Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей
Кафедра информатики
Дисциплина: Методы численного анализа

ОТЧЁТ

к лабораторной работе

на тему

Численное решение систем линейных алгебраических уравнений методом простых итераций и методом Зейделя

Выполнил: студент группы 053506
Слуцкий Никита Сергеевич

Проверил: Анисимов Владимир Яковлевич

Минск 2022

Оглавление

Вариант 6 (Номер в журнале – 21)

Цели выполнения задания:

- Изучить итерационные методы решения СЛАУ (метод простых итераций и метод Зейделя)
- Составить алгоритм решения указанными методами, применимый для организации вычислений на ЭВМ
- Составить программу решения СЛАУ по разработанному алгоритму
- Численно решить тестовые примеры и проверить правильность работы программы. Сравнить трудоёмкость решения методом простых итераций и методом Зейделя.

Краткие теоретические сведения:

Метод итерации – это численный и приближённый метод решения СЛАУ. Суть заключается в составлении рекуррентного соотношения с заданием начального вектора значений и нахождении по приближённому текущему значению величины следующего приближения, которое является более точным.

Метод позволяет получить значения корней системы с заданной точностью в виде предела последовательности некоторых векторов.

Пусть имеется система A*x = b. Чтобы применить итерационный метод, необходимо сначала привести систему к эквивалентному виду: x = B*x + d. Это значит, что из i-ой строки нужно выразить в строке i-ю переменную. Получили описанную матрицу. Затем выберу начальное приближение x^0, например, x^0 = (0.0, 0.0, 0.0, … , 0.0). На каждой итерации имею текущее приближение и следующее приближение.

Что дальше ? На начальном этапе текущее приближение равно выбранному x^0, а следующее – не определено (ну или также равно x^0).

Чтобы получить следующее приближение из имеющегося текущего, необходимо значения переменных из текущего приближения подставить в выраженное ранее B*x + d. Таким образом получу новый вектор x^1 с новым приближением. Далее процесс повторяется, то есть на место текущего уходит x^1 с прошлой итерации и считается новое следующее приближение. Рекуррентно можно записать так: $x^{i+1} = B * x^i + d$ (где i – номер итерации, а не степень !). При этом при i = 0 (x^0) x^0 равняется выбранному начальному приближению (я выбрал (0, 0, 0, ..., 0)).

Важно заметить, что для убеждения в сходимости этого итерационного процесса необходимо проверить достаточное условие сходимости. Оно заключается в том, что (в изначальной матрице, в матрице главных коэффициентов) в i-ой строке модуль коэффициента при i-й переменной был больше суммы модулей коэффициентов при остальных переменных в этой строке. И так для всех строк системы !

Теперь про метод Зейделя. Это “оптимизационная” разновидность метода простых итераций. На каждом этапе при вычислении $x^1[i]$ я буду подставлять значения не из предыдущего x^0, а из самого же себя. То есть при использовании переменных с номерами 0..i-1 я уже буду обращаться к уже посчитанным НОВЫМ следующим значениям. Таким образом уменьшается количество итераций. Так в тестовом примере в данной лабораторной работе количество итераций снизилось с 9 до 6.

Задание

Методом простых итераций и Зейделя найти с точностью 0.0001 численное решение системы A*x = b, где A = k * C + D, исходные данные заданы ниже:

$$C = \begin{bmatrix} 0,01 & 0 & -0,02 & 0 & 0 \\ 0,01 & 0,01 & -0,02 & 0 & 0 \\ 0 & 0,01 & 0,01 & 0 & -0,02 \\ 0 & 0 & 0,01 & 0,01 & 0 \\ 0 & 0 & 0 & 0,01 & 0,01 \end{bmatrix}, \quad D = \begin{bmatrix} 1,33 & 0,21 & 0,17 & 0,12 & -0,13 \\ -0,13 & -1,33 & 0,11 & 0,17 & 0,12 \\ 0,12 & -0,13 & -1,33 & 0,11 & 0,17 \\ 0,17 & 0,12 & -0,13 & -1,33 & 0,11 \\ 0,11 & 0,67 & 0,12 & -0,13 & -1,33 \end{bmatrix}.$$

Вектор $\boldsymbol{b} = (1,2;\ 2,2;\ 4,0;\ 0,0;\ -1,2)^{T}$.

Программная реализация:

Ниже можно ознакомиться с программной реализацией главных функций по триангуляции, обратному ходу, а также вызов этого всего из главного файла. Вспомогательные функции и перегруженные операторы (реализации) опущены.

```
bool CheckConvergence(const std::vector<std::vector<double>> &main_coefficients)
{
    for (std::size_t row = 0; row < main_coefficients.size(); ++row)
    {
        const auto main_absolute{std::abs(main_coefficients.at(row).at(row))};
        std::size_t counter{0};
        const auto rest_sum_absolute{std::accumulate(
                std::begin(main_coefficients.at(row)),
                std::end(main_coefficients.at(row)),
                0.00,
                [&](const double response, const double current) -> double {
                    return counter++ != row ? response + std::abs(current) : response;
                })};

        if (main_absolute < rest_sum_absolute)
            return false;
    }

    return true;
}
-----------------------------------------------------------------------------------------------
std::vector<std::vector<double>> ExpressMainVariables(const std::vector<std::vector<double>>
&main_coefficients,
                                                      const std::vector<double> &free_coefficients)
{
    std::vector<std::vector<double>> response{};
    response.resize(main_coefficients.size());

    for (std::size_t row = 0; row < main_coefficients.size(); ++row)
    {
        const auto full_width{main_coefficients.at(row).size() + 1};
        const auto current_variable_ratio{main_coefficients.at(row).at(row)};
        response.at(row).resize(full_width);
```

```
        response.at(row).at(0) = free_coefficients.at(row) / current_variable_ratio;

        for (std::size_t col = 1; col < full_width; ++col)
            response.at(row).at(col) =
                    col - 1 != row
                    ? -1 * main_coefficients.at(row).at(col - 1) / current_variable_ratio
                    : 0;
    }

    return response;
}
-----------------------------------------------------------------------------------------------------
double GetError(const std::vector<double> &current_variables_set,
                const std::vector<double> &previous_variables_set)
{
    double error{0.00};
    for (std::size_t counter = 0; counter < current_variables_set.size(); ++counter)
        error = std::max(error, std::abs(current_variables_set.at(counter) -
previous_variables_set.at(counter)));
    return error;
}
-----------------------------------------------------------------------------------------------------
std::pair<std::vector<double>, std::size_t>
SolveBySimpleIterations(const std::vector<std::vector<double>> &main_coefficients,
                        const std::vector<double> &free_coefficients,
                        const double epsilon,
                        const std::vector<double> &initial_values,
                        const SolvingType &type)
{
    if (!CheckConvergence(main_coefficients))
        throw std::invalid_argument("System of linear algebraic equations does not converge");

    const auto check_error([](const std::vector<double> &current_variables_set,
                              const std::vector<double> &previous_variables_set,
                              const double epsilon) -> bool {
        return GetError(current_variables_set, previous_variables_set) <= std::abs(epsilon);
    });

    const auto copy_vectors([](const std::vector<double> &from, std::vector<double> &to) -> void {
        for (std::size_t counter = 0; counter < from.size(); ++counter)
            to.at(counter) = from.at(counter);
    });

    const auto count_variable_value([](const std::vector<double> &variable_expression,
                                       const std::vector<double> &variables_set) -> double {
        double response{variable_expression.at(0)};
        for (std::size_t counter = 1; counter < variable_expression.size(); ++counter)
            response += variable_expression.at(counter) * variables_set.at(counter - 1);
        return response;
    });

    auto previous_iteration{initial_values}; // предыдущая i-я итерация
    auto current_iteration{initial_values}; // текущая i+1-я итерация
    const auto variables_expressions{
            ExpressMainVariables(main_coefficients, free_coefficients)}; // выраженные переменные с главной
диагонали

    std::size_t iterations_count{0};
    do
    {
        ++iterations_count;
        copy_vectors(current_iteration, previous_iteration);
        for (std::size_t counter = 0; const auto &variable_expression : variables_expressions)
        {
            current_iteration.at(counter) = count_variable_value(
                    variable_expression,
                    type == SolvingType::kSimpleIterations ? previous_iteration : current_iteration
            );
            ++counter;
        }
    } while (!check_error(current_iteration, previous_iteration, epsilon));
```

```
    return {current_iteration, iterations_count};
}
-----------------------------------------------------------------------------------------------------
int main()
{
    const auto main_coefficients{kMatrixC * kOption + kMatrixD};
    const auto &free_coefficients{kVectorB};
    const auto &initial_values{kInitialValues};
    const auto &epsilon{kAccuracy};

    const auto[solution, number_of_iterations]{SolveBySimpleIterations(
            main_coefficients,
            free_coefficients,
            epsilon,
            initial_values,
            SolvingType::kSeidel
    )};

    std::cout << "Solution: " << std::setprecision(6) << solution;
    std::cout << "Used " << number_of_iterations << " iterations";

    return 0;
}
```

Полученные результаты:

```
Solution: 1.2085 -1.77305 -2.93219 0.14108 -0.170624
Used 9 iterations
Process finished with exit code 0
```

```
Solution: 1.20851 -1.77305 -2.93219 0.141072 -0.170664
Used 6 iterations
Process finished with exit code 0
'
```

Выводы

Метод простой итерации или метод Зейделя применяется для нахождения корней системы линейных уравнений с заданной точностью. В физике может быть полезно и, главное, быстро.