)}{b_{ii}}, i = N:1$$

Где $k0(i)$ и $kN(i)$ - Вспомогательные функции для вычисления границ ненулевых элементов $i -$ й строки матрицы:

$$k0(i) = \begin{cases} 1, & \text{если } i \le L \\ i - L + 1, & \text{если } i > L \end{cases}$$

$$kN(i) = \begin{cases} i + L - 1, & \text{если } i \le N - L \\ N, & \text{если } i > N - L \end{cases}$$

Основные процедуры

Для работы программы нужно использовать файл matrix.txt для подгрузки данных для обработки оных программой

Программа использует следующие переменные для подсчета

1. Выбор ввода (1 - ввод из файла matrix.txt, 2 - заполнение матрицы случайными данными)

2. Выбор вывода (1 - вывод в консоль, 2 - вывод в файл output.txt, 3 - аналогично второму пункту, но k раз)

3. Размерность матрицы - n

4. Отношение L/N

5. Начало диапазона случайных чисел

6. Конец диапазона случайных чисел

7. Матрица чисел если выбран ввод из файла или значение для k

Первый Этап

1. Считывание данных из входного файла

2. На основании данных, полученных из файла, определить тип ввода: брать данные из файла или заполнить матрицу случайным образом

Второй этап

1. Находим решение матрицы методом Халецкого. Т.к. наша матрица представлена в виде симметричной матрицы, то искать матрицу C для поиска решения нет смысла, таким образом наше решение выглядит следующим образом:

```
//B
    COMPLEX_DOUBLE sum = 0;
    *b_matrix_ = *matrix_;
    for (int j = 0; j < n_; j++)
    {
        for (int i = j; i <= kn(j) + 1; i++)
        {
            for (int k = k0(i); k < j; k++)
            {
                COMPLEX_DOUBLE b_ik = get_from_matrix(b_matrix_, i, k);
                COMPLEX_DOUBLE b_jk = get_from_matrix(b_matrix_, j, k);
                COMPLEX_DOUBLE b_kk = get_from_matrix(b_matrix_, k, k);
                sum += b_ik * b_jk / b_kk;
            }
            COMPLEX_DOUBLE a_ik = get_from_matrix(b_matrix_, i, j);
            //Добавление в матрицу B нового элемента
            add_to_matrix(
                b_matrix_,
                i,
                j,
                a_ik - sum
```

```
            );
            sum = 0;
        }
    }

    const auto y = new SOLUTION_VECTOR(n_);
    //Y
    for (int i = 0; i < n_; i++)
    {
        for (int k = k0(i); k < i; k++)
        {
            COMPLEX_DOUBLE b_ik = get_from_matrix(b_matrix_, i, k);
            sum += b_ik * (*y)[k];
        }
        (*y)[i] = ((*f_)[i] - sum) / get_from_matrix(b_matrix_, i, i);
        sum = 0;
    }

    //X
    for (int i = n_ - 1; i >= 0; i--)
    {
        for (int k = i + 1; k <= kn(i); k++)
        {
            COMPLEX_DOUBLE b_ki = get_from_matrix(b_matrix_, k, i);
            sum += b_ki * (*x_)[k];
        }
        COMPLEX_DOUBLE b_ii = get_from_matrix(b_matrix_, i, i);
        (*x_)[i] = (*y)[i] - sum / b_ii;
        sum = 0;
    }
```

2. На основании данных из файла определяем тип вывода: консольный или в файл

3. Вычисляем погрешность:

В силу заданных условий, выбирается следующая формула для вычисления погрешности:

$$\left|\frac{x_i - x_i^*}{x_i^*}\right|$$

Где x_i - текущее решение для СЛАУ, а x_i^* - точное решение для изначального СЛАУ методом Халецкого.

4. Для хорошо обусловленной матрицы:

```
COMPLEX_DOUBLE find_error(
    const int count,
   const int n,
   const double l_n,
   const double range_begin,
   const double range_end
)
{
    COMPLEX_DOUBLE sum_error = 0;
    for (int t = 0; t < count; t++)
    {
        const auto solution = new lab_matrix(n, static_cast<int>(l_n * n));
         solution->fill_random(range_begin, range_end);
        solution->solve();
```

```
        solution->fill_x1_and_f2(range_begin, range_end);
        sum_error += solution->find_errors(range_begin, range_end);
        delete solution;
    }

    return sum_error / static_cast<COMPLEX_DOUBLE>(count);
 }
```

Метод `solution->fill_random(range_begin, range_end);` заполняет матрицу значениями в диапазоне от `range_begin` до `range_end.`
Методы `solution->fill_x1_and_f2(range_begin, range_end);` и `solution->find_errors(range_begin, range_end);` Заполняют векторы нового решения x1 и f2 и находят погрешность соответственно.
5. Для плохо обусловленной матрицы:

```
COMPLEX_DOUBLE find_error_bad(
     const int count,
     const int n,
     const double l_n,
     const double range_begin,
     const double range_end,
     const int k
)
{
    COMPLEX_DOUBLE sum_error = 0;
    for (int t = 0; t < count; t++)
    {
        const auto solution = new lab_matrix(n, static_cast<int>(l_n * n));
        solution->fill_random_bad(
            range_begin,
            range_end,
            k
        );
        solution->solve();
        solution->fill_x1_and_f2(range_begin, range_end);
        sum_error += solution->find_errors(range_begin, range_end);
        delete solution;
    }
    return sum_error / static_cast<COMPLEX_DOUBLE>(count);
}
```

Методы, которые используются для поиска погрешности в плохо обусловленной матрице идентичны методам для поиска погрешностей для хорошо обусловленной матрицы, за исключением метода `solution->fill_random_bad(range_begin, range_end, k);` , который в отличии от `solution->fill_random(range_begin, range_end);` заполняет матрицу «плохими значениями», умножая её диагональные элементы на 10^{-k}.

Результаты вычислительных экспериментов

Данные о решении систем уравнений с ленточными матрицами

№ теста	Размерность системы	Отношение L/N	Средняя относительная погрешность решения
1	10	2/10	2.22045e-16
1	10	1/4	1.77636e-15
2	10	3/10	2.56045e-15
3	10	4/10	3.55271e-15
5	100	2/10	3.14564e-13
6	100	1/4	2.75755e-13
7	100	3/10	4.45632e-13
8	100	4/10	3.13975e-13

Данные о решении систем уравнений с хорошо обусловленными квадратными матрицами

№ теста	Размерность системы	Средняя относительная погрешность решения
1	10	3.09197e-15
2	20	8.07132e-14
3	40	3.04529e-13
4	60	6.64375e-12
5	100	3.76623e-11
6	200	5.94875e-10
7	500	7.784651e-8
8	800	2.467521e-6

Данные о решении систем уравнений с плохо обусловленными матрицами

№ теста	Порядок k	Размерность системы	Средняя относительная погрешность решения
1	2	10	4.82947e-10
2	4	10	1.19468e-10
3	6	10	8.607337e-9
4	2	100	2.47237e-06
5	4	100	6.70822e-06
6	6	100	8.36256e-06