Cuprins

Fișiere comume

utils.h

```cpp
#pragma once
#include <initializer_list>
#include <iostream>

constexpr size_t getSize(std::initializer_list<double> l) {
    size_t n = 0;
    auto it = l.begin();
    auto end = l.end();
    while (it++ != end) ++n;
    return n;
}

constexpr void getSize(std::initializer_list<
                           std::initializer_list<double>> list,
                       int& s1, int& s2) {
    constexpr int invalid = -1;
    s1 = 0;
    s2 = invalid;

    for (const auto& l: list) {
        int i = getSize(l);
        ++s1;
        if (s2 != invalid && i != s2) {
            //std::cerr << i << "!=" << s2 << "\n";
            throw std::logic_error("Fixed line size expected");
        }
        s2 = i;
    }
    //std::cerr << "size: " << s1 << ", " << s2 << "\n";
}
constexpr int MAX_SZ = 256;
inline size_t readSize(const char* name, int a = 1, int b = MAX_SZ) {
    int res;
    do {
        std::cout << name << ": ";
        std::cin >> res;
    } while (res < a || res >= b);
    return res;
}
```

```
40
41  inline void assert(bool cond, const char* msg) {
42      if (!cond) throw std::logic_error(msg);
43  }
44  template<typename T>
45  static T read(const char* name) {
46      T res;
47      std::cout << name << ": ";
48      std::cin >> res;
49      return res;
50  }
```

matrix.h

```
1   #pragma once
2   #include "utils.h"
3
4   #include <iostream>
5   #include <cmath>
6   #include <cstring>
7   #include <utility>
8
9   struct Mat {
10      int m, n;
11      double *data;
12      Mat() : m(0), n(0), data(nullptr) {}
13      Mat(int m, int n) : m(m), n(n) { data = new double[m*n]; }
14      //template<int m, int n>
15      Mat(std::initializer_list<std::initializer_list<double>> list) {
16          getSize(list, m, n);
17          data = new double[m*n];
18
19          auto it = begin();
20          for (const auto& l : list) {
21              for (const auto& v : l) *(it++) = v;
22          }
23      }
24      Mat(Mat&& rhs) noexcept :
25          m(rhs.m), n(rhs.n),
26          data(std::exchange(rhs.data, nullptr)) {
27          //rhs.n = rhs.m = 0;
28      }
29
30      Mat(const Mat& rhs) : Mat(rhs.m, rhs.n) {
31          std::memcpy(data, rhs.data, sizeof(double) *m*n);
32      }
33      Mat& operator=(const Mat& rhs) {
34          setSize(rhs.m, rhs.n);
35          std::memcpy(data, rhs.data, sizeof(double) *m*n);
36          return *this;
```

```
37          }
38          Mat& operator=(Mat&& rhs) noexcept {
39              this->~Mat();
40              m = rhs.m;
41              n = rhs.n;
42              data = std::exchange(rhs.data, nullptr);
43
44              return *this;
45          }
46
47          ~Mat() {
48              delete[] data;
49          }
50          double& at(int i, int j) {
51              assert(i < m && j < n, "out of range");
52              return data[i *n +j];
53          }
54          double at(int i, int j) const { return data[i * n + j]; }
55          double* begin() { return data; }
56          const double* begin() const { return data; }
57          double* end() { return data + m*n; }
58          const double* end() const { return data + m*n; }
59
60          void setSize(int m, int n) {
61              if (this->m == m && this->m == n) return;
62              this->~Mat();
63              new (this) Mat(m, n);
64          }
65
66          static Mat read() {
67              Mat res(readSize("m"), readSize("n"));
68              for (auto& i : res)
69                  std::cin >> i;
70              return res;
71          }
72          void print(const char* name) const {
73              std::cout << name << " = Mat " << m << "x" << n << "{\n";
74
75              for (int i = 0; i < m; ++i) {
76                  for (int j = 0; j < n; ++j) {
77                      std::cout << at(i, j) << " ";
78                  }
79                  std::cout << "\n";
80              }
81              std::cout << "}\n";
82          }
83
84          friend std::ostream& operator<<(std::ostream& s, Mat& m) {
85              s << "Mat " << m.m << "x" << m.n << "{\n";
86              for (int i = 0; i < m.m; ++i) {
```

```cpp
87              for (int j = 0; j < m.n; ++j) {
88                  s << m.at(i, j) << " ";
89              }
90              s << "\n";
91          }
92          return s<< "}\n";
93      }
94
95      double norm1() const { return normImpl<Type::Row>(n, m); }
96      double normInf() const { return normImpl<Type::Col>(m, n); }
97      double normF() const {
98          double res = 0;
99          for (int i = 0; i < m; ++i)
100             for (int j = 0; j < n; ++j)
101                 res += at(i, j) * at(i, j);
102
103         return std::sqrt(res);
104     }
105     bool isStrictlyRowDiagonallyDominant() const {
106         return isStrictlyDiagonallyDominantImpl<Type::Row>();
107     }
108
109     bool isStrictlyColDiagonallyDominant() const {
110         return isStrictlyDiagonallyDominantImpl<Type::Col>();
111     }
112
113 private:
114
115     enum class Type { Row, Col };
116     template<Type type>
117     bool isStrictlyDiagonallyDominantImpl() const {
118         assert(m == n, "Matrix must be square");
119         for (int i = 0; i < m; ++i) {
120             double val = std::abs(at(i, i));
121             double sum = -val;
122             for (int j = 0; j < m; ++j) {
123                 sum += std::abs(type == Type::Col? at(j, i): at(i, j));
124             }
125             if (sum >= val) {
126                 std::cout << "(" << sum << "," << val << ")";
127                 return false;
128             }
129         }
130         return true;
131     }
132     template<Type type>
133     double normImpl(int sz1, int sz2) const {
134         double max = -1;
135         for (int j = 0; j < sz1; ++j) {
136             double x = 0;
```

```
137            for (int i = 0; i < sz2; ++i) {
138                x += std::abs(type == Type::Col? at(j, i) : at(i, j));
139            }
140            if (x > max) max = x;
141        }
142        return max;
143    }
144 };
145
146 Mat& add(const Mat& a, const Mat& b, Mat& res) {
147     assert(a.m == b.m && a.n == b.n, "Sizes don't match, can't add");
148     res.setSize(a.m, a.n);
149     for (int i = 0; i < a.m; ++i)
150         for (int j = 0; j < a.n; ++j)
151             res.at(i, j) = a.at(i, j) + b.at(i, j);
152     return res;
153 }
154
155 Mat& mul(double a, const Mat& b, Mat& res) {
156     res.setSize(b.m, b.n);
157
158     for (int i = 0; i < res.m; ++i)
159         for (int j = 0; j < res.n; ++j)
160             res.at(i, j) = a * b.at(i, j);
161     return res;
162 }
163
164 Mat& neg(const Mat& a, Mat& res) {
165     return mul(-1, a, res);
166 }
167 Mat& sub(const Mat& a, const Mat& b, Mat& res) {
168     return add(a, neg(b, res), res);
169 }
170 Mat& mul(const Mat& a, const Mat& b, Mat& res) {
171     assert(a.n == b.m, "Sizes don't match, can't multiply");
172     res.setSize(a.m, b.n);
173
174     for (int i = 0; i < res.m; ++i)
175         for (int j = 0; j < res.n; ++j) {
176             res.at(i, j) = 0;
177             for (int k = 0; k < a.n; ++k)
178                 res.at(i, j) += a.at(i, k) * b.at(k, j);
179         }
180     return res;
181 }
182
183 Mat& trans(const Mat& a, Mat& res) {
184     assert(a.data != res.data, "Can't transpose inplace");
185     res.setSize(a.n, a.m);
186
```

```
187         for (int i = 0; i < res.m; ++i)
188             for (int j = 0; j < res.n; ++j)
189                 res.at(i, j) = a.at(j, i);
190         return res;
191 }
```

vector.h

```
1  #pragma once
2  #include "utils.h"
3
4  #include <iostream>
5  #include <utility>
6  #include <cmath>
7
8  struct Vec {
9      double *_begin, *_end;
10
11     constexpr double* begin() { return _begin; }
12     constexpr const double* begin() const { return _begin; }
13
14     constexpr double* end() { return _end; }
15     constexpr const double* end() const { return _end; }
16
17     constexpr Vec() : _begin(nullptr), _end(nullptr) {}
18     explicit Vec(size_t n) : _begin(new double[n]), _end(_begin+n) {}
19     Vec(std::initializer_list<double> list) : Vec(getSize(list)) {
20         auto it = _begin;
21         for (const auto& v : list) *(it++) = v;
22     }
23     Vec(const Vec& rhs) : Vec(rhs.size()) {
24         auto it = _begin;
25         for (const auto& v : rhs) *(it++) = v;
26     }
27     Vec(Vec&& rhs) noexcept
28         : _begin(std::exchange(rhs._begin, nullptr)),
29         _end(std::exchange(rhs._end, nullptr)) {}
30
31     Vec& operator=(const Vec& rhs) {
32         if (size() != rhs.size()) {
33             this->~Vec();
34             new (this) Vec(rhs.size());
35         }
36         auto it = _begin;
37         for (const auto& v : rhs) *(it++) = v;
38         return *this;
39     }
40     Vec& operator=(Vec&& rhs) noexcept {
41         this->~Vec();
42         _begin = std::exchange(rhs._begin, nullptr);
```

```cpp
        _end = std::exchange(rhs._end, nullptr);
        return *this;
    }

    ~Vec() { delete[] _begin; }

    constexpr size_t size() const { return _end - _begin; }

    constexpr double& operator[](size_t i) { return _begin[i]; }
    constexpr double operator[](size_t i) const { return _begin[i]; }

    void setSize(size_t n) {
        if (size() == n) return;
        *this = Vec(n);
    }

    friend std::ostream& operator<<(std::ostream& s, const Vec& v) {
        s << "(";
        double* it = v._begin;
        for (double* end = v._end - 1; it < end; ++it)
            s << *it << ", ";

        if (it < v._end) s << *it;

        return s << ")";
    }
    static Vec read() {
        Vec res(readSize("n"));
        for (auto& v: res) std::cin >> v;
        return res;
    }
    double norm() const;
};
void assertSizes(const Vec& a, const Vec& b) {
    assert(a.size() == b.size(), "Sizes don't match");
}

Vec& add(const Vec& a, const Vec& b, Vec& res) {
    assertSizes(a, b);
    res.setSize(a.size());
    auto aIt = a.begin();
    auto bIt = b.begin();
    for (auto& v : res) v = *(aIt++) + *(bIt++);
    return res;
}

Vec& mul(double a, const Vec& b, Vec& res) {
    res.setSize(b.size());
    auto it = b.begin();
    for (auto& v : res) v = a * (*(it++));
```

```cpp
93      return res;
94  }
95
96  Vec& neg(const Vec& b, Vec& res) { return mul(-1, b, res); }
97
98  Vec& sub(const Vec& a, const Vec& b, Vec& res) {
99      return add(a, neg(b, res), res);
100 }
101 double dot(const Vec& a, const Vec& b) {
102     double res = 0;
103     assertSizes(a, b);
104     auto bIt = b.begin();
105     for (auto& v : a) res += v * (*(bIt++));
106     return res;
107 }
108 double norm(const Vec& a) {
109     return std::sqrt(dot(a, a));
110 }
111 double Vec::norm() const {
112     return ::norm(*this);
113 }
```

Alocarea dinamică a memoriei.
Tipuri specifice.

Laborator 1

16. Scrieți funcții pentru implemetarea operațiilor specifice pe matrice de numere reale cu m linii și n coloane: suma, diferența și produsul al două matrice, produsul dintre o matrice și un scalar real, transpusa unei matrice, norme matriceale specifice[1], citirea de la tastatură a componentelor unei matrice, afișarea componentelor matricei. Pentru cazul particular al unei matrice patratice de ordin n, să se testeze dacă aceasta satisface criteriul de dominanță pe linii[2] sau pe coloane[3]. Se vor folosi tablouri bidimensionale alocate static.

```cpp
#include "utils.h"

#include <iostream>
#include <cmath>

struct Mat {
    double data[MAX_SZ][MAX_SZ] {};
    int m, n;

    Mat() : m(0), n(0) {}
    Mat(int m, int n) : m(m), n(n) {}

    static Mat read() {
        Mat res(readSize("m"), readSize("n"));

        for (int i = 0; i < res.m; ++i)
            for (int j = 0; j < res.n; ++j)
                std::cin >> res.data[i][j];
        return res;
    }
    void setSize(int m, int n) {
        this->m = m;
        this->n = n;
    }
```

---

[1]Dacă $A \in \mathcal{M}_{m \times n}(\mathbb{R})$, atunci $||A||_1 = \max\limits_{1 \leq j \leq n} \sum\limits_{i=1}^{m} |a_{ij}|$, $||A||_\infty = \max\limits_{1 \leq i \leq m} \sum\limits_{j=1}^{n} |a_{ij}|$, $||A||_F = \sqrt{\sum\limits_{i=1}^{m} \sum\limits_{j=1}^{n} a_{ij}^2}$.

[2]$A \in \mathcal{M}_n(\mathbb{R})$ este strict diagonal dominantă pe linii dacă $|a_{ii}| > \sum\limits_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}|$, pentru orice $i = 1, ..., n$.

[3]$A \in \mathcal{M}_n(\mathbb{R})$ este strict diagonal dominantă pe colonane dacă $|a_{jj}| > \sum\limits_{\substack{i=1 \\ i \neq j}}^{n} |a_{ij}|$, pentru orice $j = 1, ..., n$.

```cpp
        double& at(int i, int j) { return data[i][j]; }
        double at(int i, int j) const { return data[i][j]; }

        void print(const char* name) const {
            std::cout << name << " = Mat " << m << "x" << n << "{\n";
            for (int i = 0; i < m; ++i) {
                for (int j = 0; j < n; ++j)
                    std::cout << at(i, j) << " ";
                std::cout << "\n";
            }
            std::cout << "}\n";
        }
private:
        enum class Type { Row, Col };
        template<Type type>
        bool isStrictlyDiagonallyDominantImpl() const {
            assert(m == n, "Matrix must be square");
            for (int i = 0; i < m; ++i) {
                double val = std::abs(at(i, i));
                double sum = -val;
                for (int j = 0; j < m; ++j)
                    sum += std::abs(type == Type::Col ? at(j, i) : at(i, j));
                if (sum >= val) return false;
            }
            return true;
        }

        template<Type type>
        double normImpl(int sz1, int sz2) const {
            double max = -1;
            for (int j = 0; j < sz1; ++j) {
                double x = 0;
                for (int i = 0; i < sz2; ++i)
                    x += std::abs(type == Type::Col ? at(j, i) : at(i, j));
                if (x > max) max = x;
            }
            return max;
        }
public:
        double norm1() const { return normImpl<Type::Row>(n, m); }
        double normInf() const { return normImpl<Type::Col>(m, n); }
        double normF() const {
            double res = 0;
            for (int i = 0; i < m; ++i)
                for (int j = 0; j < n; ++j)
                    res += at(i, j) * at(i, j);

            return std::sqrt(res);
        }
        bool isStrictlyRowDiagonallyDominant() const {
```

```cpp
            return isStrictlyDiagonallyDominantImpl<Type::Row>();
        }
        bool isStrictlyColDiagonallyDominant() const {
            return isStrictlyDiagonallyDominantImpl<Type::Col>();
        }
};

Mat& add(const Mat& a, const Mat& b, Mat& res) {
    assert(a.m == b.m && a.n == b.n, "Sizes don't match, can't add");
    res.setSize(a.m, a.n);
    for (int i = 0; i < a.m; ++i)
        for (int j = 0; j < a.n; ++j)
            res.at(i, j) = a.at(i, j) + b.at(i, j);
    return res;
}
Mat& mul(double a, const Mat& b, Mat& res) {
    res.setSize(b.m, b.n);

    for (int i = 0; i < res.m; ++i)
        for (int j = 0; j < res.n; ++j)
            res.at(i, j) = a * b.at(i, j);
    return res;
}
Mat& neg(const Mat& a, Mat& res) { return mul(-1, a, res); }
Mat& sub(const Mat& a, const Mat& b, Mat& res) {
    return add(a, neg(b, res), res);
}
Mat& mul(const Mat& a, const Mat& b, Mat& res) {
    assert(a.n == b.m, "Sizes don't match, can't multiply");
    res.setSize(a.m, b.n);

    for (int i = 0; i < res.m; ++i)
        for (int j = 0; j < res.n; ++j) {
            res.at(i, j) = 0;
            for (int k = 0; k < a.n; ++k)
                res.at(i, j) += a.at(i, k) * b.at(k, j);
        }
    return res;
}
Mat& trans(const Mat& a, Mat& res) {
    assert(a.data != res.data, "Can't calculate the transpose inplace");
    res.setSize(a.n, a.m);

    for (int i = 0; i < res.m; ++i)
        for (int j = 0; j < res.n; ++j)
            res.at(i, j) = a.at(j, i);
    return res;
}
```

Laborator 2

18. Scrieți funcții pentru implementarea operațiilor specifice pe vectori din $\mathbb{R}^n$: suma, diferența și produsul scalar al doi vectori, produsul dintre un vector și un scalar real, negativarea unui vector, norma euclidiană a unui vector, citirea de la tastură a celor n componente ale unui vector, afișarea componentelor vectorului sub forma unui n-uplu de elemente. Se vor folosi tablouri unidimensionale alocate dinamic.

```cpp
#include "utils.h"

#include <iostream>
#include <utility>
#include <cmath>

struct Vec {
    double *_begin, *_end;

    constexpr double* begin() { return _begin; }
    constexpr const double* begin() const { return _begin; }


    constexpr double* end() { return _end; }
    constexpr const double* end() const { return _end; }

    constexpr Vec() : _begin(nullptr), _end(nullptr) {}
    explicit Vec(size_t n) : _begin(new double[n]), _end(_begin+n) {}
    Vec(std::initializer_list<double> list) : Vec(getSize(list)) {
        auto it = _begin;
        for (const auto& v : list) *(it++) = v;
    }
    Vec(const Vec&) = delete;
    Vec(Vec&& rhs) noexcept
        : _begin(std::exchange(rhs._begin, nullptr)),
        _end(std::exchange(rhs._end, nullptr)) {}

    Vec& operator=(const Vec&) = delete;
    Vec& operator=(Vec&& rhs) noexcept {
        this->~Vec();
        _begin = std::exchange(rhs._begin, nullptr);
        _end = std::exchange(rhs._end, nullptr);
        return *this;
    }

    ~Vec() { delete[] _begin; }

    constexpr size_t size() const { return _end - _begin; }

    constexpr double& operator[](size_t i) { return _begin[i]; }
    constexpr double operator[](size_t i) const { return _begin[i]; }

    void setSize(size_t n) {
```

```cpp
            if (size() == n) return;
            *this = Vec(n);
        }

        friend std::ostream& operator<<(std::ostream& s, const Vec& v) {
            s << "(";
            double* it = v._begin;
            for (double* end = v._end - 1; it < end; ++it)
                s << *it << ", ";

            if (it < v._end) s << *it;

            return s << ")";
        }
        static Vec read() {
            Vec res(readSize("n"));
            for (auto& v: res) std::cin >> v;
            return res;
        }
        double norm() const;
};
void assertSizes(const Vec& a, const Vec& b) {
    assert(a.size() == b.size(), "Sizes don't match");
}

Vec& add(const Vec& a, const Vec& b, Vec& res) {
    assertSizes(a, b);
    res.setSize(a.size());
    auto aIt = a.begin();
    auto bIt = b.begin();
    for (auto& v : res) v = *(aIt++) + *(bIt++);
    return res;
}

Vec& mul(double a, const Vec& b, Vec& res) {
    res.setSize(b.size());
    auto it = b.begin();
    for (auto& v : res) v = a * (*(it++));
    return res;
}

Vec& neg(const Vec& b, Vec& res) { return mul(-1, b, res); }

Vec& sub(const Vec& a, const Vec& b, Vec& res) {
    return add(a, neg(b, res), res);
}
double dot(const Vec& a, const Vec& b) {
    double res = 0;
    assertSizes(a, b);
    auto bIt = b.begin();
```

```cpp
    for (auto& v : a) res += v * (*(bIt++));
    return res;
}
double norm(const Vec& a) {
    return std::sqrt(dot(a, a));
}
double Vec::norm() const {
    return ::norm(*this);
}
```

Tablouri

Laborator 3

7. Folosind structurile de date `VECTOR` și `MATRICE` definite la curs și funcțiile necesare, rezolvați următorul sistem algebric liniar cu n ecuații și n necunoscute folosind metoda lui Gauß de eliminare.

$$\begin{cases} 2x_1 - x_2 = 1 \\ -x_1 + 2x_2 - x_3 = 1 \\ -x_2 + 2x_3 - x_4 = 1 \\ \quad\cdots\cdots\cdots \\ -x_{n-2} + 2x_{n-1} - x_n = 1 \\ -x_{n-1} + 2x_n = 1, \quad n \in \mathbb{N}, 2 \le n \le 50 \end{cases}.$$

```cpp
#include "utils.h"

#include "vector.h"
#include "matrix.h"

#include <iostream>
#include <cmath>

constexpr double eps = 1e-7;

Vec& mul(const Mat& m, const Vec& v, Vec& res) {
    assert(v.begin() != res.begin(), "Can't multiply inplace");
    assert(v.size() == size_t(m.n), "Sizes don't match");
    res.setSize(m.m);


    for (size_t i = 0; i < res.size(); ++i) {
        res[i] = 0;
        for (size_t k = 0; k < v.size(); ++k)
            res[i] += m.at(i, k) * v[k];
    }
    return res;
}

// A * X = b
struct System {
    Mat A;
```

```
      Vec b;

      System(int n, int m) : A(n, m), b(n) {}

      System(std::initializer_list<std::initializer_list<double>> A,
             std::initializer_list<double> b) : A(A), b(b) {
          assert(std::size_t(this->A.m) == this->b.size(),
                 "sizes don't match");
      }

      friend std::ostream& operator<<(std::ostream& s, const System& sys) {
          s << "System " << sys.A.m << "x" << sys.A.n << ": \n";
          auto& A = sys.A;
          for (int i = 0; i < A.m; ++i) {
              s << "{";
              for (int j = 0; j < A.n; ++j) {
                  //showpos shows a '+' in front of positive numbers
                  if (std::abs(A.at(i, j)) > eps)
                      s << std::showpos << A.at(i, j)
                        << std::noshowpos << "*x" << (j+1) << " ";
              }
              s<< "= " << sys.b[i] << "\n";
          }

          return s;
      }

      // L_i += f * L_j
      void addLines(int i, double f, int j) {
          for (int k = 0; k < A.n; ++k) {
              A.at(i, k) += f * A.at(j, k);
          }
          b[i] += f * b[j];
      }

      // L_i *= f
      void multiplyLine(int i, double f) {
          for (int k = 0; k < A.n; ++k) {
              A.at(i, k) *= f;
          }
          b[i] *= f;
      }

      Vec solveTriangulated() {
          for (int i = A.m-1; i > 0; --i) {
              addLines(i-1, -A.at(i-1, i), i);
              multiplyLine(i, 1 / A.at(i, i));
          }
          return b;
      }
```

```cpp
    bool checkSolution(const Vec& x) const {
        Vec r;
        mul(A, x, r);
        sub(b, r, r);
        for (auto& v : r) {
            if (std::abs(v) > eps) return false;
        }
        return true;
    }

    static Vec solveCustom(int n) {
        System s(n, n);
        for (auto& v : s.A) v = 0;
        s.A.at(0,0) = 2;
        s.A.at(0,1) = -1;
        s.b[0] = 1;
        for (int i = 1; i < n - 1; ++i) {
            s.b[i] = 1;

            s.A.at(i,i-1) = -1;
            s.A.at(i,i) = 2;
            s.A.at(i,i+1) = -1;
        }
        s.b[n-1] = 1;
        s.A.at(n-1, n-2) = -1;
        s.A.at(n-1, n-1) = 2;
        s.customTriangulate();
        return s.solveTriangulated();
    }
    void customTriangulate() {
        multiplyLine(0, 1 / A.at(0, 0));
        for (int i = 1; i < A.m; ++i) {
            addLines(i, 1, i-1);
            multiplyLine(i, 1 / A.at(i, i));
        }
    }
};

int main() {
    try {
        int n = readSize("n", 2, 51);
        std::cout << "x = "<< System::solveCustom(n) << "\n";
    } catch (std::exception& e) {
        std::cerr << "Error" << e.what() << "\n";
        return 1;
    }
    return 0;
}
```