БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет прикладной математики и информатики

Лабораторная работа № 3

Выполнил: Ульяницкий Владимир Александрович
2 курс, 3 группа
Преподаватель: Полещук Максим Александрович

Минск, 2018

Задание 1.

1. Постановка задачи.

Пусть дана система линейных алгебраических уравнений вида

$$a_{11}x_1+a_{12}x_2+ \ \dots \ +a_{1n}x_n=f_1,$$
$$a_{21}x_2+a_{22}x_2+ \ \dots \ +a_{2n}x_n=f_2,$$
$$\dots\dots\dots\dots\dots\dots\dots$$
$$a_{n1}x_1+a_{n2}x_2+ \dots \ +a_{nn}x_n=f_n.$$

Разработать программу численного решения системы линейных уравнений методом Якоби. Задать матрицу системы:

$$\begin{pmatrix} 2m & -1 & 0 & \cdots & 0 & 0 & 0 \\ -m & 4m & -1 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & -m & 4m & -1 \\ 0 & 0 & 0 & \cdots & 0 & -m & 2m \end{pmatrix},$$

правую часть f умножением матрицы A на вектор $x = (m, m+1, ..., m+n–1)$: $f=Ax$. Для вычислений выбрать параметры: m — номер в списке студенческой группы; n — одно из чисел в пределах от 100 до 120.

Выйти из итерационного процесса, если выполнено одно из условий:

1. $\frac{\|B\|_\infty}{1-\|B\|_\infty}\left\|x^{(k)}-x^{(k-1)}\right\|_\infty \le \varepsilon,$ где матрица B — итерационная матрица Якоби; $x^{(0)}=b$, где вектор b — свободный член итерационного процесса $x^{(k+1)}=Bx^{(k)}+b$; параметр ε — верхняя граница абсолютной погрешности решения в нормах;
2. Номер итерации $k > k_{\max}$, где параметр $k_{\max}$ — ограничение числа итераций.

Задать параметры итерационного процесса $\varepsilon = 0,001, k_{\max} = 1000$.

Вычислить и представить в отчёте:

1. Вектор приближённого решения $\hat{x}$, полученный на последней итерации метода, и номер такой итерации;
2. Относительная погрешность решения в нормах — величина $\frac{\|x-\hat{x}\|_\infty}{\|\hat{x}\|_\infty}$, в которой вектор x — точное решение;
3. Среднее уменьшение ошибки за k итераций — значение $-\ln\frac{\left\|\varepsilon^{(k)}\right\|_2}{\left\|\varepsilon^{(0)}\right\|_2}$, где $\varepsilon^{(k)}$ — вектор ошибки $\hat{x}-x$ на k-й итерации метода, вектор x — точное решение;

4. Значение нормы $\|B\|_\infty$;
5. Теоретическая оценка числа итераций для достижения точности ε — значение $\left\lceil \frac{\lg\varepsilon - \lg\|b\|_\infty + \lg(1-\|B\|_\infty)}{\lg\|B\|_\infty} - 1 \right\rceil$. Сравнить эту величину с числом потребовавшихся итераций метода.

2. Входные данные
m = 23, n = 111

3. Ход работы

1.Создание матрицы, соответствующей условию поставленной задачи.

2.Умножение матрицы на заданный столбец *X*. Получение столбца свободных членов *f*.

3.Создание матрицы *B* — итерационной матрицы Якоби, построение вектора b — свободного члена итерационного процесса $x^{(k+1)} = Bx^{(k)} + b$.

$$B = \begin{bmatrix} 0 & -\frac{a_{12}}{a_{11}} & \cdots & -\frac{a_{1n}}{a_{11}} \\ -\frac{a_{21}}{a_{22}} & 0 & \cdots & -\frac{a_{2n}}{a_{22}} \\ \vdots & \vdots & \ddots & \vdots \\ -\frac{a_{n1}}{a_{nn}} & -\frac{a_{n2}}{a_{nn}} & \cdots & 0 \end{bmatrix}, b = \begin{bmatrix} \frac{f_1}{a_{11}} \\ \vdots \\ \frac{f_n}{a_{nn}} \end{bmatrix}.$$

Каждое следующее приближение в методе Якоби рассчитывается по формуле:

$$x_i^{k+1} = \frac{1}{a_{ii}}\left(f_i - \sum_{j=1, j\neq i}^{n} a_{ij}x_j^k\right), i = 1, 2, \ldots, n, k = 0, 1, 2, \ldots$$

4.Задаем условия выхода из итерационного процесса.
5.Вычисляем вектор приближенного решения

6.Вычисляем относительную погрешность решения в нормах по формуле: $\frac{\|x - \hat{x}\|_\infty}{\|\hat{x}\|_\infty}$

7.Вычисляем среднее уменьшение ошибки за k итераций по формуле: $-\ln\frac{\|\varepsilon^{(k)}\|_2}{\|\varepsilon^{(0)}\|_2}$,

8.Находим значение нормы $\|B\|_\infty$

9.Вычисляем теоретическую оценку числа итераций для достижения точности по формуле $\left\lceil \frac{\lg\varepsilon - \lg\|b\|_\infty + \lg(1-\|B\|_\infty)}{\lg\|B\|_\infty} - 1 \right\rceil$ и сравниваем эту величину с числом потребовавшихся итераций метода.

4. Вывод программы

b = (2495.087891, 1995.587891, 2077.631836, 2159.675537, 2241.718506, 2323.762207, 2405.804199, 2487.850342, 2569.891357, 2651.937988, 2733.979004, 2816.021484, 2898.061768, 2980.109863, 3062.149414, 3144.197998, 3226.237305, 3308.2854, 3390.324219, 3472.373535, 3554.412109, 3636.461914, 3718.5, 3800.537598, 3882.587891, 3964.625488, 4046.675781, 4128.713379, 4210.763184, 4292.800293, 4374.851074, 4456.890137, 4538.936523, 4620.976074, 4703.024414, 4785.064453, 4867.108398, 4949.148926, 5031.200684, 5113.241699, 5195.282715, 5277.324219, 5359.375977, 5441.418945, 5523.45752, 5605.5, 5687.542969, 5769.580078, 5851.623535, 5933.675781, 6015.719727, 6097.754883, 6179.798828, 6261.851563, 6343.895996, 6425.929688, 6507.974609, 6590.025391, 6672.070801, 6754.103027, 6836.148438, 6918.20166, 7000.24707, 7082.27832, 7164.324219, 7246.377441, 7328.423828, 7410.453613, 7492.5, 7574.546387, 7656.575195, 7738.62207, 7820.675781, 7902.723145, 7984.750977, 8066.797852, 8148.851563, 8230.899414, 8312.926758, 8394.973633, 8477.026367, 8559.072266, 8641.100586, 8723.148438, 8805.202148, 8887.25, 8969.280273, 9051.329102, 9133.373047, 9215.420898, 9297.451172, 9379.5, 9461.548828, 9543.580078, 9625.629883, 9707.666992, 9789.71582, 9871.749023, 9953.797852, 10035.85156, 10117.90137, 10199.93457, 10281.9834, 10364.0166, 10446.06543, 10528.09863, 10610.14844, 10692.20215, 10774.25195, 10856.28711, 7437)

x = (23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133)

m = 23; n = 111

Метод Якоби:

Относительная погрешность ||x - x*||/||x|| = 1.147277402e-07

норма B: 0.5

Приближённый вектор значений:

(23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 34.99999619, 35.99999619, 36.99999619, 37.99999619, 38.99999619, 39.99999619, 41, 42, 43, 44, 44.99999619, 45.99999619, 47, 48, 49, 50, 51, 51.99999619, 52.99999619, 53.99999619, 54.99999619, 55.99999619, 56.99999619, 57.99999619, 58.99999619, 59.99999619, 60.99999619, 61.99999619, 62.99999619, 63.99999619, 64.99999237, 65.99999237, 66.99999237, 67.99999237, 68.99999237, 69.99999237, 70.99999237, 71.99999237, 72.99999237, 73.99999237, 74.99999237, 75.99999237, 76.99999237, 77.99999237, 78.99999237, 79.99999237, 81, 82, 83, 84, 85, 86, 87, 88, 89, 89.99999237, 90.99999237, 91.99999237, 92.99999237, 93.99999237, 94.99999237, 95.99999237, 96.99999237, 97.99999237, 98.99999237, 99.99999237, 100.9999924, 101.9999924, 102.9999924, 103.9999924, 104.9999924, 105.9999924, 106.9999924, 107.9999924, 108.9999924, 109.9999924, 110.9999924, 111.9999924, 112.9999924, 113.9999924, 114.9999924, 115.9999924, 116.9999924, 117.9999924, 118.9999924, 119.9999924, 120.9999924, 121.9999924, 122.9999924, 123.9999924, 124.9999924, 125.9999924, 126.9999924, 127.9999924, 128.9999847, 129.9999847, 130.9999847, 131.9999847, 132.9999847)

Номер итерации:12

Теоретическая оценка числа итераций: 26

Теоретическая оценка числа итераций для достижения точности больше числа потребовавшихся итераций метода

Среднее уменьшение ошибки за k итераций: 16.38036919

5. Листинг

```
// Jacobi.cpp: определяет точку входа для консольного приложения.
//

#include "stdafx.h"
#include <iostream>
#include <iomanip>

using namespace std;

// функция для вывода матрицы

void print_matrixA(float** A, int n) {
	for (int i = 0; i < n; i++) {
		for (int j = 0; j < n; j++) {
			printf("%4.0f", A[i][j]);
		}
		cout << endl;
	}
	cout << endl;
}
void print_matrixB(float** A, int n) {
	for (int i = 0; i < n; i++) {
		for (int j = 0; j < n; j++) {
			printf("%8.4f", A[i][j]);
		}
		cout << endl;
	}
	cout << endl;
}
void  matrixNormB(float** B, int n) {
	float maxSumInRow = -1;
	for (int i = 0; i < n; i++) {
		float sumIRow = 0;
		for (int j = 0; j < n; j++) {
			sumIRow += abs(B[i][j]);
		}
		if (sumIRow > maxSumInRow) {
			maxSumInRow = sumIRow;
		}
	}
	cout << "норма B: " << maxSumInRow << endl;
}

// функция для вывода расширенной матрицы

void printx_matrix(float** A, int n) {
	for (int i = 0; i < n; i++) {
		for (int j = 0; j < n + 1; j++) {
			cout << "\t" << setprecision(3) << A[i][j];
		}
		cout << endl;
	}
	cout << endl;
}

//функция для вывода вектора

void print_vector(float* x, int n) {
	cout << "(";
	for (int i = 0; i < n - 1; i++)
```

```
			cout << setprecision(10) << x[i] << ", ";
		cout << x[n - 1] << ")" << endl << endl;
}

//функция для приравнивания векторов
void vect_equal(float* X1, float* X2, int n) {
		for (int i = 0; i < n; i++) {
			X1[i] = X2[i];
		}
}
//функция для проверки условий о выходе из итерационного процесса
bool task(float* currx, float* prevx, int n, float eps, int kmax, int kteq) {
		if (kteq == 0)
			return false;
		if (kteq == kmax) {
			//исключение
			throw kteq;
		}
		else {
			float maxsub = 0;
			for (int i = 0; i < n; i++) {
				if (abs(currx[i] - prevx[i]) > maxsub) {
					maxsub = abs(currx[i] - prevx[i]);
				}
			}
			return maxsub < eps ? true : false;
		}
}
int main()
{
		int m = 23; //номер в списке студенческой группы
		int n = 111;
		float eps = 0.0001;
		int kmax = 1000;
		setlocale(LC_ALL, ".1251");
		// Зададим матрицу A и вектор x
		float* x = new float[n];
		for (int i = 0; i < n; i++)
			x[i] = (float)m + i;

		float** A = new float*[n];
		for (int i = 0; i < n; i++) {
			A[i] = new float[n + 1];
		}
		for (int i = 0;i < n;i++) {
			for (int j = 0;j< n;j++) {
				A[i][j] = 0;
			}
		}
		for (int i = 1; i < n; i++) {
			A[i][i - 1] = -m;
			A[0][0] = 2 * m;

			A[0][1] = -1;
			if (i < n - 1) {
				A[i][i + 1] = -1;
			}

			A[i][i] = 4 * m;
			A[n - 1][n - 1] = A[0][0];
		}
```

```
	// Выводим исходную матрицу A
	cout << "Матрица A :" << endl;
	print_matrixA(A, n);

	// Создаём вектор f по формуле Ax = f
	float* f = new float[n];
	for (int i = 0; i < n; i++) {
		f[i] = 0;
		for (int j = 0; j < n; j++)
			f[i] += A[i][j] * x[j];
	}
	cout << "f = ";
	print_vector(f, n);

	for (int i = 0; i < n; i++) {
		A[i][n] = f[i];
	}

	//матрица B
	float** B = new float*[n];
	for (int i = 0; i < n; i++) {
		B[i] = new float[n + 1];
	}
	for (int i = 0; i < n; i++) {
		for (int j = 0; j < n; j++) {
			if (i != j) {
				B[i][j] = -A[i][j] / A[i][i];
			}
			else {
				B[i][j] = 0;
			}
		}
	}
	cout << endl;
	cout << "Матрица B :" << endl;
	print_matrixB(B, n);
	//b
	float* b = new float[n];
	for (int i = 0; i < n; i++) {
		b[i] = 0;
		for (int j = 0; j < n; j++)
			b[i] += f[i] / A[i][i];
	}
	cout << "b = ";
	print_vector(b, n);
	cout << "x = ";
	print_vector(x, n);
	cout << "m = " << m << "; n = " << n << endl;

	cout << "Метод Якоби: " << endl;
	float* currx = new float[n];
	float* prevx = new float[n];
	for (int i = 0; i < n; i++) {
		currx[i] = 0;
		prevx[i] = 0;
	}
	int currk = 0;
	try {
		while (!task(currx, prevx, n, eps, kmax, currk)) {
			//если условие о выходе из итерационного процесса не выполняется,
то выполняем итерацию
			vect_equal(prevx, currx, n); // приравниваем векторы
```

```
                for (int i = 0; i < n; i++) {
                        float sum1 = 0;
                        float sum2 = 0;
                        for (int j = 0; j < n; j++) {
                                if (j < i) {
                                        sum1 += A[i][j] * prevx[j];
                                }
                                else if (j > i) {
                                        sum2 += A[i][j] * prevx[j];
                                }
                        }
                        currx[i] = 1 / A[i][i] * (A[i][n] - sum1 - sum2);
                }
                currk++;
        }
    }
    catch (int num) {
            cout << endl;
            cout << "Выход из итерационного процесса: ( текущее k = " << num << " при
max k = 1000)" << endl << endl;
    }

    // Найдем относительную погрешность через кубическую норму по формуле ||X -
X*||/||X||
    float err = abs(x[0] - currx[0]);
    for (int i = 1; i < n; i++)
            if (abs(x[i] - currx[i]) > err)
                    err = abs(x[i] - currx[i]);
    float temp = x[0];
    for (int i = 1; i < n; i++)
            if (abs(x[i]) > temp)
                    temp = abs(x[i]);
    cout << "Относительная погрешность ||x - x*||/||x|| = ";
    cout << err / temp << endl << endl;

    matrixNormB(B, n);

    int theoreticalNumberOfIterations = 0;
    float vectNorm = b[0];
    for (int i = 1; i < n; i++)
            if (abs(b[i]) > vectNorm)
                    vectNorm = abs(b[i]);

    float maxSumInRow = -1;
    for (int i = 0; i < n; i++) {
            float sumIRow = 0;
            for (int j = 0; j < n; j++) {
                    sumIRow += abs(B[i][j]);
            }
            if (sumIRow > maxSumInRow) {
                    maxSumInRow = sumIRow;
            }
    }
    cout << "Приближённый вектор значений:" << endl;
    print_vector(currx, n);
    cout << "Номер итерации:" << currk << endl << endl;
    theoreticalNumberOfIterations = ((log(eps) - log(vectNorm) + log(1 -
maxSumInRow)) / log(maxSumInRow) - 1);
    cout << "Теоретическая оценка числа итераций: " << theoreticalNumberOfIterations
<< endl;
    if (theoreticalNumberOfIterations > currk)
    {
```

```
			cout << "Теоретическая оценка числа итераций для достижения точности
больше числа потребовавшихся итераций метода " << endl;
		}
		else if (theoreticalNumberOfIterations < currk)
		{
			cout << "Теоретическая оценка числа итераций для достижения точности
меньше числа потребовавшихся итераций метода " << endl;
		}
		else
		{
			cout << "Теоретическая оценка числа итераций для достижения точности
равна числу потребовавшихся итераций метода " << endl;
		}
		float CountNormaV2 = 0.0f;
		for (int j = 0; j < n; j++)
			CountNormaV2 += (abs(x[j] - currx[j])*abs(x[j] - currx[j]));
		CountNormaV2 = powf(CountNormaV2, 0.5f);

		float CountNormaV3 = 0.0f;
		for (int j = 0; j < n; j++)
			CountNormaV3 += (abs(x[j])*abs(x[j]));
		CountNormaV3 = powf(CountNormaV3, 0.5f);

		float sk = -log(CountNormaV2 / CountNormaV3);
		cout << "Среднее уменьшение ошибки за k итераций: " << sk << endl;

		system("pause");
		return 0;
}
```

Задание 2.

1. Постановка задачи

Разработать программу численного решения системы линейных уравнений методом Гаусса-Зейделя. Параметры итерационного процесса $\varepsilon, k_{\max}$ использовать из задания 1.

Вычислить и представить в отчете величины, указанные в пунктах 1, 2, 3 задания 1.

2. Входные данные

m = 23, n = 111

3. Ход работы

1.Создание матрицы, соответствующей условию поставленной задачи.

2.Умножение матрицы на заданный столбец *X*. Получение столбца свободных членов *f*.

3.Создание матрицы *B* — итерационной матрицы Якоби, построение вектора b — свободного члена итерационного процесса $x^{(k+1)} = Bx^{(k)} + b$.

4.Задание условия выхода из итерационного процесса.

5.Вычисление вектора приближенного решения

6.Вычисление относительной погрешности решения в нормах по формуле: $\frac{\|x - \hat{x}\|_\infty}{\|\hat{x}\|_\infty}$

7.Вычисление среднего уменьшения ошибки за k итераций по формуле: $-\ln\frac{\left\|\varepsilon^{(k)}\right\|_2}{\left\|\varepsilon^{(0)}\right\|_2}$,

4. Вывод программы

```
b = (22.47826004, 17.97826004, 12.43478298, 19.45652199, 13.39130402, 20.93478203,
14.347826, 22.41304398, 15.30434799, 23.89130402, 16.26086998, 25.36956596, 17.21739197,
26.847826, 18.17391396, 28.32608604, 19.13043404, 29.80434799, 20.08695602, 31.28260803,
21.04347801, 32.76086807, 22, 34.23913193, 22.95652199, 35.71739197, 23.91304398,
37.19565201, 24.86956596, 38.67391205, 25.82608604, 40.15217209, 26.78260803, 41.63043594,
27.73913002, 43.10869598, 28.69565201, 44.58695602, 29.652174, 46.06521606, 30.60869598,
47.54347992, 31.56521797, 49.02173996, 32.52173996, 50.5, 33.47826004, 51.97826004,
34.43478394, 53.45652008, 35.39130402, 54.93478394, 36.34782791, 56.41304398, 37.30434799,
57.89130402, 38.26086807, 59.36956406, 39.21739197, 60.84782791, 40.17391205, 62.32608795,
41.13043594, 63.80434799, 42.08695602, 65.28260803, 43.04347992, 66.76087189, 44,
68.23912811, 44.95652008, 69.71739197, 45.91304398, 71.19565582, 46.86956406, 72.67391205,
47.82608795, 74.1521759, 48.78260803, 75.63043213, 49.73913193, 77.10869598, 50.69565201,
78.58695984, 51.65217209, 80.06521606, 52.60869598, 81.54347992, 53.56521606, 83.02173615,
54.52173996, 84.5, 55.47826004, 85.97826385, 56.43478394, 87.45652008, 57.39130402,
88.93478394, 58.34782791, 90.41304016, 59.30434799, 91.89130402, 60.26086807, 93.36956787,
61.21739197, 94.8478241, 62.17391205, 96.32608795, 63.13043594, 97.80434418, 67)



x = (23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43,
44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66,
67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89,
90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109,
110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127,
128, 129, 130, 131, 132, 133)



m = 23; n = 111

Метод Гаусса-Зейделя:

Приближённый вектор значений:

(23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44,
45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,
68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90,
91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110,
111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128,
129, 130, 131, 132, 133)



Номер итерации:6



Относительная погрешность ||x - x*||/||x|| = 0



Среднее уменьшение ошибки за k итераций: nan
```

Вывод: за счёт использования уточнённых координат метод Гаусса-Зейделя сошёлся быстрее.

5.Листинг

```
// Gauss-Zeydel.cpp: определяет точку входа для консольного приложения.
//
```

```cpp
#include "stdafx.h"
#include <iostream>
#include <iomanip>

using namespace std;

// функция для вывода матрицы

void print_matrixA(float** A, int n) {
	for (int i = 0; i < n; i++) {
		for (int j = 0; j < n; j++) {
			printf("%4.0f", A[i][j]);
		}
		cout << endl;
	}
	cout << endl;
}
void print_matrixB(float** A, int n) {
	for (int i = 0; i < n; i++) {
		for (int j = 0; j < n; j++) {
			printf("%8.4f", A[i][j]);
		}
		cout << endl;
	}
	cout << endl;
}

// функция для вывода расширенной матрицы

void printx_matrix(float** A, int n) {
	for (int i = 0; i < n; i++) {
		for (int j = 0; j < n + 1; j++) {
			cout << "\t" << setprecision(3) << A[i][j];
		}
		cout << endl;
	}
	cout << endl;
}

//функция для вывода вектора

void print_vector(float* x, int n) {
	cout << "(";
	for (int i = 0; i < n - 1; i++)
		cout << setprecision(10) << x[i] << ", ";
	cout << x[n - 1] << ")" << endl << endl;
}

//функция для приравнивания векторов
void vect_equal(float* X1, float* X2, int n) {
	for (int i = 0; i < n; i++) {
		X1[i] = X2[i];
	}
}
//функция для проверки условий о выходе из итерационного процесса
bool task(float* currx, float* prevx, int n, float eps, int kmax, int kteq) {
	if (kteq == 0)
		return false;
	if (kteq == kmax) {
		//исключение
		throw kteq;
```

```
        }
        else {
                float maxsub = 0;
                for (int i = 0; i < n; i++) {
                        if (abs(currx[i] - prevx[i]) > maxsub) {
                                maxsub = abs(currx[i] - prevx[i]);
                        }
                }
                return maxsub < eps ? true : false;
        }
}
int main()
{
        int m = 23; //номер в списке студенческой группы
        int n = 111;
        float eps = 0.0001;
        int kmax = 1000;
        setlocale(LC_ALL, ".1251");
        // Зададим матрицу A и вектор x
        float* x = new float[n];
        for (int i = 0; i < n; i++)
                x[i] = (float)m + i;

        float** A = new float*[n];
        for (int i = 0; i < n; i++) {
                A[i] = new float[n + 1];
        }
        for (int i = 0;i < n;i++) {
                for (int j = 0;j< n;j++) {
                        A[i][j] = 0;
                }
        }
        for (int i = 0; i < n - 1; i++) {
                A[i + 1][i] = -m;
                A[i][i + 1] = -1;
                A[i][i] = (2 + 2*(i % 2))*m;
        }
        A[n-1][n-1] = (4 - 2*(n % 2))*m;

        // Выводим исходную матрицу A
        cout << "Матрица A:" << endl;
        print_matrixA(A, n);

        // Создаём вектор f по формуле Ax = f
        float* f = new float[n];
        for (int i = 0; i < n; i++) {
                f[i] = 0;
                for (int j = 0; j < n; j++)
                        f[i] += A[i][j] * x[j];
        }
        cout << "f = ";
        print_vector(f, n);

        for (int i = 0; i < n; i++) {
                A[i][n] = f[i];
        }

        //матрица B
        float** B = new float*[n];
        for (int i = 0; i < n; i++) {
                B[i] = new float[n + 1];
        }
```

```
	for (int i = 0; i < n; i++) {
		for (int j = 0; j < n; j++) {
			if (i != j) {
				B[i][j] = -A[i][j] / A[i][i];
			}
			else {
				B[i][j] = 0;
			}
		}
	}
	cout << endl;
	cout << "Матрица B:" << endl;
	print_matrixB(B, n);
	//b
	float* b = new float[n];
	for (int i = 0; i < n; i++) {
		b[i] = f[i] / A[i][i];
	}

	cout << "b = ";
	print_vector(b, n);
	cout << "x = ";
	print_vector(x, n);
	cout << "m = " << m << "; n = " << n << endl;

	cout << "Метод Гаусса-Зейделя: " << endl;
	float* currx = new float[n];
	float* prevx = new float[n];
	float* TempX = new float[n];
	for (int i = 0; i < n; i++) {
		currx[i] = f[i];
		prevx[i] = f[i];
	}
	int currk = 0;
	try {
		while (!task(currx, prevx, n, eps, kmax, currk)) {
			//если условие о выходе из итерационного процесса не выполняется,
то выполняем итерацию
			vect_equal(prevx, currx, n); // приравниваем векторы
			for (int i = 0; i < n; i++) {
				float sum1 = 0;
				float sum2 = 0;
				for (int j = 0; j < n; j++) {
					if (j < i) {
						sum1 += A[i][j] * currx[j];
					}
					else if (j > i) {
						sum2 += A[i][j] * prevx[j];
					}
				}
				currx[i] = 1 / A[i][i] * (f[i] - sum1 - sum2);
			}
			currk++;

		}
	}
	catch (int num) {
		cout << endl;
		cout << "Выход из итерационного процесса: (текущее k = " << num << " при
max k = 1000)" << endl << endl;
	}
```

```
	cout << "Приближённый вектор значений:" << endl;
	print_vector(currx, n);
	cout << "Номер итерации:" << currk << endl << endl;

	float err = abs(x[0] - currx[0]);
	for (int i = 1; i < n; i++)
		if (abs(x[i] - currx[i]) > err)
			err = abs(x[i] - currx[i]);
	float temp = x[0];
	for (int i = 1; i < n; i++)
		if (abs(x[i]) > temp)
			temp = abs(x[i]);
	cout << "Относительная погрешность ||x - x*||/||x|| = ";
	cout << err / temp << endl << endl;

	float CountNormaV2 = 0.0f;
	for (int j = 0; j < n; j++)
		CountNormaV2 += (abs(currx[j] - x[j])*abs(currx[j] - x[j]));
	CountNormaV2 = powf(CountNormaV2, 0.5f);

	float CountNormaV3 = 0.0f;
	for (int j = 0; j < n; j++)
		CountNormaV3 += (abs(x[j])*abs(x[j]));
	CountNormaV3 = powf(CountNormaV2, 0.5f);

	float sk = -log(CountNormaV2 / CountNormaV3);
	cout << "Среднее уменьшение ошибки за k итераций: " << sk << endl;

	system("pause");
	return 0;
}
```

Задание 3.

1. Постановка задачи

Разработать программу численного решения полной проблемы собственных значений матрицы A методом Данилевского. Задать матрицу A:

$$\frac{1}{n(n-1)}\begin{pmatrix} 0 & n-1 & n-2 & \cdots & 3 & 2 & 1 \\ n-1 & 0 & n-1 & \cdots & 4 & 3 & 2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 2 & 3 & 4 & \cdots & n-1 & 0 & n-1 \\ 1 & 2 & 3 & \cdots & n-2 & n-1 & 0 \end{pmatrix} + diag\{10m, 10(m+1), \ldots, 10(m+n-1)\}$$

Для вычислений выбрать параметры: m — номер в списке студенческой группы; n — одно из чисел в пределах от 15 до 20.

Вычислить и представить в отчёте коэффициенты характеристического многочлена матрицы A_1 после последней итерации.

2. Входные данные

m = 23, n = 17

3. Ход работы

1.Создание матрицы, соответствующей условию поставленной задачи.

2. Получить 1 на месте элемента $a_{n\,n-1}$, все остальные элементы n-й строки обратить в 0.

Для этого, по аналогии с методом Гаусса (вариант, в котором на месте ведущего элемента получается 1), следует разделить $(n-1)$-й столбец матрицы A на $a_{n\,n-1}$, а потом элементы $(n-1)$-го столбца умножить на a_{ni} и вычесть из соответствующих элементов i-го столбца. Это действие эквивалентно умножению матрицы A справа на матрицу

$$M_{n-1}=\begin{bmatrix} 1 & \cdots & 0 & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & 1 & 0 & 0 \\ -\dfrac{a_{n1}}{a_{n\,n-1}} & \cdots & -\dfrac{a_{n\,n-2}}{a_{n\,n-1}} & \dfrac{1}{a_{n\,n-1}} & -\dfrac{a_{nn}}{a_{n\,n-1}} \\ 0 & \cdots & 0 & 0 & 1 \end{bmatrix}.$$

3. Умножить получившуюся в пункте 1 матрицу слева на матрицу

$$M_{n-1}^{-1}=\begin{bmatrix} 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{n\,n-1} & a_{nn} \\ 0 & 0 & \cdots & 0 & 1 \end{bmatrix}.$$

Это делается для того, чтобы получившаяся матрица была подобна матрице A.

Матрица M_{n-1}^{-1} существует, так как $\det M_{n-1}=\dfrac{1}{a_{n\,n-1}}\neq 0$ и предполагается $a_{n\,n-1}\neq 0$. В результате получим матрицу

$$A_1=M_{n-1}^{-1}AM_{n-1}=\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1\,n-1}^{(1)} & a_{1n}^{(1)} \\ a_{21}^{(1)} & a_{22}^{(1)} & \cdots & a_{2\,n-1}^{(1)} & a_{2n}^{(1)} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{n-1\,1}^{(1)} & a_{n-1\,2}^{(1)} & \cdots & a_{n-1\,n-1}^{(1)} & a_{n-1\,n}^{(1)} \\ 0 & 0 & \cdots & 1 & 0 \end{bmatrix}.$$

4. Повторить действия пункта 2 и пункта 3, обращая в 1 элементы на месте $a_{n-1\,n-2}$, …, a_{21}.

Сначала получим

$$A_2=M_{n-2}^{-1}A_1M_{n-2}=M_{n-2}^{-1}\,M_{n-1}^{-1}AM_{n-1}M_{n-2}=\begin{bmatrix} a_{11}^{(2)} & \cdots & a_{1\,n-2}^{(2)} & a_{1\,n-1}^{(2)} & a_{1n}^{(2)} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{n-2\,1}^{(2)} & \cdots & a_{n-2\,n-2}^{(2)} & a_{n-2\,n-1}^{(2)} & a_{n-2\,n}^{(2)} \\ 0 & \cdots & 1 & 0 & 0 \\ 0 & \cdots & \cdots & 1 & 0 \end{bmatrix},$$

где

$$M_{n-2}=\begin{bmatrix} 1 & \cdots & 0 & 0 & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ -\dfrac{a^{(1)}_{n-1\,1}}{a^{(1)}_{n-1\,n-2}} & \cdots & -\dfrac{a^{(1)}_{n-1\,n-3}}{a^{(1)}_{n-1\,n-2}} & \dfrac{1}{a^{(1)}_{n-1\,n-2}} & -\dfrac{a^{(1)}_{n-1\,n-1}}{a^{(1)}_{n-1\,n-2}} & -\dfrac{a^{(1)}_{n-1\,n}}{a^{(1)}_{n-1\,n-2}} \\ 0 & \cdots & 0 & 0 & 1 & 0 \\ 0 & \cdots & 0 & 0 & 0 & 1 \end{bmatrix},$$

$$M^{-1}_{n-2}=\begin{bmatrix} 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a^{(1)}_{n-1\,1} & a^{(1)}_{n-1\,2} & \cdots & a^{(1)}_{n-1\,n-1} & a^{(1)}_{n-1\,n} \\ 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & \cdots & 0 & 1 \end{bmatrix}.$$

После всех преобразований получим

$$A_{n-1}=M^{-1}_1\,M^{-1}_2\ldots M^{-1}_{n-2}\,M^{-1}_{n-1}\,A\,M_{n-1}M_{n-2}\ldots M_2M_1=S^{-1}AS=$$

$$\begin{bmatrix} a^{(n-1)}_{11} & a^{(n-1)}_{12} & \cdots & a^{(n-1)}_{1\,n-1} & a^{(n-1)}_{1\,n} \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & 1 & 0 \end{bmatrix}=\begin{bmatrix} p_1 & p_2 & \cdots & p_{n-1} & p_n \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & 1 & 0 \end{bmatrix}=\Phi.$$

По первой строке матрицы Ф составляется собственный многочлен матрицы A:

$$P(\lambda)=\lambda^n-p_1\lambda^{n-1}-p_2\lambda^{n-2}-\ldots-p_n.$$

Все преобразования регулярного случая возможны, если $a_{n\,n-1}\neq 0$, $a^{(1)}_{n-1\,n-2}\neq 0$, … , $a^{(n-2)}_{21}\neq 0$.

4. Вывод программы

```
Матрица A :

46 0.0588235 0.0551471 0.0514706 0.0477941 0.0441176 0.0404412 0.0367647 0.0330882
0.0294118 0.0257353 0.0220588 0.0183824 0.0147059 0.0110294 0.00735294 0.00367647

0.0588235 56 0.0588235 0.0551471 0.0514706 0.0477941 0.0441176 0.0404412 0.0367647
0.0330882 0.0294118 0.0257353 0.0220588 0.0183824 0.0147059 0.0110294 0.00735294

0.0551471 0.0588235 66 0.0588235 0.0551471 0.0514706 0.0477941 0.0441176 0.0404412
0.0367647 0.0330882 0.0294118 0.0257353 0.0220588 0.0183824 0.0147059 0.0110294

0.0514706 0.0551471 0.0588235 76 0.0588235 0.0551471 0.0514706 0.0477941 0.0441176
0.0404412 0.0367647 0.0330882 0.0294118 0.0257353 0.0220588 0.0183824 0.0147059

0.0477941 0.0514706 0.0551471 0.0588235 86 0.0588235 0.0551471 0.0514706 0.0477941
0.0441176 0.0404412 0.0367647 0.0330882 0.0294118 0.0257353 0.0220588 0.0183824
```

0.0441176 0.0477941 0.0514706 0.0551471 0.0588235 96 0.0588235 0.0551471 0.0514706
0.0477941 0.0441176 0.0404412 0.0367647 0.0330882 0.0294118 0.0257353 0.0220588

0.0404412 0.0441176 0.0477941 0.0514706 0.0551471 0.0588235 106 0.0588235 0.0551471
0.0514706 0.0477941 0.0441176 0.0404412 0.0367647 0.0330882 0.0294118 0.0257353

0.0367647 0.0404412 0.0441176 0.0477941 0.0514706 0.0551471 0.0588235 116 0.0588235
0.0551471 0.0514706 0.0477941 0.0441176 0.0404412 0.0367647 0.0330882 0.0294118

0.0330882 0.0367647 0.0404412 0.0441176 0.0477941 0.0514706 0.0551471 0.0588235 126
0.0588235 0.0551471 0.0514706 0.0477941 0.0441176 0.0404412 0.0367647 0.0330882

0.0294118 0.0330882 0.0367647 0.0404412 0.0441176 0.0477941 0.0514706 0.0551471 0.0588235
136 0.0588235 0.0551471 0.0514706 0.0477941 0.0441176 0.0404412 0.0367647

0.0257353 0.0294118 0.0330882 0.0367647 0.0404412 0.0441176 0.0477941 0.0514706 0.0551471
0.0588235 146 0.0588235 0.0551471 0.0514706 0.0477941 0.0441176 0.0404412

0.0220588 0.0257353 0.0294118 0.0330882 0.0367647 0.0404412 0.0441176 0.0477941 0.0514706
0.0551471 0.0588235 156 0.0588235 0.0551471 0.0514706 0.0477941 0.0441176

0.0183824 0.0220588 0.0257353 0.0294118 0.0330882 0.0367647 0.0404412 0.0441176 0.0477941
0.0514706 0.0551471 0.0588235 166 0.0588235 0.0551471 0.0514706 0.0477941

0.0147059 0.0183824 0.0220588 0.0257353 0.0294118 0.0330882 0.0367647 0.0404412 0.0441176
0.0477941 0.0514706 0.0551471 0.0588235 176 0.0588235 0.0551471 0.0514706

0.0110294 0.0147059 0.0183824 0.0220588 0.0257353 0.0294118 0.0330882 0.0367647 0.0404412
0.0441176 0.0477941 0.0514706 0.0551471 0.0588235 186 0.0588235 0.0551471

0.00735294 0.0110294 0.0147059 0.0183824 0.0220588 0.0257353 0.0294118 0.0330882 0.0367647
0.0404412 0.0441176 0.0477941 0.0514706 0.0551471 0.0588235 196 0.0588235

0.00367647 0.00735294 0.0110294 0.0147059 0.0183824 0.0220588 0.0257353 0.0294118 0.0330882
0.0367647 0.0404412 0.0441176 0.0477941 0.0514706 0.0551471 0.0588235 206

Метод Данилевского:

2142 -2.13874e+06 1.3217e+09 -5.6603e+11 1.7822e+14 -4.27067e+16 7.95295e+18 -1.16481e+21
1.34895e+23 -1.23478e+25 8.87963e+26 -4.95316e+28 2.09731e+30 -6.50715e+31 1.39334e+33 -
1.83753e+34 1.12323e+35

1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0

```
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
```

```
Коэффициенты характеристического многочлена матрицы:

1 2142 -2.13874e+06 1.3217e+09 -5.6603e+11 1.7822e+14 -4.27067e+16 7.95295e+18 -1.16481e+21
1.34895e+23 -1.23478e+25 8.87963e+26 -4.95316e+28 2.09731e+30 -6.50715e+31 1.39334e+33 -
1.83753e+34 1.12323e+35
```

5.Листинг

```
#include "stdafx.h"
#include <iostream>
#include <iomanip>
#include<vector>
using namespace std;

void showMatrix(vector<vector<float>> A, int n) {
	for (int i = 0; i < n; i++) {
		for (int j = 0; j < n; j++) {
			cout << A[i][j] << " ";
		}
		cout << endl;
	}
	cout << endl;
}

vector<vector<float>> copyMatrix(vector<vector<float>>  A, int n) {
	vector<vector<float>> copyA(n);
	for (int i = 0; i < n; i++) {
		for (int j = 0; j < n; j++) {
			copyA[i].push_back(A[i][j]);
		}
	}
	return copyA;
}
vector<vector<float>>  matrixMk(vector<vector<float>>  A, int n, int k) {
	float a = A[k][k - 1];
	vector<vector<float>> Mk(n);
	for (int i = 0; i < n; i++) {
		for (int j = 0; j < n; j++) {
			if (j == i) {
				if (j == k - 1) {
					Mk[i].push_back(1 / a);
				}
				else {
					Mk[i].push_back(1);
				}
			}
			else if (i == k - 1) {
				Mk[i].push_back(-A[i + 1][j] / a);
			}
			else {
```

```
                        Mk[i].push_back(0);
                    }
                }
            }
            return Mk;
}
vector<vector<float>>  matrixMk_1(vector<vector<float>>  A, int n, int k) {
            vector<vector<float>>  Mk_1(n);
            for (int i = 0; i < n; i++) {
                        for (int j = 0; j < n; j++) {
                                    if (j == i) {
                                                if (j == k - 1) {
                                                            Mk_1[i].push_back(A[i + 1][j]);
                                                }
                                                else {
                                                            Mk_1[i].push_back(1);
                                                }
                                    }
                                    else if (i == k - 1) {
                                                Mk_1[i].push_back(A[i + 1][j]);
                                    }
                                    else {
                                                Mk_1[i].push_back(0);
                                    }
                        }
            }
            return Mk_1;
}

int main()
{
            setlocale(LC_ALL, "Rus");
            int n = 17;
            int m = 23;
            float koeff = n*(n - 1);
            float d = 2 * m;
            vector<vector<float>>  A(n);
            for (int i = 0; i < n; i++) {
                        for (int j = 0; j < n; j++) {
                                    if (j < i) {
                                                A[i].push_back((n - i + j) / koeff);
                                    }
                                    else if (j > i) {
                                                A[i].push_back((n + i - j) / koeff);
                                    }
                                    else {
                                                A[i].push_back(d);
                                                d += 10;
                                    }
                        }
            }
            cout << "Матрица A :" << endl;
            showMatrix(A, n);

            float sum;

            cout << "Метод Данилевского: " << endl << endl;
            for (int k = n - 1; k >= 1; k--) {
                        vector<vector<float>>  Mk = matrixMk(A, n, k);
                        vector<vector<float>> Mk_1 = matrixMk_1(A, n, k);
                        for (int i = 0; i < n; i++) {
                                    sum = 0.0;
```

```
                for (int j = 0; j < n; j++) {
                        sum += Mk_1[k - 1][j] * A[j][i];
                }
                A[k - 1][i] = sum;
        }
        vector<vector<float>> copyA = copyMatrix(A, n);
        for (int p = 0; p < n; p++) {
                for (int i = 0; i < n; i++) {
                        sum = 0.0;
                        if (i == k - 1) {
                                sum += copyA[p][k - 1] * Mk[k - 1][k - 1];
                        }
                        else {
                                sum += copyA[p][i];
                                sum += copyA[p][k - 1] * Mk[k - 1][i];
                        }
                        if (p == k && i == k - 1) {
                                A[p][i] = 1;
                        }
                        else if (p == k && i != k - 1) {
                                A[p][i] = 0;
                        }
                        else {
                                A[p][i] = sum;
                        }
                }
        }
    }
    showMatrix(A, n);

    cout << endl << "Коэффициенты характеристического многочлена матрицы:" << endl;
    int sign = (n % 2) ? 1 : -1;
    cout << sign << " ";
    for (int i = 0; i < n; i++) {
        cout << sign*A[0][i] << " ";
    }
    cout << endl;
    system("pause");
    return 0;
}
```