Gaussova metoda řešení soustav lineárních rovnic

1 Úvod

Soustavu lineárních rovnic můžeme řešit mnoha způsoby. Pokud jde o soustavy jednoduché, často vyjadřujeme neznámé z jednotlivých rovnic a dosazujeme je do dalších. Můžeme to vyzkoušet třeba s touto soustavou:

$$
\begin{array}{lrcl}
(1) & x_1 + 2x_2 + 3x_3 & = & 4 \\
(2) & 2x_1 + x_2 & = & 4 \\
(3) & x_1 + 4x_2 + 2x_3 & = & 3 \\
\hline
(1) & x_1 & = & 4 - 2x_2 - 3x_3 \\
(2) & \frac{4-3x_2}{6} & = & x_3 \\
(3) & 4 - 2x_2 - 3_x 3 + 4x_2 + 2x_3 & = & 3 \\
\hline
(3) & 1 + 2x_2 - x_3 & = & 0 \\
\hline
(3) & 6 + 12x_2 - 4 + 3_x 2 & = & 0 \\
\hline
(3) & -\frac{2}{15} & = & x_2 \\
(2) & \frac{11}{15} & = & x_3 \\
(1) & \frac{31}{15} & = & x_1
\end{array}
$$

Tuto soustavu není obtížné vyřešit, dosazením můžeme ověřit, jestli je řešení správné. (Když to uděláme, zjistíme, že správné je.) Pokud však budeme potřebovat vyřešit složitou soustavu rovnic o mnoha neznámých, bude lepší celý výpočet rovnou přenechat počítači. Napsat program, který by uměl upravit výraz či vyjádřit neznámou, není právě jednoduché, proto se k řešení takových soustav užívá tzv. *Gaussova eliminační metoda.*

2 Gaussova eliminační metoda

Jedná se o úpravu matice na tzv. *redukovaný stupňovitý* tvar. Přeloženo do obyčejného jazyka to znamená, že v této matici budou na diagonále samé jedničky[1]. Toho dosáhneme pomocí *elementárních řád-*

[1] Z toho vyplývá, že taková matice by měla být čtvercová. Pokud není, nedostaneme z ní jednotkovou matici.

kových operací. Jsou to tyto:

- záměna libovolných dvou řádků
- násobení kteréhokoli řádku nenulovým číslem (skalárem)
- přičtení jednoho řádku k jinému řádku

Souvislost těchto úprav s lineárními rovnicemi není na první pohled zřejmá. Z koeficientů vhodně zapsané soustavy lineárních rovnic (třeba takové, jako je ta v příkladu výše), můžeme sestavit matici. Pokud je rovnic tolik jako neznámých, vznikne tak matice s počtem sloupců o jedničku vyšším než řádků (v pravém sloupci budou absolutní členy z pravých stran rovnic[2]). Záměna řádků je obdoba záměny dvou rovnic. Libovolnou rovnici lze také vynásobit nenulovým číslem, tak jako řádek matice. Nakonec je také možné k jedné rovnici přičíst jinou, protože její levá a pravá strana jsou si rovny a k oběma stranám rovnice lze přičíst jakékoli číslo.

Matice soustavy lineárních rovnic, má-li řešení, se bude nakonec skládat z matice jednotkové a matice s jediným sloupcem tvořeným kořeny soustavy. Lze to předvést na výše uvedené soustavě:

$$\left(\begin{array}{ccc|c} 1 & 2 & 3 & 4 \\ 2 & 1 & 0 & 4 \\ 1 & 4 & 2 & 3 \end{array}\right) \sim \left(\begin{array}{ccc|c} 1 & 2 & 3 & 4 \\ 0 & -3 & -6 & -4 \\ 0 & 2 & -1 & -1 \end{array}\right) \sim$$

$$\left(\begin{array}{ccc|c} 1 & 0 & 4 & 5 \\ 0 & 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & 0 & -\frac{15}{2} & -\frac{11}{2} \end{array}\right) \sim \left(\begin{array}{ccc|c} 1 & 0 & 0 & \frac{31}{15} \\ 0 & 1 & 0 & -\frac{2}{15} \\ 0 & 0 & 1 & \frac{11}{15} \end{array}\right)$$

Nyní je dobré se zmínit, jak lze poznat, že soustava nemá řešení, nebo jich má nekonečně mnoho. Pokud při úpravách dostaneme řádek, který má jedinou nenulovou hodnotu v pravém sloupci, nemá soustava řešení (to odpovídá nulové levé a nenulové pravé straně – levá strana se pravé nerovná). Může se také stát, že lze dva řádky od sebe beze zbytku odečíst – potom je jedna rovnice součtem násobků ostatních[3]. Zbude-li řádků méně, než kolik má část matice pro levou stranu soustavy sloupců, má soustava řešení nekonečně mnoho.

Těžko říci, který způsob výpočtu je jednodušší, ten maticový má ale jednu výhodu – lze poměrně jednoduše vytvořit program, který bude soustavu tímto způsobem řešit.

[2] Pro úplnost je třeba dodat, že pokud jsou tyto absolutní členy nulové, jde o *homogenní soustavu* a také o *homogenní matici.*

[3] Je jejich lineární kombinací.

3 Gaussova eliminace pomocí počítače

3.1 Zavedení základního maticového počtu v C++

Gaussovu eliminační metodu můžeme naprogramovat všelijak. Zde bude rozebráno provedení této metody v jazyce C++. Pak je možné program postavit na operacích s maticemi, jako je sčítání matic, násobení skalárem a tak podobně.

To je možné provést například zavedením třídy `CMatrix`, která umožnuje součet a rozdíl matic stejného typu, násobení skalárem, součin dvou matic vhodného typu, porovnání matic a transponování matic. Tyto operace lze naprogramovat velmi snadno.

```
class CMatrix
{
public:
  /* základní konstruktor */
  CMatrix();
  /* vytvoří prázdnou matici */
  CMatrix(int _m, int _n);
  /* zkopíruje matici */
  CMatrix(const CMatrix& Matrix);
  /* vytvoří matici na základě řetězce */
  CMatrix(int _m, int _n, char *_matrix);
  virtual ~CMatrix();

  /* naplní matici reálnými čísly (musí mít des. tečku!) */
  void Setup(double d, ...);
  /* naplní matici celými čísly */
  void SetupInt(long int d, ...);
  /* naplní matici čísly z řetězce */
  void SetupStr(char *_matrix);

  /* vrátí hodnotu v j-tém sloupci i-tého řádku */
  double GetField(int i, int j) const;
  /* nastaví hodnotu v j-tém sloupci i-tého řádku */
  void SetField(int i, int j, double d);

  /* vrátí rozměprivatery matice */
  void GetSize(int &_m, int &_n) const;
  int GetRows();
  int GetCols();

  /* vrátí hodnotu všech prvků matice i její rozměry */
  double *GetMatrix();
  double *GetMatrix(int &_m, int &_n);

  /* operátory */
  CMatrix& operator =  (const CMatrix& rvalue);
  CMatrix& operator += (CMatrix& rvalue);
  CMatrix& operator -= (CMatrix& rvalue);
  CMatrix& operator *= (double rvalue);
  CMatrix& operator *= (CMatrix& rvalue);

  /* vytvoří kopii a transponuje ji */
  CMatrix MakeTransposed();
```

```
  /* transponuje tuto matici */
  void Transpose();

  /* součet matic */
  friend CMatrix operator + (const CMatrix& left, const CMatrix& right);
  /* násobení matice skalárem (zleva i zprava) */
  friend CMatrix operator * (const CMatrix& left, double right);
  friend CMatrix operator * (double left, const CMatrix& right);
  /* opačná matice, tj. * (-1) */
  friend CMatrix operator - (const CMatrix& right);
  /* odčítání matic */
  friend CMatrix operator - (const CMatrix& left, const CMatrix& right);
  /* součin matic */
  friend CMatrix operator * (const CMatrix& left, const CMatrix& right);

  /* porovnání matic */
  friend int operator == (const CMatrix& left, const CMatrix& right);
  friend int operator != (const CMatrix& left, const CMatrix& right);
private:
  int m, n;
  double *matrix;

  /* vytvoří matici (m) x (n) */
  void CreateMatrix(int _m, int _n);
  /* vytvoří kopii jiné matice */
  void CreateMatrix(const CMatrix& Matrix);
  /* odstraní matici z paměti */
  void DestroyMatrix();
};

/* třída pro výjimku při chybě při výpočtu */
class CMatrixError: public std::exception
{
public:
  CMatrixError(int _id);
  virtual ~CMatrixError() throw();
  int ID();
  virtual const char* what() const throw();

  CMatrixError& operator = (const CMatrixError& rvalue);
  friend int operator == (CMatrixError& left, CMatrixError& right);
private:
  int id;
};
```

Na těchto funkcích se ještě nevyplatí eliminaci stavět, k tomu se lépe hodí takováto třída `CEnhancedMatrix`:

```
/* vytvoření ručně rozdělené blokové matice */
#define BM_USERDEFINED 0
/* vytvoření blokové matice řádky/sloupci dané šířky */
#define BM_SIZE        -1
/* vytvoření blokové matice s daným počtem řádků/sloupců */
#define BM_COUNT       -2

class CEnhancedMatrix : public CMatrix
{
public:
  int m_blocks, n_blocks;
  CEnhancedMatrix **BlockMatrix;
```

```
  /* konstruktory a destruktory jako u předchozí třídy */
  CEnhancedMatrix();
  CEnhancedMatrix(int _m, int _n);
  CEnhancedMatrix(const CMatrix& Matrix);
  CEnhancedMatrix(int _m, int _n, char *_matrix);
  virtual ~CEnhancedMatrix();

  /* Vytvoří z této matice blokovou matici.
     Toto rozdělí matici na bloky daných rozměrů:
       SetupBlockMatrix(velikost_m_bloku_0, ..., BM_USERDEFINED,
         velikost_m_bloku_0, ..., BM_USER_DEFINED);
     Takto se dělí matice na řádky/sloupce dané velikosti:
       SetupBlockMatrix(výška_bloku, BM_SIZE, šířka_bloku, BM_SIZE);
     A tímhle způsobem se rozdělí matice na daný počet bloků:
       SetupBlockMatrix(počet_řádků, BM_COUNT, počet_sloupců, BM_SIZE);
  */
  void SetupBlockMatrix(int m0, ...);
  void SetupCustomBlockMatrix(int _m, int *m_list, int _n, int *n_list);
  /* Odstraní blokovou matici z paměti */
  void DestroyBlockMatrix();
  /* Zkopíruje hodnoty z blokové matice do původní matice */
  void UpdateMatrix();

  /* Vrátí rozměry */
  void GetBlocksSize(int &_m, int &_n);
  int GetBlocksRows();
  int GetBlocksCols();

  /* Vrátí blok z matice */
  CEnhancedMatrix& GetBlockMatrix(int _m, int _n);

  /* Prohodí dva řádky blokové matice */
  void ExchangeBlockRows(int _m1, int _m2);
};
```

3.2 Vlastní Gaussova eliminace

Nyní už je možné se dostat ke Gaussově eliminaci. K tomuto účelu mějme například třídu `CGaussMatrix`:

```
class CGaussMatrix : public CEnhancedMatrix
{
public:
  /* konstruktory a destruktory jsou obdobné jako u předchozí třídy */
  CGaussMatrix();
  CGaussMatrix(int _m, int _n);
  CGaussMatrix(const CMatrix& Matrix);
  CGaussMatrix(int _m, int _n, char *_matrix);
  virtual ~CGaussMatrix();

  /* Provede eliminaci na stanoveném počtu sloupců */
  void Eliminate(int columns);
  /* Vyřeší soustavu lineárních rovnic danou touto maticí. */
  void LinearEquation();
};
```

Zde následuje zdrojový text této třídy:

```
#include <cstdio>
#include <iostream>

#define Uses_EnhancedMatrix
#define Uses_GaussMatrix
#include "matrix"

using namespace std;

CGaussMatrix::CGaussMatrix() :
  CEnhancedMatrix() { }
CGaussMatrix::CGaussMatrix(int _m, int _n) :
  CEnhancedMatrix(_m, _n) { }
CGaussMatrix::CGaussMatrix(const CMatrix& Matrix) :
  CEnhancedMatrix(Matrix) { }
CGaussMatrix::CGaussMatrix(int _m, int _n, char *_matrix) :
  CEnhancedMatrix(_m, _n, _matrix) { }
CGaussMatrix::~CGaussMatrix() { }

/* Vlastní eliminace */
void CGaussMatrix::Eliminate(int columns)
{
  int i, j;
  CEnhancedMatrix tmp;

  /* zabraň přetečení */
  if (columns > GetCols())
    throw CMatrixError(M_OUT_OF_BOUNDS);

  /* rozděl matici na řádkové bloky */
  SetupBlockMatrix(1, BM_SIZE, 1, BM_COUNT);

  /* první průchod - převod na základní schodivitý tvar */
  for (i = 0; i < columns; i++)
  {
    /* nechť mají všechny řádky na i-tém místě 1 nebo 0 */
    for (j = i; j < GetRows(); j++)
    {
      /* pokud má j-tý řádek 0 na i-tém místě, posuň jej dolů */
      if ((GetBlockMatrix(j, 0).GetField(0, i) == 0)
        && (j < GetRows() - 1))
        ExchangeBlockRows(j, j + 1);
      /* nemá-li tam nulu, poděl celý řádek prvním nenulovým číslem */
      if (GetBlockMatrix(j, 0).GetField(0, i) != 0)
        GetBlockMatrix(j, 0) *=
          1.0 / GetBlockMatrix(j, 0).GetField(0, i);
    }

    /* nyní odečti i-tý řádek od všech ostatních */
    for (j = i+1; j < GetRows(); j++)
    {
      /* ...ale jen od takových, které nemají v i-tém sloupci 0 */
      if (GetBlockMatrix(j, 0).GetField(0, i) != 0)
        GetBlockMatrix(j, 0) -= GetBlockMatrix(i, 0);
    }
  }

  /* druhý průchod - zpětná eliminace */
  int maxrows = (columns > GetRows() ? GetRows() : columns);

  for (i = maxrows - 1; i >= 0; i--)
```

```
  {
    /* pokud má i-tý řádek na i-tém místě nulu, není možné jeho
       vhodný násobek odečíst od řádků nad ním */
    if (GetBlockMatrix(i, 0).GetField(0, i) != 0)
      /* odečti i-tý řádek od těch nad ním 'i' from all other rows */
      for (j = i - 1; j >= 0; j--)
      {
        tmp = GetBlockMatrix(i, 0) *
          (GetBlockMatrix(j, 0).GetField(0, i) /
          GetBlockMatrix(i, 0).GetField(0, i));
        GetBlockMatrix(j, 0) -= tmp;
      }
  }

  /* zkopíruj výsledek do původní matice */
  UpdateMatrix();

  /* odstraň blokovou matici */
  DestroyBlockMatrix();
}

/* vyřeší lineární soustavu zadanou touto maticí */
void CGaussMatrix::LinearEquation()
{
  Eliminate(GetCols() - 1);
}
```

V programu je k řešení použit mírně odlišný postup. Ve funkci `CGaussMatrix::Eliminate`je matice nejprve rozdělena na bloky – jednotlivé řádky. Ty pak jsou v prvním průchodu děleny svými prvními nenulovými prvky, aby začínaly jedničkou, což je pro řešení pomocí počítače nejvhodnější. Poté jsou postupně vyšší řádky odečítány od nižších. Záměna dvou řádků tak není téměř třeba – je použita jen k odsunutí nulových řádků na spodek matice. Potom dojde na druhý průchod, při němž se nulují všechny prvky po prvním nenulovém prvku v každém řádku, aby v matici na každém řádku zůstala jen jedna jednička. Ne každá matice je však ideální, proto je eliminace provedena pouze na její čtvercové části.

Za zmínku ještě stojí, že u této funkce je potřeba stanovit, u kterých sloupců má být eliminace provedena. Tuto funkci tak můžeme použít nejen k řešení soustavy lineárních rovnic, ale i k určení inverzní matice (nejprve je nutno tu původní rozšířit zprava jednotkovou maticí).

Soustavu lineárních rovnic potom řeší pomocí předchozí metody funkce `CGaussMatrix::LinearEquation`. Z eliminace je však vyňat poslední sloupec – ten obsahuje absolutní členy jednotlivých rovnic. V tomto sloupci je potom řešení soustavy, pokud bylo nalezeno.

Pro úplnost je vhodné dodat časovou a paměťovou složitost funkce. Časová složitost je přibližně $O(N^2)$ a paměťová $M(N^2)$ pro matici $N \times N$.

3.3 Zadání vstupní matice a ověření výsledků

Pro jednoduchost a přehlednost je dobré ve zkušebním programu vytvořit matici z náhodných čísel takovou, aby měla sloupců o jeden více než řádků[4]. Můžeme to zapsat takto: $X = (x_{ij})_{n\times(n+1)}$. Zároveň si tuto matici můžeme rozdělit na bloky takto:

$$X = \left(\begin{array}{c|c} A & B \end{array} \right)$$

Při tom platí:

$$\begin{aligned} A &= (a_{ij})_{n\times n} \\ B &= (b_{ij})_{n\times 1} \end{aligned}$$

Tato bloková matice X odpovídá nehomogenní (když $B \neq 0$) soustavě n rovnic o n neznámých. Matematicky to lze zapsat jako:

$$A \cdot x = B$$

Po Gaussově eliminaci dostaneme:

$$\begin{aligned} \widetilde{A} \cdot x &= \widetilde{B} \\ \widetilde{A} &= (\delta_{ij}) \quad \delta_{ij} = \left\{ \begin{array}{ll} 1 \Leftrightarrow i & = j \\ 0 \Leftrightarrow i & \neq j \end{array} \right. \quad \ldots\text{jednotková matice} \end{aligned}$$

Protože $\widetilde{A}$ je jednotková matice a $\widetilde{B}$ řešení soustavy, dostaneme řešení $x = \widetilde{B}$.

Toto je zdrojový text zkušebního programu:

```
#include <cstdio>
#include <cmath>
#include <ctime>
#include <cstdlib>
#include <iostream>

#define Uses_Matrix
#include "matrix"

using namespace std;

#define USE_FILE 1
#define USE_RAND 2
int action = 0;
char infile[256] = { 0, };
int rand_m = 0, rand_n = 0;

/* vypíše matici na obrazovku */
void PrintMatrix(CMatrix& Matrix)
{
  int i, j, m, n;
  Matrix.GetSize(m, n);
  for (i = 0; i < m; i++)
```

[4] To splňuje např. matice z úvodního příkladu.

```
  {
    printf("( ");
    for (j = 0; j < n; j++) printf("%8.3f ", Matrix.GetField(i, j));
    if (i + 1 < m) printf(")\n"); else printf(") %ux%u\n", m, n);
  }
}

void checkparameters(int argc, char *argv[])
{
  int i;
  for (i = 1; i < argc; i++)
  {
    if (argv[i][0] == '-')
    {
      if (!strcmp(argv[i]+1, "file"))
      {
        if (i + 1 < argc)
        {
          action |= USE_FILE;
          strcpy(infile, argv[++i]);
        }
        else cout << "Pozor: jméno souboru chybí" << endl;
      }

      if (!strcmp(argv[i]+1, "rand"));
      {
        if (i + 1 < argc)
        {
          if (sscanf(argv[++i], "%d", &rand_m) == 1)
            action |= USE_RAND;
          else cout << "Pozor: chybí velikost matice" << endl;
        }

        if (i + 1 < argc)
        {
          if (sscanf(argv[++i], "%d", &rand_n) < 1)
            rand_n = 0;
        }
      }
    }
  }
}

int main(int argc, char *argv[])
{
  FILE *in;
  int m_rows, m_cols;
  int i, j;
  double r;

  srand(time(0));
  checkparameters(argc, argv);
  if (action == 0)
  {
    printf("Zkušební program na Gaussovu eliminaci.\n"
           "Návod: matrixtest <-parametr> [vstupní soubor]\n"
           "  -file vstupní_soubor - načte matici ze souboru\n"
           "  -rand m [n]          - vytvoří náhodnou matici m,n či m,m\n");

    return 0;
  }
```

```
try {
  CGaussMatrix Matrix;

  if (action & USE_FILE) /* file mode */
  {
    cout << "Matice je dána souborem \"" << infile << "\"..." << endl;
    in = fopen(infile, "r");
    if (in == NULL)
    {
      cout << "Nelze otevřít '" << infile << "'" << endl;
      return 1;
    }

    fscanf(in, "%d,%d", &m_rows, &m_cols);
    cout << "Matice (" << m_rows << "," << m_cols << ")" << endl;
    Matrix = CGaussMatrix(m_rows, m_cols);
    for (i = 0; i < m_rows; i++)
      for (j = 0; j < m_cols; j++)
      {
        fscanf(in, "%lf", &r);
        Matrix.SetField(i, j, r);
      }
    fclose(in);
  }

  if (action & USE_RAND) /* načíst ze souboru */
  {
    if (rand_m <= 0)
    {
      cout << "Error: incorrect count of rows: " << rand_m << endl;
      return 1;
    }
    if (rand_n <= 0)
      rand_n = rand_m + 1;
    m_rows = rand_m;
    m_cols = rand_n;

    cout << "Náhodná matice " << m_rows << "x" << m_cols << endl;

    Matrix = CGaussMatrix(m_rows, m_cols);
    for (i = 0; i < m_rows; i++)
      for (j = 0; j < m_cols; j++)
        Matrix.SetField(i, j, (double) rand() / (double) RAND_MAX);
  }

  PrintMatrix(Matrix);
  /* matice jiné než m,m+1 nelze eliminovat zcela */
  if (m_rows + 1 < m_cols)
  {
    cout << "Pozor - tuto matici nelze zcela eliminovat." << endl;
  }

  CGaussMatrix Matrix2 = Matrix;
  cout << "Počítá se..." << endl;
  Matrix2.LinearEquation();
  cout << "Výsledek:" << endl;
  PrintMatrix(Matrix2);

  if (m_rows + 1 >= m_cols)
  {
```

```
      int stop = 0, limit = m_rows;

      cout << "Ověření: ";
      if (m_rows + 1 != m_cols)
      {
        for (i = m_cols - 1; i < m_rows; i++)
          /* takovéto porovnání stačí - pokud řešení existuje,
             bude tady jistě nula */
          if (Matrix2.GetField(i, m_cols - 1) != 0.0)
          {
            cout << "nemá řešení" << endl;
            stop = 1;
            break;
          }
        limit = m_cols - 1;
      }

      if (!stop)
      {
        /* rozděl matici na bloky A a B */
        if (limit == m_rows)
        {
          Matrix.SetupBlockMatrix(1, BM_COUNT,
            m_cols - 1, 1, BM_USERDEFINED);
          Matrix2.SetupBlockMatrix(1, BM_COUNT,
            m_cols - 1, 1, BM_USERDEFINED);
        }
        else
        {
          Matrix.SetupBlockMatrix(limit, m_rows - limit,
            BM_USERDEFINED, m_cols - 1, 1, BM_USERDEFINED);
          Matrix2.SetupBlockMatrix(limit, m_rows - limit,
            BM_USERDEFINED, m_cols - 1, 1, BM_USERDEFINED);
        }

        CGaussMatrix Matrix3 =
          Matrix.GetBlockMatrix(0, 0) * Matrix2.GetBlockMatrix(0, 1);

        cout << (Matrix3 == Matrix.GetBlockMatrix(0, 1)
          ? "v pořádku" : "špatně") << endl;
      }
    }
  } catch (CMatrixError &Error) {
    cout << "Chyba matice #" << Error.ID() << ": " << Error.what()
      << endl;
  } catch (exception &Error) {
    cout << "Jiná chyba: " << Error.what() << endl;
  }
}
```

Program si nejem umí vytvořit matici z náhodných čísel (pro jednoduchost jde jen o čísla od nuly do jedné, aby je bylo možné v případě potřeby vypsat bez exponentů), ale lze použít i soubor[5]. Matice nemusí mít ideální rozměr (n řádků, $n+1$ sloupců), potom ale soustava nemá

[5]Ten je textový a na prvním řádku má dvě čísla oddělená čárkou, udávající rozměr matice. Potom následují na dalších řádcích jednotlivá čísla matice oddělená mezerami.

řešení žádné, nebo jich má naopak nekonečně mnoho[6].

Po (alespoň částečném) vyřešení soustavy program vypíše výsledek. Pokud bylo řešení nalezeno, dosadí jej do původní matice a ověří výsledek.

Toto ověření také stojí za zmínku. Protože jde o maticovou operaci, je k tomu použit operátor == ze pro třídu `CMatrix`. Protože kvůli nepřesnostem při výpočtu (ty jsou dány procesorem) není možné obvykle použít prosté porovnání, je potřeba použít jiný způsob:

```
int operator == (const CMatrix& left, const CMatrix& right)
{
  int _m, _n;
  int _m2, _n2;
  int i, j;

  left.GetSize(_m, _n);
  left.GetSize(_m2, _n2);

  /* incompatible matrices? */
  if ((_m != _m2) || (_n != _n2))
    throw CMatrixError(M_BAD_SIZE);

  for (i = 0; i < _m; i++)
    for (j = 0; j < _n; j++)
    {
      if (fabs(left.GetField(i, j) - right.GetField(i, j)) >=
        lf_eps2 * (lf_eps2 + right.GetField(i, j)))
        return 0;
    }
  return 1;
}
```

Proměnná `lf_eps2` určuje vhodně zvolené malé číslo (např. 10^{-10}). Mohlo by být i menší, ale čím je matice větší, tím více průchodů je k eliminaci potřeba a nepřesnost výpočtu roste. Pokud je třeba vypočítat matici 50×51, jde o dostatečnou hodnotu, ale pro rozměr 300×301 by už měla být vyšší.

Teď už zbývá jen doplnit výstup z programu (parametr `-rand 4`):

```
Náhodná matice 4x5
(    0.781    0.372    0.459    0.374    0.790 )
(    0.449    0.077    0.117    0.777    0.718 )
(    0.848    0.097    0.237    0.016    0.353 )
(    0.195    0.939    0.660    0.786    0.721 ) 4x5
Počítá se...
Výsledek:
(    1.000    0.000    0.000    0.000   -0.253 )
(    0.000    1.000    0.000    0.000   -2.101 )
(   -0.000   -0.000    1.000    0.000    3.205 )
(   -0.000    0.000   -0.000    1.000    0.799 ) 4x5
Ověření: v pořádku
```

[6]Byla by až příliš velká náhoda, že by se náhodně vytvořily dva řádky, které by byly stejné, nebo by alespoň byly násobky. Nicméně program počítá i s tímto.