Cuprins

Alocarea dinamica a memoriei. Tipuri specifice. **2**
Laborator 1 2
Laborator 2 5

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 <iostream>
#include <cmath>

constexpr int MAX_SZ = 8;
int readSize(const char* name) {
    int res;
    do {
        std::cout << name << ": ";
        std::cin >> res;
    } while (res <= 0 || res >= MAX_SZ);
    return res;
}

void assert(bool cond, const char* msg) {
    if (!cond) throw std::logic_error(msg);
}

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) {}

```

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

[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 $j = 1, ..., n$.

```
    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;
    }
    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;
    }
```

```
75  public:
76      double norm1() const { return normImpl<Type::Row>(n, m); }
77      double normInf() const { return normImpl<Type::Col>(m, n); }
78      double normF() const {
79          double res = 0;
80          for (int i = 0; i < m; ++i)
81              for (int j = 0; j < n; ++j)
82                  res += at(i, j) * at(i, j);
83
84          return std::sqrt(res);
85      }
86      bool isStrictlyRowDiagonallyDominant() const {
87          return isStrictlyDiagonallyDominantImpl<Type::Row>();
88      }
89      bool isStrictlyColDiagonallyDominant() const {
90          return isStrictlyDiagonallyDominantImpl<Type::Col>();
91      }
92  };
93
94  Mat& add(const Mat& a, const Mat& b, Mat& res) {
95      assert(a.m == b.m && a.n == b.n, "Sizes don't match, can't add");
96      res.setSize(a.m, a.n);
97      for (int i = 0; i < a.m; ++i)
98          for (int j = 0; j < a.n; ++j)
99              res.at(i, j) = a.at(i, j) + b.at(i, j);
100     return res;
101 }
102 Mat& mul(double a, const Mat& b, Mat& res) {
103     res.setSize(b.m, b.n);
104
105     for (int i = 0; i < res.m; ++i)
106         for (int j = 0; j < res.n; ++j)
107             res.at(i, j) = a * b.at(i, j);
108     return res;
109 }
110 Mat& neg(const Mat& a, Mat& res) { return mul(-1, a, res); }
111 Mat& sub(const Mat& a, const Mat& b, Mat& res) {
112     return add(a, neg(b, res), res);
113 }
114 Mat& mul(const Mat& a, const Mat& b, Mat& res) {
115     assert(a.n == b.m, "Sizes don't match, can't multiply");
116     res.setSize(a.m, b.n);
117
118     for (int i = 0; i < res.m; ++i)
119         for (int j = 0; j < res.n; ++j) {
120             res.at(i, j) = 0;
121             for (int k = 0; k < a.n; ++k)
122                 res.at(i, j) += a.at(i, k) * b.at(k, j);
123         }
124     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.

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

constexpr size_t getSize(const std::initializer_list<double>& l) {
    size_t n = 0;
    auto it = l.begin();
    auto end = l.end();
    while (it++ != end) ++n;
    return n;
}
constexpr int MAX_SZ = 256;
size_t readSize(const char* name) {
    int res;
    do {
        std::cout << name << ": ";
        std::cin >> res;
    } while (res <= 0 || res >= MAX_SZ);
    return res;
}

void assert(bool cond, const char* msg) {
    if (!cond) throw std::logic_error(msg);
}

struct Vec {
    double *_begin, *_end;

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

```

```
32
33      constexpr double* end() { return _end; }
34      constexpr const double* end() const { return _end; }
35
36      constexpr Vec() : _begin(nullptr), _end(nullptr) {}
37      explicit Vec(size_t n) : _begin(new double[n]), _end(_begin+n) {}
38      Vec(std::initializer_list<double> list) : Vec(getSize(list)) {
39          auto it = _begin;
40          for (const auto& v : list) *(it++) = v;
41      }
42      Vec(const Vec&) = delete;
43      Vec(Vec&& rhs) noexcept
44          : _begin(std::exchange(rhs._begin, nullptr)),
45          _end(std::exchange(rhs._end, nullptr)) {}
46
47      Vec& operator=(const Vec&) = delete;
48      Vec& operator=(Vec&& rhs) noexcept {
49          this->~Vec();
50          _begin = std::exchange(rhs._begin, nullptr);
51          _end = std::exchange(rhs._end, nullptr);
52          return *this;
53      }
54
55      ~Vec() { delete[] _begin; }
56
57      constexpr size_t size() const { return _end - _begin; }
58
59      constexpr double& operator[](size_t i) { return _begin[i]; }
60      constexpr double operator[](size_t i) const { return _begin[i]; }
61
62      void setSize(size_t n) {
63          if (size() == n) return;
64          *this = Vec(n);
65      }
66
67      friend std::ostream& operator<<(std::ostream& s, const Vec& v) {
68          s << "(";
69          double* it = v._begin;
70          for (double* end = v._end - 1; it < end; ++it)
71              s << *it << ", ";
72
73          if (it < v._end) s << *it;
74
75          return s << ")";
76      }
77      static Vec read() {
78          Vec res(readSize("n"));
79          for (auto& v: res) std::cin >> v;
80          return res;
81      }
```

```
82      double norm() const;
83  };
84  void assertSizes(const Vec& a, const Vec& b) {
85      assert(a.size() == b.size(), "Sizes don't match");
86  }
87
88  Vec& add(const Vec& a, const Vec& b, Vec& res) {
89      assertSizes(a, b);
90      res.setSize(a.size());
91      auto aIt = a.begin();
92      auto bIt = b.begin();
93      for (auto& v : res) v = *(aIt++) + *(bIt++);
94      return res;
95  }
96
97  Vec& mul(double a, const Vec& b, Vec& res) {
98      res.setSize(b.size());
99      auto it = b.begin();
100     for (auto& v : res) v = a * (*(it++));
101     return res;
102 }
103
104 Vec& neg(const Vec& b, Vec& res) { return mul(-1, b, res); }
105
106 Vec& sub(const Vec& a, const Vec& b, Vec& res) {
107     return add(a, neg(b, res), res);
108 }
109 double dot(const Vec& a, const Vec& b) {
110     double res = 0;
111     assertSizes(a, b);
112     auto bIt = b.begin();
113     for (auto& v : a) res += v * (*(bIt++));
114     return res;
115 }
116 double norm(const Vec& a) {
117     return std::sqrt(dot(a, a));
118 }
119 double Vec::norm() const {
120     return ::norm(*this);
121 }
```