Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ: «Информатика и системы управления»
КАФЕДРА: «Теоретическая информатика и компьютерные технологии»

Лабораторная работа №7
«Методы быстрого умножения матриц»

по курсу

«Численные методы линейной алгебры»

Выполнил:
студент группы ИУ9-72Б
Караник А.А.

Проверено:
Посевин Д.П.

Москва, 2024

Цель работы

1. Реализовать алгоритм Винограда. Реализовать метод Штрассена.
2. Сравнить точность результата со стандартным алгоритмом умножения.
3. Построить на одном графике зависимость времени t (сек) умножения двух матриц размера N x N стандартным алгоритмом, алгоритмом Винограда и методом Штрассена от размера матрицы N. N изменяется от 2 до 400.

Реализация

Исходный код программы:

```
using LinearAlgebra
using Random
using Statistics
using Plots

function standard_multiply(A, B)
    m, n = size(A)
    nB, p = size(B)

    if n != nB
        error("Матрицы несовместимы для умножения")
    end

    C = zeros(Float64, m, p)

    for i in 1:m
        for j in 1:p
            for k in 1:n
                C[i, j] += A[i, k] * B[k, j]
            end
        end
    end

    return C
end

function winograd_multiply(A, B)
    m, n = size(A)
    nB, p = size(B)
    C = zeros(Float64, m, p)
    if n != nB
        error("Матрицы должны быть совместимы для умножения")
    end

    rowFactor = [sum(A[i, 1:2:n-1] .* A[i, 2:2:n]) for i in 1:m]
    colFactor = [sum(B[1:2:n-1, j] .* B[2:2:n, j]) for j in 1:p]

    for i in 1:m
        for j in 1:p
            C[i, j] = -rowFactor[i] - colFactor[j]
            for k in 1:2:n-1
                C[i, j] += (A[i, k] + B[k + 1, j]) * (A[i, k + 1] + B[k, j])
            end
        end
    end

    if isodd(n)
        for i in 1:m
            for j in 1:p
```

```
                C[i, j] += A[i, n] * B[n, j]
            end
        end
    end

    return C
end

function pad_to_power_of_two(A)
    n, m = size(A)
    new_size = 2^ceil(Int, log2(max(n, m)))
    padded_matrix = zeros(eltype(A), new_size, new_size)
    padded_matrix[1:n, 1:m] .= A
    return padded_matrix, n, m
end

function strassen_multiply(A, B; max_depth=64)
    A, orig_n, orig_m = pad_to_power_of_two(A)
    B, orig_p, orig_q = pad_to_power_of_two(B)

    if size(A, 2) != size(B, 1)
        error("Матрицы несовместимы для умножения")
    end

    function strassen_recursive(A, B, depth)
        n = size(A, 1)

        if n == 1 || depth > max_depth
            return A * B
        end

        n2 = div(n, 2)
        A11, A12 = A[1:n2, 1:n2], A[1:n2, n2+1:end]
        A21, A22 = A[n2+1:end, 1:n2], A[n2+1:end, n2+1:end]
        B11, B12 = B[1:n2, 1:n2], B[1:n2, n2+1:end]
        B21, B22 = B[n2+1:end, 1:n2], B[n2+1:end, n2+1:end]

        M1 = strassen_recursive(A11 + A22, B11 + B22, depth + 1)
        M2 = strassen_recursive(A21 + A22, B11, depth + 1)
        M3 = strassen_recursive(A11, B12 - B22, depth + 1)
        M4 = strassen_recursive(A22, B21 - B11, depth + 1)
        M5 = strassen_recursive(A11 + A12, B22, depth + 1)
        M6 = strassen_recursive(A21 - A11, B11 + B12, depth + 1)
        M7 = strassen_recursive(A12 - A22, B21 + B22, depth + 1)

        M1, M2, M3, M4, M5, M6, M7 = fetch(M1), fetch(M2), fetch(M3), fetch(M4),
         ↪  fetch(M5), fetch(M6), fetch(M7)

        C11 = M1 + M4 - M5 + M7
        C12 = M3 + M5
        C21 = M2 + M4
        C22 = M1 - M2 + M3 + M6

        C = [C11 C12; C21 C22]
        return C
    end

    C_padded = strassen_recursive(A, B, max_depth)

    return C_padded[1:orig_n, 1:orig_q]
end

function compare_accuracy(A, B)
    standard_result = standard_multiply(A, B)
    winograd_result = winograd_multiply(A, B)
```

```
    strassen_result = strassen_multiply(A, B)

    println("Ошибка Винограда:", norm(standard_result - winograd_result))
    println("Ошибка Штрассена:", norm(standard_result - strassen_result))
end

function benchmark_algorithms()
    Ns = 2:10:400
    standard_times = []
    winograd_times = []
    strassen_times = []

    for N in Ns
        A = rand(N, N)
        B = rand(N, N)

        t = @elapsed standard_multiply(A, B)
        push!(standard_times, t)

        t = @elapsed winograd_multiply(A, B)
        push!(winograd_times, t)

        t = @elapsed strassen_multiply(A, B)
        push!(strassen_times, t)

        compare_accuracy(A, B)
    end

    plot(Ns, standard_times, label="Стандартное умножение", lw=2)
    plot!(Ns, winograd_times, label="Алгоритм Винограда", lw=2)
    plot!(Ns, strassen_times, label="Метод Штрассена", lw=2)
    xlabel!("Размер матрицы (N)")
    ylabel!("Время (сек)")
    title!("Сравнение времени выполнения алгоритмов умножения")
end

benchmark_algorithms()
```

Результаты

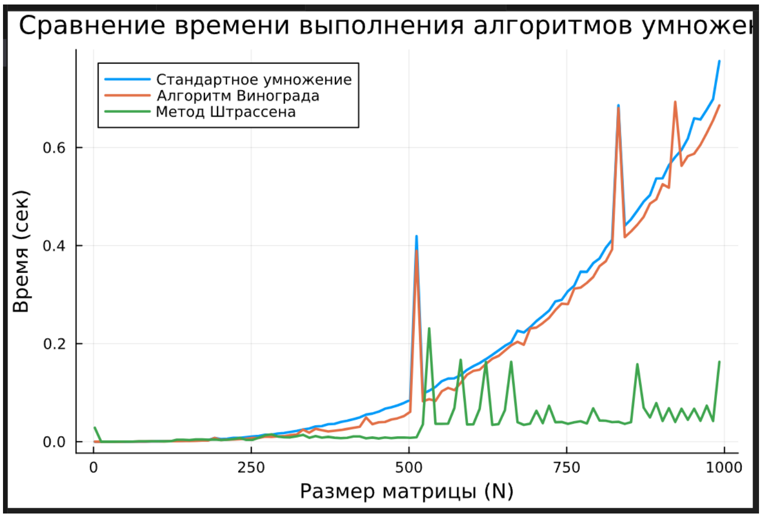

Рис. 1: результаты

Вывод

В ходе лабораторной работы были реализованы стандартный алгоритм умножения, алгоритм Винограда и метод Штрассена, проведено сравнение точности вычислений, а также построен график зависимости времени выполнения от размера матриц для каждого алгоритма. Результаты показали, что метод Штрассена работает эффективнее стандартного метода и метода Винограда на больших матрицах.