Gaussova eliminační metoda I.

1 Rozšířená matice soustavy

Gaussova eliminační metoda (GEM) je numerická metoda pro řešení soustav N lineárních rovnic o N neznámých. Pro naši implementaci bude soustava reprezentována pomocí rozšířené matice soustavy, kde je původně čtvercová matice *koeficientů* o rozměrech N×N rozšířená o sloupec *absolutních členů*. Vzniká tím matice o rozměrech N×(N+1).

1.1 Formát rozšířené matice

Matici budeme načítat z textového souboru ve formátu, který jsme si zavedli v předchozím cvičení:

```
3 5
 1.5  3.2   -.1   6.4 -2
-5    2.4  10     1.4  4.1
 1.5 -7.2   2   -64.1  7
```

- Na prvním řádku budou rozměry v pořadí *počet řádků*, *počet sloupců*.
- Poté následují prvky matice jejichž počet odpovídá rozměrům.
- Oddělovačem jsou bílé znaky (mezery, tabulátory, konce řádků). Na formátování nezáleží, tj. není potřeba počítat řádky nebo přebytečné mezery.
- Rozměry jsou celá kladná čísla, prvky matice jsou desetinná čísla.

Pro reprezentaci matice v programu použijeme typ Tmatice zavedený v minulém cvičení.

Poznámka

Tato ukázková matice pochopitelně není řádnou maticí soustavy – nemá správné rozměry. Při pokusu o načtení této matice by to měl náš program rozpoznat!

Tip

Pro ladění je možné proměnnou typu Tmatice inicializovat přímo ve zdrojovém kódu programu, aniž by bylo třeba ji načítat ze souboru.

Do funkcí je ji ale třeba předávat odkazem pomocí operátoru &. Při použití dynamické alokace použití toto odpadá.

```
// výroba proměnné
Tmatice m = {
  .radku = 3, .sloupcu = 5,
  .prvek = {
    {1.5, 3.2, -.1, 6.4, -2},
    {-5, 2.4, 10, 1.4, 4.1},
    {1.5, -7.2, 2, -64.1, 7},
  },
};
```

Tip pro pokročilé

Složku struktury prvek je ale potřeba mít deklarovánu se statickým rozměrem. U dynamické alokace není taková inicializace možná.

Chcete-li v programu pracovat s dynamicky alokovanou maticí, je díky použití struktury možné si nejprve připravit typ Tmatice obsahující pole s pevnými rozměry, inicializovat jej na začátku programu a použít jej pro ladění svých funkcí. Po odladění lze v typu Tmatice nahradit pole za ukazatel a přidat dynamickou alokaci. Díky použití struktury se již odladěné funkce pro práci s maticí nemusí modifikovat.

Úkol č. 1

Uprav funkci ctiM2 z minulého cvičení tak, aby přečetla ze souboru rozměry i prvky rozšířené matice soustavy.

Pokud matici vytváříš dynamicky, je vhodnější varianta, která vyrobí celou matici:

```
Tmatice *ctiRM(FILE *in, int *kod);
```

Funkce bude

- kontrolovat, zda rozměry dávají smysl a odpovídají rozšířené matici soustavy,
- vracet jako kód chyby nulu, když bude vše v pořádku,
- vracet jiné chybové kódy, když něco v pořádku nebude,
- vracet místo matice hodnotu NULL, když ji nejde vyrobit či správně načíst.
- Tip: Kódy chyb si pojmenuj a vyrob pro ně konstanty. Použij typ enum.

```
enum eChyby {EOK, EROZMER, ECTENI};
```

Úkol č. 2

Zkus ve funkci main vyrobit proměnnou typu Tmatice, kterou inicializuj rovnou v kódu bez použití funkce ctiRM. Zapiš to bez dívání do návodu na této stránce. Poté ji vytiskni pomocí funkce *tiskM2*, kterou máš hotovou z minulého cvičení.

2 Přímý chod GEM

Přímý chod GEM využívá řádkových ekvivalentních úprav matice soustavy k eliminaci prvků pod diagonálou. To znamená, že se snažíme dostat pod diagonálu nulové prvky, aniž by došlo ke změně řešení soustavy.

Tvar matice, kdy pod diagonálou jsou nulové prvky, na diagonále nejsou nulové a nad ní jsou libovolné nazýváme *horní trojúhelníková matice*.

2.1 Ekvivalentní úpravy

Použitelnými ekvivalentními úpravami jsou

- výměna řádků,
- násobení řádku konstantou a
- lineární kombinace dvou řádků.

Výměnu řádků využijeme při optimalizaci během tzv. pivotování (viz dále). Násobení řádků nebudeme dělat samostatně, protože je to zbytečně pracné a neefektivní. Budeme jej dělat zároveň s lineární kombinací řádků.

Úkol č. 3

Vytvoř funkci, která bezpečně vymění dva zadané řádky v matici. Bude-li zadán index řádku mimo matici nebo oba indexy řádků stejné, nic nedělej.

2.2 Pivotování

Protože při počítání s reálnými čísly na počítači vznikají zaokrouhlovací chyby, je potřeba numerické metody upravovat tak, aby tyto chyby zbytečně nenarůstaly. U GEM je takovou metodou tzv. *pivotování*.

Při ekvivalentních úpravách dochází k násobení koeficientů zlomkem, v němž se nachází diagonální prvek ve jmenovateli. Je-li tento zlomek větší než 1, dochází při výpočtu k tomu, že koeficienty ve spodních řádcích postupně exponenciálně narůstají.

Důsledkem je, že při řešení posledních rovnic se počítají podíly potenciálně obrovských čísel. Ačkoli matematicky jde o správný postup, v praxi dojde při počítání s velkými čísly nutně ke ztrátě přesnosti.

Pivotování tento jev omezuje tak, že vhodnou výměnou řádků dostane na pozici diagonálního prvku prvek s maximální absolutní hodnotou. Toto maximum hledá ve stejném sloupci jako je diagonální prvek od jeho pozice až dolů.

Úkol č. 4

Napiš a otestuj funkci *maxAbsPivot*, která pro zadaný řádek *r* najde v matici *m* pozici řádku pro výměnu. Hledá se maximální absolutní hodnota ve sloupci *r* od diagonálního prvku včetně směrem dolů.

Nalezený index řádku (ne hodnotu maximálního prvku) vracej jako funkční hodnotu.

```
int maxAbsPivot(Tmatice *m, int r);
```

2.3 Realizace přímého chodu

Algoritmus přímého chodu prochází postupně všechny řádky (index *r*). Nejprve pro něj provede pivotování spolu s případnou výměnou řádků. Poté je řádek *r* považován za referenční a už se s ním nesmí hýbat.

Zůstane-li po pivotování v diagonálním prvku hodnota 0, algoritmus končí, protože soustava nemá jedno řešení.

Pro každý řádek k pod referenčním řádkem se spočítá jejich vzájemný eliminační koeficient c v němž je diagonální prvek ve jmenovateli[1].

$$c = \frac{a_{kr}}{a_{rr}} \tag{1}$$

Po jeho spočtení se provede lineární kombinace řádků k a *r*, přičemž měnit se bude řádek k, nikoli r. Řádky stačí procházet od sloupcových indexů s začínajících hodnotou $r+1$, prochází se tedy prvky vpravo od sloupce *r*.

$$a_{ks} = c\,a_{rs} - a_{ks} \tag{2}$$

Prvek na pozici a_{kr} není třeba počítat. Má zde vyjít nula, proto ji sem rovnou stačí dosadit.

1 Šlo by to počítat i jinak, ale pak by pivotování mohlo ztratit svůj smysl a naše úsilí s ním by přišlo vniveč.

Úkol č. 5

Napiš funkci jeHorni, která otestuje, zda je zadaná matice ve tvaru horní trojúhelníkové matice. Takovou funkci lze využít pro otestování, zda se lze vyhnout přímému chodu a pokračovat rovnou zpětným chodem.

Z funkce vracej logickou funkční hodnotu typu bool. Použij hlavičkový soubor stdbool.h.

Funkci na vhodných datech otestuj.

```
bool jeHorni(Tmatice *m);
```

Úkol č. 6

Pomocí dříve vytvořených funkcí napiš funkci *gemPrimy* realizující přímý chod Gaussovy eliminační metody podle vztahů (1) a (2).

Nevypisuj v ní žádná chybová hlášení. Vracej místo toho číselnou funkční hodnotu s kódem chyby. Skončí-li běh bez chyb, vrať funkční hodnotu 0. V parametru m bude příslušně upravená rozšířená matice ekvivalentní soustavy.

```
int gemPrimy(Tmatice *m);
```