ДОДАТОК А

СЛАЙДИ ПРЕЗЕНТАЦІЇ

ДОСЛІДЖЕННЯ КАСКАДНОЇ КОМПОЗИЦІЇ ДЛЯ АНАЛІЗУ ДЕСКРИПТОРНИХ СИСТЕМ

ВИКОНАВ:

СТ. ГР. ІПЗМ-18-4

ПРИХОДЬКО К.Д.

КЕРІВНИК:

ПРОФЕСОР РУТКАС А.Г.

АНАЛІЗ ПРОБЛЕМНОЇ ОБЛАСТІ

СФЕРИ, ДЕ ВИКОРИСТОВУЮТЬСЯ ДЕСКРИПТОРНІ СИСТЕМИ

- СИСТЕМИ ЕЛЕКТРОЖИВЛЕННЯ
- ЕЛЕКТРИЧНІ МЕРЕЖІ
- АЕРОКОСМІЧНА ІНЖЕНЕРІЯ
- ХІМІЧНІ ПРОЦЕСИ
- СОЦІАЛЬНО-ЕКОНОМІЧНІ СИСТЕМИ
- МЕРЕЖЕВИЙ АНАЛІЗ
- БІОЛОГІЧНІ СИСТЕМИ
- АНАЛІЗ ЧАСОВИХ РЯДІВ

2

АНАЛІЗ ПРОБЛЕМНОЇ ОБЛАСТІ

ЯК ОБЧИСЛЮЮТЬСЯ ДЕСКРИПТОРНІ СИСТЕМИ

- СКЛАДНО ТА ПОТРЕБУЄ РЕСУРСІВ
- НЕ МАЄ ЗАГАЛЬНОГО ПІДХОДУ
- ЯКЩО СИСТЕМА СКЛАДНА, КОЖНА СКЛАДОВА ОБЧИСЛЮЄТЬСЯ ОКРЕМО
- КОМПОЗИЦІЯ ДЕСКРИПТОРНИХ СИСТЕМ

3

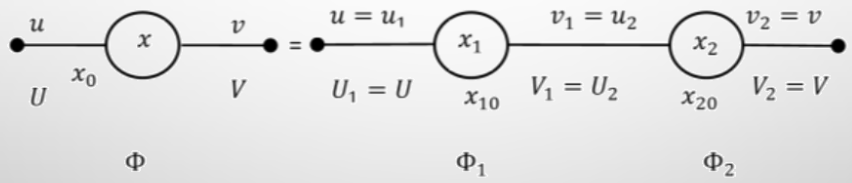
ЗАГАЛЬНЕ РІВНЯННЯ ДЛЯ СТАНУ X ТА ВИХОДУ V

$$\frac{d}{dt}(Ax) + Bx(t) = Fu(t), 0 \le t \le T,$$

$$v(t) = \frac{d}{dt}(Mx) + Nx + Ku, 0 \le t \le T$$

4

АЛЬТЕРНАТИВНИЙ МЕТОД КАСКАДНОЇ КОМПОЗИЦІЇ ДЕСКРИПТОРНИХ СИСТЕМ



Візуалізація станів систем при послідовній композиції

$$u(t) = u_1(t), \qquad x_0 = x_{10} + x_{20}, \qquad x(t) = x_1(t) + x_2(t)$$

5

ОПЕРАТОРНА ФОРМА СИСТЕМ

$$\frac{d}{dt}(A_1 x_1) + B_1 x_1(t) = F_1 u_1(t),$$

$$v_1(t) = \frac{d}{dt}(M_1 x_1) + N_1 x_1 + K_1 u_1$$

$$\frac{d}{dt}(A_2 x_2) + B_2 x_2(t) = F_2 u_2(t),$$

$$v_2(t) = \frac{d}{dt}(M_2 x_2) + N_2 x_2 + K_2 u_2$$

$$\frac{d}{dt}(A_2 x_2) + B_2 x_2(t) = F_2\left(\frac{d}{dt}(M_1 x_1) + N_1 x_1 + K_1 u_1\right)$$

6

ОПЕРАТОРИ В БЛОЧНІЙ ФОРМІ

$$A = \begin{bmatrix} A_1 & 0 \\ -F_2 M_1 & A_2 \end{bmatrix},$$

$$B = \begin{bmatrix} B_1 & 0 \\ -F_2 N_1 & B_2 \end{bmatrix},$$

$$F = \begin{bmatrix} F_1 \\ F_2 K_1 \end{bmatrix},$$

$$M = [K_2 M_1 \quad M_2],$$

$$N = [K_2 N_1 \quad N_2],$$

$$K = K_2 K_1$$

7

РЕАЛІЗАЦІЯ АЛГОРИТМУ

- ЗЧИТУВАННЯ КОЕФІЦІЄНТІВ ВИХІДНИХ СИСТЕМ
- ПІДСТАНОВКА КОЕФІЦІЄНТІВ ДО ВИРАЗІВ
- ОБЧИСЛЕННЯ ЕЛЕМЕТІВ
- ЗАПИС РЕЗУЛЬТУЮЧИХ КОЕФІЦІЄНТІВ

8

УНІВЕРСАЛЬНИЙ МЕТОД МНОЖЕННЯ МАТРИЦЬ

```
std::vector<Telement> multiply(std::vector<Telement> m, std::vector<Telement> n)
{
  int row = sqrt(m.size());
  int col = n.size()/row;
  std::vector<Telement> result;

  for(int i = 0; i < row; i++)
    for(int  j = 0; j < col; j++)
    {
      Telement element;

      for(int k = 0; k < row; k++)
      {
        element = element + m[k + row * i] * n[j + col * k];
      }
      result.push_back(element);
    }

  return result;
}
```

9

ОБЧИСЛЕННЯ КОЕФІЦІЄНТІВ

```
{
int size = sqrt(A2.size());

A.resize((size + 1) * (size + 1));
A[0] = A1[0];
std::vector<Telement> F2M1 = multiply(F2, M1);
for(int i = 1; i < size + 1; i++)
{
  A[i * (size + 1)] = F2M1[i - 1];
  A[i * (size + 1)].number = -A[i * (size + 1)].number;
}
for(int i = 0; i < size; i++)
  for(int  j = 0; j < size; j++)
  {
    A[j + 1 + (i + 1) * (size + 1)] = A2[j + i * size];
  }
}
```

РОЗРАХУНОК БЛОКУ А

```
{
int size = sqrt(B2.size());

B.resize((size + 1) * (size + 1));
B[0] = B1[0];
std::vector<Telement> F2N1 = multiply(F2, N1);
for(int i = 1; i < size + 1; i++)
{
  B[i * (size + 1)] = F2N1[i - 1];
  B[i * (size + 1)].number = -B[i * (size + 1)].number;
}

for(int i = 0; i < size; i++)
  for(int  j = 0; j < size; j++)
  {
    B[j + 1 + (i + 1) * (size + 1)] = B2[j + i * size];
  }
}
```

РОЗРАХУНОК БЛОКУ В

10

ОБЧИСЛЕННЯ КОЕФІЦІЄНТІВ

```
{
F.insert(F.end(), F1.begin(), F1.end());

std::vector<Telement> F2K1 = multiply(F2, K2);
F.insert(F.end(), F2K1.begin(), F2K1.end());
}
```

РОЗРАХУНОК БЛОКУ F

```
{
int size = sqrt(K2.size());

std::vector<Telement> K2N1 = multiply(K2, N1);

N.resize(size * (size + 1));
for(int i = 0; i < size; i++)
  {
    N[i * size] = K2N1[i];
  }

size = sqrt(N2.size());

for(int i = 0; i < size; i++)
  for(int  j = 0; j < size; j++)
  {
    N[j + i + i * (size + 1)] = N2[j + i * size];
  }
}
```

РОЗРАХУНОК БЛОКУ N

```
{
int size = sqrt(K2.size());

std::vector<Telement> K2M1 = multiply(K2, M1);

M.resize(size * (size + 1));

for(int i = 0; i < size; i++)
  {
    M[i * size] = K2M1[i];
  }

size = sqrt(M2.size());

for(int i = 0; i < size; i++)
  for(int  j = 0; j < size; j++)
  {
    M[j + 1 + i * (size + 1)] = M2[j + i * size];
  }
}
```

РОЗРАХУНОК БЛОКУ M

```
{
  K = multiply(K2, K2);
}
```

РОЗРАХУНОК БЛОКУ K

11

ВИСНОВКИ

- ПРОВЕДЕНО АНАЛІЗ ДОСЛІДЖЕНЬ ПОСЛІДОВНОЇ КОМПОЗИЦІЇ ДЕСКРИПТОРНИХ СИСТЕМ КЕРУВАННЯ
- ПРОАНАЛІЗОВАНО АЛЬТЕРНАТИВНИЙ МЕТОД КАСКАДНОЇ КОМПОЗИЦІЇ ДЕСКРИПТОРНИХ СИСТЕМ
- РОЗРОБЛЕНО АЛГОРИТМ, ЩО РОБИТЬ МОЖЛИВИМ АВТОМАТИЧНИЙ РОЗРАХУНОК КОЕФІЦІЄНТІВ ПІДСУМКОВОЇ ОБ'ЄДНАННОЇ ДЕСКРИПТОРНОЇ СИСТЕМИ
- ВИКОНАНО ПРОГРАМНУ РЕАЛІЗАЦІЮ АЛГОРИТМУ

12

ДОДАТОК Б

ЛІСТИНГ КОДУ

Файл MatrixComposer.cpp:

```
#include "pch.h"
#include <iostream>
#include <string>
#include <fstream>
#include <sstream>
#include <vector>
#include <map>

std::string inputFile;
std::string outputFile;

struct Telement
{
  float number = 0;
  std::map<std::string, float> var;

  Telement operator*(const Telement& other) const
  {
    Telement result = *this;
    if(number == 0 && !var.empty())
    {
      result.number = 1;
    }
```

```
int otherNumber = other.number;
if(other.number == 0 && !other.var.empty())
{
  otherNumber = 1;
}

result.number *= otherNumber;
std::string total;
for(auto & x:other.var)
{
  if(x.second == 0)
    continue;

  result.number *= x.second;
  total += x.first;
}

for(auto & x:var)
{
  if(x.second == 0)
    continue;

  result.number *= x.second;

  if(other.var.find(x.first) != other.var.end())
  {
    total += x.first;
  }
```

```
    }

    result.var.clear();
    if(!total.empty())
    {
      result.var[total] = number;
      result.number = 0;
    }

    return result;
  }

  Telement operator+(const Telement& other) const
  {
    Telement result = *this;

    result.number += other.number;
    for(auto & x:other.var)
    {
      result.var[x.first] += x.second;
    }

    return result;
  }
};

struct Tresult
{
```

```
  std::vector<Telement> A;
  std::vector<Telement> B;
  std::vector<Telement> F;
  std::vector<Telement> M;
  std::vector<Telement> N;
  std::vector<Telement> K;
};

std::vector<Telement> parseVector(std::string data){

  std::istringstream parser(data);
  std::vector<Telement> result;
  std::string element;

  while(parser>>element)
  {
    bool isDigit = true;
    for(auto c:element)
    {
      if(c < '0' || c > '9')
      {
        isDigit = false;
        break;
      }
    }
    Telement temp;
    if(isDigit)
    {
```

```
      temp.number = std::stof(element);
    }
    else
    {
      temp.var[element] = 1;
    }
    result.push_back(temp);
  }

  return result;
}

std::vector<Telement> multiply(std::vector<Telement> m,
std::vector<Telement> n)
{
  int row = sqrt(m.size());
  int col = n.size()/row;
  std::vector<Telement> result;

  for(int i = 0; i < row; i++)
    for(int  j = 0; j < col; j++)
    {
      Telement element;

      for(int k = 0; k < row; k++)
      {
        element = element + m[k + row * i] * n[j + col * k];
      }
```

```
    result.push_back(element);
   }

  return result;
}

Tresult compose(std::string path)
{

  std::ifstream file;
  file.open(path.c_str());

  int count;
  file>>count;

  std::string data;
  std::getline(file, data);
  std::getline(file, data);
  std::vector<Telement> A2 = parseVector(data);

  std::getline(file, data);
  std::vector<Telement> B2 = parseVector(data);

  std::getline(file, data);
  std::vector<Telement> F2 = parseVector(data);

  std::getline(file, data);
  std::vector<Telement> M2 = parseVector(data);
```

```
std::getline(file, data);
std::vector<Telement> N2 = parseVector(data);

std::getline(file, data);
std::vector<Telement> K2 = parseVector(data);

std::getline(file, data);
std::vector<Telement> A1 = parseVector(data);

std::getline(file, data);
std::vector<Telement> B1 = parseVector(data);

std::getline(file, data);
std::vector<Telement> F1 = parseVector(data);

std::getline(file, data);
std::vector<Telement> M1 = parseVector(data);

std::getline(file, data);
std::vector<Telement> N1 = parseVector(data);

std::vector<Telement> A;
std::vector<Telement> B;
std::vector<Telement> F;
std::vector<Telement> M;
std::vector<Telement> N;
std::vector<Telement> K;
```

```
{
int size = sqrt(A2.size());

A.resize((size + 1) * (size + 1));
A[0] = A1[0];
std::vector<Telement> F2M1 = multiply(F2, M1);
for(int i = 1; i < size + 1; i++)
{
  A[i * (size + 1)] = F2M1[i - 1];
  A[i * (size + 1)].number = -A[i * (size + 1)].number;
}
for(int i = 0; i < size; i++)
  for(int  j = 0; j < size; j++)
  {
    A[j + 1 + (i + 1) * (size + 1)] = A2[j + i * size];
  }
}

{
int size = sqrt(B2.size());

B.resize((size + 1) * (size + 1));
B[0] = B1[0];
std::vector<Telement> F2N1 = multiply(F2, N1);
for(int i = 1; i < size + 1; i++)
{
  B[i * (size + 1)] = F2N1[i - 1];
```

```
  B[i * (size + 1)].number = -B[i * (size + 1)].number;
}

for(int i = 0; i < size; i++)
  for(int  j = 0; j < size; j++)
  {
    B[j + 1 + (i + 1) * (size + 1)] = B2[j + i * size];
  }
}

{
F.insert(F.end(), F1.begin(), F1.end());

std::vector<Telement> F2K1 = multiply(F2, K2);
F.insert(F.end(), F2K1.begin(), F2K1.end());
}

{
int size = sqrt(K2.size());

std::vector<Telement> K2M1 = multiply(K2, M1);

M.resize(size * (size + 1));

for(int i = 0; i < size; i++)
  {
    M[i * size] = K2M1[i];
  }
```

```
size = sqrt(M2.size());

for(int i = 0; i < size; i++)
  for(int  j = 0; j < size; j++)
  {
    M[j + 1 + i * (size + 1)] = M2[j + i * size];
  }
}

{
int size = sqrt(K2.size());

std::vector<Telement> K2N1 = multiply(K2, N1);

N.resize(size * (size + 1));
for(int i = 0; i < size; i++)
  {
    N[i * size] = K2N1[i];
  }

size = sqrt(N2.size());

for(int i = 0; i < size; i++)
  for(int  j = 0; j < size; j++)
  {
    N[j + 1 + i * (size + 1)] = N2[j + i * size];
  }
```

```
  }

  {
    K = multiply(K2, K2);
  }

  return{
      A, B, F, M, N, K
    };
}

void printVector(std::vector<Telement> data, std::string name,
std::ofstream & file)
{
  std::cout<<name<<" = ";
  file<<name<<" = ";

  for(auto x:data)
  {
    if(x.number != 0 || x.var.empty())
    {
      std::cout<<x.number;
      file<<x.number;
    }

    for(auto y:x.var)
    {
      if(y.second != 1)
```

```
      {
        std::cout<<y.second;
        file<<y.second;
      }

      if(y.second != 0)
      {
        std::cout<<y.first;
        file<<y.first;
      }
    }

    std::cout<<" ";
    file<<" ";
  }

  std::cout<<"\n";
  file<<"\n";
}

int main(int argc, const char * const * argv)
{
    inputFile = argv[1];
    outputFile = argv[2];

    Tresult result = compose(inputFile);

    std::ofstream output;
```

```
    output.open(outputFile);

    printVector(result.A, "A", output);
    printVector(result.B, "B", output);
    printVector(result.F, "F", output);
    printVector(result.M, "M", output);
    printVector(result.N, "N", output);
    printVector(result.K, "K", output);

    output.close();

    return 0;
}
```