

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ**  
**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ  
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**  
**НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ  
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**  
**Факультет информационных технологий**  
**Кафедра параллельных вычислений**

**ОТЧЕТ**

**О ВЫПОЛНЕНИИ ПРАКТИЧЕСКОЙ РАБОТЫ**

**ВЕКТОРИЗАЦИЯ ВЫЧИСЛЕНИЙ**

Студентки 2 курса, группы 21205

**Евдокимовой Дари Евгеньевны**

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:  
Кандидат технических наук, доцент  
А.Ю. Власенко

Новосибирск 2022

## СОДЕРЖАНИЕ

ЦЕЛЬ.....	3
ЗАДАНИЕ .....	3
ОПИСАНИЕ РАБОТЫ .....	4
ЗАКЛЮЧЕНИЕ .....	7
Приложение 1. Листинг программы без векторизации .....	8
Приложение 2. Тесты, проведенные для программы без векторизации матрицы .....	12
Приложение 3. Листинг программы с использованием встроенных SIMD функций компилятора.....	14
Приложение 4. Тесты, проведенные для программы с ручной векторизацией при использовании SIMD-функций .....	19
Приложение 5. Листинг программы при использовании библиотеки CBLAS .....	21
Приложение 6. Тесты, проведенные для программы при использовании библиотеки CBLAS .....	25

## ЦЕЛЬ

1. Ознакомление с различными видами векторизации.
2. Изучение SIMD-расширений архитектуры x86/x86-64.
3. Изучение способов использования SIMD-расширений в программах на языке Си.
4. Получение навыков использования SIMD-расширений.
5. Изучение библиотеки CBLAS для использования векторизации расширений в программах на языке Си.
6. Получение навыков использования библиотеки CBLAS.

## ЗАДАНИЕ

1. Реализовать программу, вычисляющую матрицу, обратную данной (см. Рис. 1)

Алгоритм обращения матрицы  $A$  размером  $N \times N$  с помощью разложения в ряд:  $A^{-1} = (I + R + R^2 + \dots)B$ , где  $R = I - BA$ ,  $B = \frac{A^T}{\|A\|_1 \cdot \|A\|_\infty}$ ,  $\|A\|_1 = \max_j \sum_i |A_{ij}|$ ,  $\|A\|_\infty = \max_i \sum_j |A_{ij}|$ ,  $I$  – единичная матрица (на главной диагонали – единицы, остальные – нули). Параметры алгоритма:  $N$  – размер матрицы,  $M$  – число членов ряда (число итераций цикла в реализации алгоритма).

Рис. 1

2. Написать три варианта программы, реализующей вышеприведенный алгоритм:
  - 2.1 вариант без векторизации;
  - 2.2 вариант с ручной векторизацией (выбрать любой вариант из возможных трех: ассемблерная вставка, встроенные функции компилятора, расширение GCC);
  - 2.3 вариант с матричными операциями, выполненными с использованием оптимизированной библиотеки BLAS. Для элементов матриц использовать тип данных float. Использовать минимум 2 BLAS-функции.
3. Проверить правильность работы программ на нескольких небольших тестовых наборах входных данных.
4. Каждый вариант программы оптимизировать по скорости, насколько это возможно.
5. Сравнить время работы трех вариантов программы для  $N=2048$ ,  $M=10$ .

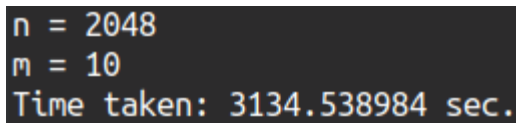
## ОПИСАНИЕ РАБОТЫ

1. Был создан файл `novect.c`, в котором был реализован алгоритм построения матрицы обратной данной и добавлено время измерения программы с помощью функции `clock_gettime()`. Полный компилируемый листинг программы см. Приложение 1.

Команда для компиляции:

```
gcc -o novect novect.c  
./novect
```

2. Для того, чтобы проверить корректность работы программы, было запущено несколько тестовых вариантов с выводом изначальной матрицы и конечной матрицы, см. Приложение 2. Полученные результаты можно сравнить с данными с сайта онлайн-калькулятора матриц (URL: <https://matrixcalc.org/>). По результатам сравнения можно сделать вывод о том, что реализованная программа работает корректно, порядок каждого числа совпадает.
3. Данная программа (`novect.c`) была запущена с данными  $n = 2048$ ,  $m = 10$ . Затраченное время на преобразование матрицы см. Рис. 2.



```
n = 2048  
m = 10  
Time taken: 3134.538984 sec.
```

Рис.2

4. Был создан файл `withvect.c`, в котором был реализован алгоритм построения матрицы обратной данной и добавлено время измерения программы с помощью функции `clock_gettime()`. Здесь использовались встроенные функции SIMD-функций компилятора. Полный компилируемый листинг программы см. Приложение 3.

Команда для компиляции:

```
gcc -o withvect withvect.c  
./withvect
```

5. Результаты, полученные программой, были сравнены с результатами, полученными с сайта, см. Приложение 4.
6. Данная программа (`withvect.c`) была запущена с данными  $n = 2048$ ,  $m = 10$ . Затраченное время на преобразование матрицы см. Рис. 3.

```
n = 2048
m = 10
Time taken: 1105.743701 sec.
```

Рис.3

7. Был создан файл blas.c, в котором был реализован алгоритм построения матрицы обратной данной и добавлено время измерения программы с помощью функции clock\_gettime(). Здесь использовались встроенные функции SIMD-функций компилятора. Полный компилируемый листинг программы см. Приложение 5. Команда для компиляции:

```
gcc -o blas blas.c -lblas
./blas
```

8. Данная программа (blas.c) была запущена с данными  $n = 2048$ ,  $m = 10$ . Затраченное время на преобразование матрицы см. Рис. 4.

```
n = 2048
m = 10
Time taken: 2.186288 sec.
```

Рис. 4

9. В программе были использованы следующие функции CBLAS-а:
- cblas\_sasum, вычисляющая сумму абсолютных значений в векторе
  - cblas\_isamax, вычисляющая индекс вектора, в котором лежит наибольшее значение
  - cblas\_saxpy, вычисляющая сумму двух матриц
  - cblas\_sgemm, умножающая 2 матрицы

10. Результаты сравнения работы трёх программ представлены в таблице 1.

Таблица 1. Результаты сравнения работы программ

	Без векторизации	Ручная оптимизация	Библиотека BLAS
Время работы (при $n = 2048$ , $m = 10$ )	~52,2 мин	~18,1 мин	~2,1 сек
Точность вычислений	Совпадают минимум 5 знаков после запятой	Совпадают минимум 6 знаков после запятой	Совпадают минимум 5 знаков после запятой

Стоит отметить, что при небольших значениях  $m$  обратная матрица считается с неточностями. То есть чем больше значение  $m$  (количество итераций цикла), тем точнее программа выдаст результат.

## ЗАКЛЮЧЕНИЕ

В ходе работы были изучены основные принципы работы с встроенными функциями SIMD и с библиотекой CBLAS.

Так же были получены данные об измерении скорости работы вычисления алгоритма по преобразованию матриц. Алгоритм включает в себя стандартные операции работы с матрицами, такие как транспонирование, сложение и вычитание матриц, умножение на скаляр, и в том числе – перемножения матриц, на которое требуется много времени для вычисления. Уменьшение времени работы программы достигается за счет ручной векторизации и функций из библиотеки CBLAS.

Программа с функциями SIMD работает почти в 3 раза быстрее программы без векторизации. Программа с CBLAS работает в  $(52 * 60 / 2.8) = 1114$  раза быстрее, чем без векторизации и в  $(18 * 60 / 2.8) = 385$  раз быстрее, чем с функциями SIMD.

По результатам сравнения работы трёх программ можно сделать вывод о том, что быстрее всего с работой с матрицами справляется библиотека CBLAS.

## Приложение 1. Листинг программы без векторизации

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

#define n 2048
#define m 10

void FillMatrix(float *A){
    srand(time(NULL));
    for (int i = 0; i < n * n; i++) {
        A[i] = rand() % 100;
    }
}

void PrintMatrix(float *matrix) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            printf("%.9f\t", matrix[i * n + j]);
        }
        printf("\n");
    }
    printf("=====\n");
}

void IdentityMatrix(float *matrix) {
    for (int i = 0; i < n; i++) {
        matrix[i * n + i] = 1;
    }
}

float CountMaxRowSum(float *matrix){
    float maxRow = 0;
    for (int i = 0; i < n; i++) {
        float tmp = 0;
        for (int j = 0; j < n; j++) {
            tmp += fabs(matrix[i * n + j]);
        }
        if (tmp > maxRow) {
            maxRow = tmp;
        }
    }
    return maxRow;
}

float CountMaxColumnSum(float *matrix){
    float maxColumn = 0;
```



```

    for (int i = 0; i < n; i++) {
        float tmp = 0;
        for (int j = 0; j < n; j++) {
            tmp += fabs(matrix[j * n + i]);
        }
        if (tmp > maxColumn) {
            maxColumn = tmp;
        }
    }
    return maxColumn;
}

void TransponateMatrix(float *transpMatr, float *matr){
    for (int i = 0; i < n; i++){
        for (int j = 0; j < n; j++){
            transpMatr[i * n + j] = matr[i * n + j];
        }
    }
}

float SubMatrixes(float *matrix1, float *matrix2, float *res){
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++){
            res[i * n + j] = matrix1[i * n + j] - matrix2[i * n + j];
        }
    }
}

void SumMatrixes(float *matrix, float *store){
    for (int i = 0; i < n * n; i++) {
        store[i] += matrix[i];
    }
}

void MultMatrixOnScalar(float *matrix, float scalar){
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            matrix[i * n + j] = matrix[i * n + j] * scalar;
        }
    }
}

void MultMatrixes(float *matrix1, float *matrix2, float *result) {
    for (int i = 0; i < n; i++) {
        for (int k = 0; k < n; k++) {
            for (int j = 0; j < n; j++) {
                result[i * n + j] += matrix1[i * n + k] * matrix2[k * n + j];
            }
        }
    }
}

```

```

void InverseMatrix(float *A) {
    float *B = (float*) calloc(n * n, sizeof(float));
    TransponateMatrix(B, A);

    float maxRow = CountMaxRowSum(A);

    float maxColumn = CountMaxColumnSum(A);

    float scalar = 1 / (maxColumn * maxRow);

    float *I = (float*) calloc(n * n, sizeof(float));
    IdentityMatrix(I);

    MultMatrixOnScalar(B, scalar);

    float *R = (float*) calloc(n * n, sizeof(float));
    MultMatrixes(B, A, R); // R contains B * A
    SubMatrixes(I, R, R); // R contatins I - BA

    float *temp = (float*) calloc(n * n, sizeof(float));
    memcpy(temp, R, sizeof(float) * n * n); //tmp contains R
    for (int iter = 0; iter < m; iter++) {
        SumMatrixes(temp, I); //I + R + R^2 + R^3 + ...
        memset(A, 0, sizeof(float) * n * n); //A consist of zeroes

        MultMatrixes(temp, R, A); //A contains R, R^2, R^3, R^4, ...
        memcpy(temp, A, sizeof(float) * n * n); //temp contains A, i.e R, R^2,
R^3, R^4, ...
    }
    memset(A, 0, sizeof(float) * n * n); //A consist of zeroes
    MultMatrixes(I, B, A); // A consist of (sum it bractets, i.e I) * B

    free(temp);
    free(I);
    free(B);
    free(R);
}

int main() {
    float *A = (float*) calloc(n * n, sizeof(float));
    FillMatrix(A);

    printf("n = %d\n", n);
    printf("m = %d\n", m);

    printf("Start matrix:\n");
    PrintMatrix(A);

    struct timespec start, end;
    clock_gettime(CLOCK_MONOTONIC_RAW, &start);

```

```
InverseMatrix(A);
clock_gettime(CLOCK_MONOTONIC_RAW, &end);

printf("Result matrix:\n");
PrintMatrix(A);

printf("Time taken: %lf sec.\n", end.tv_sec - start.tv_sec +
      0.00000001*(end.tv_nsec-start.tv_nsec));

free(A);
}
```

## Приложение 2. Тесты, проведенные для программы без векторизации матрицы

```
dasha@dasha-K501UQ:~/masec/evm/pract4$ gcc -o novect novect.c
dasha@dasha-K501UQ:~/masec/evm/pract4$ ./novect
n = 3
m = 10000
Start matrix:
5.0000000000    53.0000000000    20.0000000000
85.0000000000    79.0000000000    84.0000000000
85.0000000000    48.0000000000    99.0000000000
=====
Result matrix:
-0.037392601    0.042307511    -0.028342873
0.012582660    0.011892043    -0.012631819
0.026004009    -0.042090386    0.040560201
=====
Time taken: 0.013567 sec.
```

Рис 1. Тест1, результаты программы

$$\begin{pmatrix} 5.0000000000 & 53.0000000000 & 20.0000000000 \\ 85.0000000000 & 79.0000000000 & 84.0000000000 \\ 85.0000000000 & 48.0000000000 & 99.0000000000 \end{pmatrix}^{(-1)} =$$
$$\begin{pmatrix} -0.0373926774 & 0.0423073127 & -0.0283430376 \\ 0.0125826507 & 0.0118918385 & -0.0126319945 \\ 0.0260041449 & -0.0420902003 & 0.0405605448 \end{pmatrix}$$

Рис 2. Тест1, результаты с сайта

```
n = 5
m = 1000
Start matrix:
73.0000000000    90.0000000000    70.0000000000    3.0000000000    62.0000000000
68.0000000000    51.0000000000    33.0000000000    50.0000000000    50.0000000000
43.0000000000    3.0000000000    83.0000000000    44.0000000000    6.0000000000
13.0000000000    69.0000000000    42.0000000000    50.0000000000    80.0000000000
20.0000000000    94.0000000000    26.0000000000    64.0000000000    64.0000000000
=====
Result matrix:
0.002048107    0.017548889    -0.002982082    -0.007808061    -0.005655145
0.007563250    -0.010828510    -0.000651196    -0.015693050    0.020812150
0.004918671    -0.012777266    0.011162193    0.004466712    -0.001412161
-0.011487495    0.006439902    0.005779088    -0.003420600    0.009831373
-0.002258908    0.009171441    -0.008425709    0.027096868    -0.022434562
=====
Time taken: 0.004095 sec.
```

Рис 3. Тест2, результаты программы

$$\begin{pmatrix} 73.000000000 & 90.000000000 & 70.000000000 & 3.000000000 & 62.000000000 \\ 68.000000000 & 51.000000000 & 33.000000000 & 50.000000000 & 50.000000000 \\ 43.000000000 & 3.000000000 & 83.000000000 & 44.000000000 & 6.000000000 \\ 13.000000000 & 69.000000000 & 42.000000000 & 50.000000000 & 80.000000000 \\ 20.000000000 & 94.000000000 & 26.000000000 & 64.000000000 & 64.000000000 \end{pmatrix}^{(-1)}$$

Insert in A

Insert in B

Clean

$$= \begin{pmatrix} 0.00204827692 & 0.0175493242 & -0.00298260527 & -0.00780624461 & -0.00565725281 \\ 0.00756250617 & -0.0108287608 & -0.000650267717 & -0.0156966012 & 0.0208155057 \\ 0.00491863871 & -0.0127776298 & 0.0111624488 & 0.00446577899 & -0.00141111130 \\ -0.0114877935 & 0.00644051756 & 0.00577908370 & -0.00342050013 & 0.00983098166 \\ -0.00225792097 & 0.00917097319 & -0.00842668366 & 0.0271001119 & -0.0224376002 \end{pmatrix}$$

Рис 4. Тест2, результаты с сайта

### Приложение 3. Листинг программы с использованием встроенных SIMD функций компилятора

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <xmmintrin.h>

#define n 2048
#define m 10

void FillMatrix(float *A){
    srand(time(NULL));
    for (int i = 0; i < n * n; i++) {
        A[i] = rand() % 100;
    }
}

void PrintMatrix(float *matrix) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            printf("%.9f\t", matrix[i * n + j]);
        }
        printf("\n");
    }
    printf("=====\n");
}

void IdentityMatrix(float *matrix) {
    for (int i = 0; i < n; i++) {
        matrix[i * n + i] = 1;
    }
}

float CountMaxRowSum(float *A){
    float maxRow = 0;
    for (int i = 0; i < n; i++) {
        float tmp = 0;
        for (int j = 0; j < n; j++) {
            tmp += fabs(matrix[i * n + j]);
        }
        if (tmp > maxRow) {
            maxRow = tmp;
        }
    }
    return maxRow;
}
```

```

float CountMaxColumnSum(float *A){
    float maxColumn = 0;
    for (int i = 0; i < n; i++) {
        float tmp = 0;
        for (int j = 0; j < n; j++) {
            tmp += fabs(matrix[j * n + i]);
        }
        if (tmp > maxColumn) {
            maxColumn = tmp;
        }
    }
    return maxColumn;
}

void TransponateMatrix(float *transpA, float *A){
    for (int i = 0; i < n; i++){
        for (int j = 0; j < n; j++){
            transpA[i * n + j] = A[i * n + j];
        }
    }
}

void SubMatrixes(float *matrix1, float *matrix2, float *res){
    //fill values from matrix1 by aligned address
    __m128 maxtr1mm = _mm_load_ps(matrix1);

    //fill values from matrix2 by aligned address
    __m128 maxtr2mm = _mm_load_ps(matrix2);

    //sub values
    __m128 subMatr = _mm_sub_ps(maxtr1mm, maxtr2mm);

    //write values in res by aligned address
    _mm_store_ps(res, subMatr);
}

void SumMatrixes(float *matrix, float *store){
    for (int i = 0; i < n; i++){
        for (int j = 0; j < n; j += 4){
            //fill values from matrix by aligned address
            __m128 matrixLine = _mm_load_ps(matrix + i * n + j);

            //load values from store by aligned address
            __m128 storeLine = _mm_load_ps(store + i * n + j);

            //sum matrixes
            __m128 tmpSum = _mm_add_ps(matrixLine, storeLine);

            //write values in store by aligned address
            _mm_store_ps(store + i * n + j, tmpSum);
        }
    }
}

```

```

    }
}

void MultMatrixOnScalar(float *matrix, float scalar){
    //fill every vector component by number we wanna ultiply
    __m128 number = _mm_set1_ps(scalar);

    for (int i = 0; i < n; i++){
        for (int j = 0; j < n; j += 4){
            //fill values from matrix by aligned address
            __m128 line = _mm_load_ps(matrix + i * n + j);

            //multiply line on number and get partial sums
            __m128 result = _mm_mul_ps(line, number);

            //write values by aligned address
            _mm_store_ps(matrix + i * n + j, result);
        }
    }
}

void MultMatrixes(float *matrix1, float *matrix2, float *result) {
    for (int i = 0; i < n; i++){
        for (int k = 0; k < n; k+=4){
            __m128 sum = _mm_setzero_ps(); //zerofy values
            for (int j = 0; j < n; j++){
                //fill every vector component by value from the maxColumn of
matrix1
                __m128 line1 = _mm_set1_ps(matrix1[i * n + j]);

                //load a line from matrix2
                __m128 line2 = _mm_load_ps(matrix2 + j * n + k);

                //temporary multiplication of temp values
                __m128 tmpMult = _mm_mul_ps(line1, line2);

                //tmp sum
                sum = _mm_add_ps(sum, tmpMult);
            }
            //write values in matrix result by aligned address
            _mm_store_ps(result + i * n + k, sum);
        }
    }
}

void InverseMatrix(float *A) {
    float *B = (float*) calloc(n * n, sizeof(float));
    TransponateMatrix(B, A);

    float maxRow = CountMaxRowSum(A);

```



```

float maxColumn = CountMaxColumnSum(A);

float scalar = 1 / (maxColumn * maxRow);

float *I = (float*) calloc(n * n, sizeof(float));
IdentityMatrix(I);

MultMatrixOnScalar(B, scalar);

float *R = (float*) calloc(n * n, sizeof(float));
MultMatrixes(B, A, R); // R contains B * A

for (int i = 0; i < n * n; i+=4) {
    SubMatrixes(I + i, R + i, R + i); // R contains I - BA
}

float *temp = (float*) calloc(n * n, sizeof(float));
memcpy(temp, R, sizeof(float) * n * n); //tmp contains R
for (int iter = 0; iter < m; iter++) {
    SumMatrixes(temp, I);
    memset(A, 0, sizeof(float) * n * n); //A consist of zeroes

    MultMatrixes(temp, R, A); //A contains R, R^2, R^3, R^4, ...
    memcpy(temp, A, sizeof(float) * n * n); //temp contains A, i.e R, R^2,
R^3, R^4, ...
}
memset(A, 0, sizeof(float) * n * n); //A consist of zeroes
MultMatrixes(I, B, A); // A consist of (sum it bractets, i.e I) * B

free(temp);
free(I);
free(B);
free(R);
}

int main() {
    float *A = (float*) calloc(n * n, sizeof(float));;
    FillMatrix(A);

    printf("n = %d\n", n);
    printf("m = %d\n", m);

    printf("Start matrix:\n");
    PrintMatrix(A);

    struct timespec start, end;
    clock_gettime(CLOCK_MONOTONIC_RAW, &start);

    InverseMatrix(A);
    clock_gettime(CLOCK_MONOTONIC_RAW, &end);

```

```
printf("Result matrix:\n");
PrintMatrix(A);

printf("Time taken: %lf sec.\n", end.tv_sec - start.tv_sec +
      0.00000001*(end.tv_nsec-start.tv_nsec));

free(A);
}
```

**Приложение 4.** Тесты, проведенные для программы с ручной векторизацией при использовании SIMD-функций

```
dasha@dasha-K501UQ:~/masec/evm/pract4$ gcc -o withvect withvect.c
dasha@dasha-K501UQ:~/masec/evm/pract4$ ./withvect
n = 4
m = 100000
Start matrix:
83.0000000000    86.0000000000    77.0000000000    15.0000000000
93.0000000000    35.0000000000    86.0000000000    92.0000000000
49.0000000000    21.0000000000    62.0000000000    27.0000000000
90.0000000000    59.0000000000    63.0000000000    26.0000000000
=====
Result matrix:
-0.027576476    -0.008180536    -0.000281885    0.045150544
0.029802278     0.009536371    -0.027531460    -0.022351209
0.007585621     -0.005936297    0.035403974     -0.020137383
0.009447958     0.021060348    -0.022335574    -0.018313613
=====
Time taken: 0.137603 sec.
```

Рис 1. Тест1, результаты программы

$$\begin{pmatrix} 83.0000000000 & 86.0000000000 & 77.0000000000 & 15.0000000000 \\ 93.0000000000 & 35.0000000000 & 86.0000000000 & 92.0000000000 \\ 49.0000000000 & 21.0000000000 & 62.0000000000 & 27.0000000000 \\ 90.0000000000 & 59.0000000000 & 63.0000000000 & 26.0000000000 \end{pmatrix}^{(-1)} = \begin{pmatrix} -0.0275768756 & -0.00818079691 & -0.000282252587 & 0.0451502796 \\ 0.0298029838 & 0.00953707557 & -0.0275307319 & -0.0223509980 \\ 0.00758588866 & -0.00593614493 & 0.0354040481 & -0.0201373960 \\ 0.00944737574 & 0.0210600536 & -0.0223361199 & -0.0183138588 \end{pmatrix}$$

Рис 2. Тест1, результаты с сайта

```

dasha@dasha-K501UQ:~/masec/evm/pract4$ ./withvect
n = 8
m = 1000
Start matrix:
83.000000000 86.000000000 77.000000000 15.000000000 93.000000000 35.000000000 86.000000000 92.000000000
49.000000000 21.000000000 62.000000000 27.000000000 90.000000000 59.000000000 63.000000000 26.000000000
40.000000000 26.000000000 72.000000000 36.000000000 11.000000000 68.000000000 67.000000000 29.000000000
82.000000000 30.000000000 62.000000000 23.000000000 67.000000000 35.000000000 29.000000000 2.000000000
22.000000000 58.000000000 69.000000000 67.000000000 93.000000000 56.000000000 11.000000000 42.000000000
29.000000000 73.000000000 21.000000000 19.000000000 84.000000000 37.000000000 98.000000000 24.000000000
15.000000000 70.000000000 13.000000000 26.000000000 91.000000000 80.000000000 56.000000000 73.000000000
62.000000000 70.000000000 96.000000000 81.000000000 5.000000000 25.000000000 84.000000000 27.000000000
=====
Result matrix:
-183.524536133 172.301620483 -155.206726074 -106.910751343 66.689323425 -90.673469543 35.647552490 275.844665527
-40.790672302 56.271308899 -78.009666443 -101.422836304 45.792198181 -27.537014008 16.163902283 110.017318726
99.283348083 -106.998970032 117.293510437 117.290748596 -59.749572754 54.723148346 -25.622360229 -186.523391724
148.951629639 -56.203628540 -76.216979980 -273.872253418 89.543350220 39.192535400 9.454803467 2.367858887
-78.089202881 8.236495972 91.300338745 235.213256836 -83.475028992 -11.817169189 -14.682342529 -58.714500427
24.619079590 35.408584595 -120.666992188 -238.113571167 91.659477234 -11.900810242 22.058044434 121.346656799
-61.083431244 33.264102936 6.595619202 68.321975708 -19.209922791 -20.267704010 0.803245544 26.627834320
165.491180420 -166.071929932 165.844970703 142.659606934 -78.597290039 86.142829895 -37.021652222 -277.726776123
=====
Time taken: 0.001294 sec.

```

Рис 3. Тест2, результаты программы

$$\begin{pmatrix}
 83.000000000 & 86.000000000 & 77.000000000 & 15.000000000 & 93.000000000 & 35.000000000 & 86.000000000 & 92.000000000 \\
 49.000000000 & 21.000000000 & 62.000000000 & 27.000000000 & 90.000000000 & 59.000000000 & 63.000000000 & 26.000000000 \\
 40.000000000 & 26.000000000 & 72.000000000 & 36.000000000 & 11.000000000 & 68.000000000 & 67.000000000 & 29.000000000 \\
 82.000000000 & 30.000000000 & 62.000000000 & 23.000000000 & 67.000000000 & 35.000000000 & 29.000000000 & 2.000000000 \\
 22.000000000 & 58.000000000 & 69.000000000 & 67.000000000 & 93.000000000 & 56.000000000 & 11.000000000 & 42.000000000 \\
 29.000000000 & 73.000000000 & 21.000000000 & 19.000000000 & 84.000000000 & 37.000000000 & 98.000000000 & 24.000000000 \\
 15.000000000 & 70.000000000 & 13.000000000 & 26.000000000 & 91.000000000 & 80.000000000 & 56.000000000 & 73.000000000 \\
 62.000000000 & 70.000000000 & 96.000000000 & 81.000000000 & 5.000000000 & 25.000000000 & 84.000000000 & 27.000000000
 \end{pmatrix}^{(-1)} = \begin{pmatrix}
 -0.00323596124 & 0.00600470538 & -0.0150050488 & 0.0161444743 & -0.0171140487 & -0.0143675958 & 0.0173537353 & 0.0126382678 \\
 0.00393773984 & -0.0350256239 & 0.0173602354 & 0.00912299804 & 0.0140664094 & 0.0189093450 & -0.00934630961 & -0.0124307977 \\
 0.0105810191 & -0.0122728326 & 0.0243334875 & -0.00682169326 & 0.0200823029 & 0.0124510495 & -0.0276552832 & -0.0174012628 \\
 -0.0125312551 & 0.0240988270 & -0.0288396870 & -0.00253156278 & -0.0114723972 & -0.0181081935 & 0.0212746662 & 0.0270780265 \\
 0.0000659144686 & 0.0109056232 & -0.00878889008 & -0.00224760519 & 0.00345054302 & 0.00203304462 & -0.00245958423 & -0.00164457557 \\
 -0.00568831798 & -0.0111645385 & 0.0169552183 & 0.00857013771 & 0.00123283395 & 0.00179929293 & 0.00604621212 & -0.00857681075 \\
 -0.00135266083 & 0.0136867577 & -0.00399890234 & -0.00847065810 & -0.00888323308 & 0.00317139042 & 0.000367822413 & 0.00635666533 \\
 0.00665716771 & 0.0140957994 & -0.0121826596 & -0.0100410469 & -0.00829987145 & -0.0178868381 & 0.0126005499 & 0.00935065109
 \end{pmatrix}$$

Рис 4. Тест2, результаты с сайта

## Приложение 5. Листинг программы при использовании библиотеки CBLAS

```
#include <cblas.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#define n 2048
#define m 10

void FillMatrix(float *A){
    srand(time(NULL));
    for (int i = 0; i < n * n; i++) {
        A[i] = rand() % 100;
    }
}

void PrintMatrix(float *matrix) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            printf("%.9f\t", matrix[i * n + j]);
        }
        printf("\n");
    }
    printf("=====\n");
}

void IdentityMatrix(float *matrix) {
    for (int i = 0; i < n; i++) {
        matrix[i * n + i] = 1;
    }
}

float CountMaxRowSum(float *matrix){
    float *tmpSum = (float *) calloc(n, sizeof(float));
    for (int i = 0; i < n; i++){
        /*
        @brief counts sum of absolute values in vector
        @param n - number of elements in vector
        @param matrix + n * i - array
        @param 1 - the increment for indexing vector
        */
        tmpSum[i] = cblas_sasum(n, matrix + n * i, 1);
    }
    /*
    @brief returns thr position that has the largest
```

```

        absolute value in vector
        @param n - number of elements in vector
        @param matrix + n * i - array
        @param 1 - the increment for indexing vector
    */
    float maxSumRow = tmpSum[cblas_isamax(n, tmpSum, 1)];

    free (tmpSum);
    return maxSumRow;
}

float CountMaxColumnSum(float *matrix){
    float *tmpSum = (float *) calloc(n, sizeof(float));

    for (int i = 0; i < n; i++){
        tmpSum[i] = cblas_sasum(n, matrix + i, n);
    }
    float result = tmpSum[cblas_isamax(n, tmpSum, 1)];

    free (tmpSum);
    return result;
}

void TransponateMatrix(float *transpMatr, float *matr){
    for (int i = 0; i < n; i++){
        for (int j = 0; j < n; j++){
            transpMatr[i * n + j] = matr[i * n + j];
        }
    }
}

float SubMatrixes(float *matrix1, float *matrix2, float *res){
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++){
            res[i * n + j] = matrix1[i * n + j] - matrix2[i * n + j];
        }
    }
}

void SumMatrixes(float *matrix, float *store){
    /*
        @brief contains the sum of matrixes
        @param n * n - number of elements in vectors
        @param 1 - scalar
        @param matrix - array
        @param 1 - the increment for the elements of matrix
        @param store - array
        @param 1 - the increment for the elements of store
    */
    cblas_saxpy(n * n, 1, matrix, 1, store, 1);
}

```

```

void MultMatrixOnScalar(float *matrix, float scalar){
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            matrix[i * n + j] = matrix[i * n + j] * scalar;
        }
    }
}

void MultMatrixes(float *matrix1, float *matrix2, float *result) {
    /*
        @brief C := alpha*op(A)*op(B) + beta*C
        @param CblasRowMajor - store matrix on rows
        @param CblasNoTrans - no operations for matrix1
        @param CblasNoTrans - no operations for matrix2
        @param n, n, n - sizes of matrixes
        @param 1 - alpha - coeff
        @param matrix1
        @param n - number of elements in matrix dimension
        @param matrix2
        @param n - number of elements in matrix dimension
        @param 1 - beta - coeff
        @param result
        @param n - number of elements in matrix dimension
    */
    cblas_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,
                n, n, n,
                1, matrix1, n,
                matrix2, n, 1,
                result, n);
}

void InverseMatrix(float *A) {
    float *B = (float*) calloc(n * n, sizeof(float));
    TransponateMatrix(B, A);

    float maxRow = CountMaxRowSum(A);

    float maxColumn = CountMaxColumnSum(A);

    float scalar = 1 / (maxColumn * maxRow);

    float *IdentMatr = (float*) calloc(n * n, sizeof(float));
    IdentityMatrix(IdentMatr);

    MultMatrixOnScalar(B, scalar);

    float *R = (float*) calloc(n * n, sizeof(float));
    MultMatrixes(B, A, R); // R contains B * A
    SubMatrixes(IdentMatr, R, R); // R contains IdentMatr - BA
}

```

```

float *temp = (float*) calloc(n * n, sizeof(float));
memcpy(temp, R, sizeof(float) * n * n); //tmp contains R
for (int iter = 0; iter < m; iter++) {
    SumMatrixes(temp, IdentMatr); //IdentMatr + R + R^2 + R^3 + ...
    memset(A, 0, sizeof(float) * n * n); //A consist of zeroes

    MultMatrixes(temp, R, A); //A contains R, R^2, R^3, R^4, ...
    memcpy(temp, A, sizeof(float) * n * n); //temp contains A, i.e R, R^2,
R^3, R^4, ...
}
memset(A, 0, sizeof(float) * n * n); //A consist of zeroes
MultMatrixes(IdentMatr, B, A); // A consist of (sum it bractets, i.e
IdentMatr) * B

free(temp);
free(IdentMatr);
free(B);
free(R);
}

int main() {
float *A = (float*) calloc(n * n, sizeof(float));
FillMatrix(A);

printf("n = %d\n", n);
printf("m = %d\n", m);

// printf("Start matrix:\n");
// PrintMatrix(A);

struct timespec start, end;
clock_gettime(CLOCK_MONOTONIC_RAW, &start);

InverseMatrix(A);
clock_gettime(CLOCK_MONOTONIC_RAW, &end);

// printf("Result matrix:\n");
// PrintMatrix(A);

printf("Time taken: %lf sec.\n", end.tv_sec - start.tv_sec +
0.00000001*(end.tv_nsec-start.tv_nsec));

free(A);
}

```



## Приложение 6. Тесты, проведенные для программы при использовании библиотеки CBLAS

```
n = 3
m = 100000
Start matrix:
70.000000000  50.000000000  4.000000000
23.000000000  50.000000000  8.000000000
43.000000000  78.000000000  59.000000000
=====
Result matrix:
0.021002900  -0.023820188  0.001805929
-0.009146940  0.035739362  -0.004225855
-0.003214588  -0.029888110  0.021219652
=====
Time taken: 0.268295 sec.
```

Рис 1. Тест1, результаты программы

$$\begin{pmatrix} 70.000000000 & 50.000000000 & 4.000000000 \\ 23.000000000 & 50.000000000 & 8.000000000 \\ 43.000000000 & 78.000000000 & 59.000000000 \end{pmatrix}^{(-1)} =$$
$$\equiv$$
$$\begin{pmatrix} 0.0210030159 & -0.0238202734 & 0.00180593430 \\ -0.00914705723 & 0.0357394398 & -0.00422588626 \\ -0.00321456305 & -0.0298882127 & 0.0212197280 \end{pmatrix}$$

Рис 2. Тест1, результаты с сайта

```
dasha@dasha-K501UQ:~/masec/evm/pract4$ ./blas
n = 100
m = 1000
Time taken: 0.078979 sec.
```

Рис 3. Время вычисления матрицы 100 x 100