Dokumentace konzolové aplikace Maticová kalkulačka

Ondřej Kříž

Obsah

Anotace:

Cílem softwaru je poskytnou uživateli snadno ovladatelnou maticovou kalkulačku konzolového charakteru, usnadňující elementární práci s maticemi. Program nabízí interaktivní prostředí s vysvětlujícími tisky a podporuje celkem deset rozličných operací. Je zde také implementována základní práce se soubory, díky čemuž lze vstupní hodnoty matic aplikaci poskytnout v textové podobě (*.txt*) nebo následné výstupy jednotlivých operací v tomto formátu také ukládat.

Specifikace:

Kalkulačka je konzolová aplikace, které uživatel předloží data v podobě *reálných čísel* a v závislosti na zvolené operaci se provede výpočet:

1) maticové sčítání,
2) maticové odčítání,
3) násobení matice skalárem,
4) maticové násobení,
5) maticová transpozice,
6) převod matice na REF,
7) převod matice na RREF,
8) určení inverzní matice,
9) určení hodnosti matice,
10) určení determinantu.

Rozhraní aplikace nabízí dva rozličné módy práce s daty. Uživatel může data zadat v podobě textového formátu *.txt* (specifikace formátu dále v sekci Načtení dat ze souboru) nebo přímo prostřednictvím konzolového rozhraní. Po provedení výpočtu je uživateli nabídnuto, uložení dat do souboru.

Algoritmy

Jádro celé kalkulačky jsou algoritmy jednotlivých operací, které jsou převážně rozčleněny do samostatných tříd a modulů v balíčku *operations*. Většina algoritmů přímo kopírují obecně známé definice výpočtů, které jsou k nalezení např. v díle: „M. Hladík, Lineární algebra (nejen) pro informatiky, MatfyzPress, 2019".

Algoritmus pro výpočet REF:

Algoritmus je k nalezení v *operations.matrix_ref*. Mimo samotného výpočtu REF se v něm provádějí další dodatečné operace vztahující se k objektu MatrixREF:

Pseudokód:

Dokud *pivot_j_position* a *pivot_i_position* jsou menší než rozměry matice:

Pokud je pivot[*pivot_i_position*][*pivot_j_position*] nulový:

Najdi řádek s nenulovou hodnotou(pivotem) nejvíc nalevo;

Pokud takový řádek není:

matice je odstupňovaná –> vrať matici;

Jinak nastav *pivot_j_position* na novou pozici, která je nejvíc vlevo a prohoď aktualní řádek o – *pivot_i_position* – s nově nalezeným;

Změň znaménko determinantu; (*self.determinant_sign**(-1))

Vynuluj všechny prvky pod vedoucím pivotem
(přičtením šikovného násobku[1] aktuálního řádku ke všem podním);
(*__calculate_ref_for_pivot_row(pivot_i_position, pivot_j_position)*)

Přidej aktuální pivot do seznamu všech pivotů; (self.pivot_positions)

Posuň se na další pivot; (*pivot_i_position++, pivot_j_position++*)

[1] sikovny_nasobek = (-1/zpracovavany_pivot) * (pivot_ktery_nuluji)

Výpočet RREF:

Algoritmus využívá toho, že operace MatrixREF má uložené pozice pivotů. Na začátku je vstupní matice pomocí operace MatrixREF převedena na REF tvar. Algoritmus je v některých ohledech velmi podobný REF – jedná se o téměř totožný algoritmus, pouze průchod je prováděn zpětně(a jen pro bázické sloupce).

Pseudokód:

```
Pokud je pouze jeden pivot:
    Normalizuj pouze tento řádek s pivotem (resp. vyděl řádek 1/hodnota_pivotu);
    vrať matici;
Pokud nejsou pivoty: vrať matici;
Projdi po zpátku všechny pozice pivotů:
(Procházím zpětně, protože REF je kvůli dopřednému běhu tak vygeneruje.)
    Projdi všechny řádky od pivota k prvnímu řádku matice:
        sikovny_nasobek = (-1/zpracovavany_pivot) * pivot_ktery_nuluji;
        Projdi každý prvek od pozice pivota v sloupci do posledního sloupce:
            prvek_v_radku += prvek_v_radku_pivota * sikovny_nasobek;
            prvek_v_radku_pivota /= zpracovavany_pivot;
            (Normalizace řádku)
Normalizuj první řádek;
```

Výpočet Determinantu:

Tento algoritmus je opět závislý na operaci MatrixREF. Následující pseudokód popisuje metodu *get_determinant(mx_ref, dimensions = -1, determinant_sign = 0)* z *oprations*.MatrixDeterminant. A již očekává v parametru *mx_ref*, matici v REF.

Pseudokód:

```
determinant = 1

Pokud matice není čtvercová: vrať;

Projdi všechny prvky na hlavní diagonále:

        determinant *= prvek_na_diagonale;

determinant *= determinant_sign;
```

Alternativní přístup:

Další možnost provedení výpočtu determinantu, která se nabízela bylo např. pomocí tzv. *Laplacova rozvoje*. Nicméně ten je především pro velké rozměry matic velmi nepraktický, neboť jeho časová složitost je O(n!). Další možností by bylo zvolit metodu založenou na LU rozkladu, kde časová složitost je kubická. Vzhledem k tomu, že kalkulačka již uměla výpočet REF, tak jsem zvolil právě tento algoritmus, který díky tomu běží rovněž v čase $O(n^3)$.

Výpočet Inverzní matice:

Pro výpočet inverzní matice byly skloubeny dohromady výše uvedené operace. Výpočet kopíruje stejný postup, jako při počítání rukou na papíře, a sice pomocí rozšíření o jednotkovou matici.

Blokový popis:

1. Pokud je vstupní matice čtvercová rozšíří se o jednotkovou matici;
2. Určí se REF;
3. Vypočítá se determinant;
4. Pokud determinant matice “na pozicích” jednotkové matice není nulový -> RREF;
5. Dojde k oříznutí části, kde byla původní matice(nyní je jednotková).

Program:

Celý program je členěn do modulů a balíčků Python, kde každý soubor (tedy kromě *matrix_calculator.py* a *constants.py*) je vyhrazen pro právě jednu třídu. Název třídy je poté shodný až na podtržítko a velká písmena s názvem modulu(např. třída MatrixREF je v modulu *matrix_ref.py*).

Vstupním bodem celého programu je modul *matrix_calculator.py*, obsahující implementaci a volání globální funkce *main_loop()* zodpovědné za běh celého programu. Zde je programu rozčleněn na čtyři po sobě jdoucí sekvence:

1) uživatelské rozhraní – volba operace,
2) uživatelské rozhraní – způsob práce s daty,
3) průběh zvolené operace pomocí třídy *OperationExecution*,
4) volba uložení výstupních dat do souboru.

Zjednodušeně celý tok dat vypadá následovně: Metoda *main_loop()* získá informaci **o volbě operace** a **volbě způsobu zadání dat** (v podobě podpisů na metody v *OperationExecution*), které následně předá pomocí konstruktoru nové instanci *OperationExecution*. Poté se na tomto objektu zavolá metoda *execute()*, která spustí právě tu příslušnou metodu operace, jejíž podpis byl předána objektu při tvorbě v konsturktoru.

Nyní je řízení celého průběhu operace již pouze v rukách konkrétní metody. Detajlnější popis je v sekci třídy zde: operation_execution.py – OperationExecution.

Poté co je operace dokončena, je starost řízení poslána zpět do *main_loop()*, kde v závislosti na tom, jestli byl výsledek operace validní(podle *OperationExecution.operation_result* zdali není None) dojde ke spuštění 4. bodu – volby uložení výstupních dat do souboru.

Detaily k názvům:

- V kódu často bývá proměnná matice označena zkráceně jako: *mx*.
- Při průchodu dat *for cyklem* je pro označení řádku matice často zaveden symbol: *i* a pro průchod skrze sloupce symbol *j*.

marix_calculator.py

Vstupní bod programu, funkce *main_loop()*.

matrix.py – Matrix

Hlavní třída reprezentující matici.

Atributy:

- **m**: int – počet řádků matice
- **n**: int – počet sloupců matice
- **data**: list – 2D list obsahující data matice. První index pole udává řádek matice.(např. data[0][2] – hodnota v prvním řádku třetím sloupci).

helpers.py – Helpers

Obsahuje *statické funkce*, které se *globálně* používají v různých částech aplikace.

Funkce:

- **invalid_dims_addition(matrix: Matrix, second_mx_dims: tuple) -> bool:** Kontroluje jestli jsou rozměry matice *matrix* shodné s poskytnutými rozměry *second_mx_dims*, kde první položka *n-tice* udává *počet řádků* a druhá *počet sloupců*.
- **invalid_dims_multiplication(matrix: Matrix, second_mx_dims: tuple) -> bool:** Kontrola zdali je možné matice násobi t(V pořadí *matrix* * *second_mx_dims*).
- **correnct_file_name_exstention(file_name: str, valid_exstention: str="txt") ->str:** Oprava názvu souboru *file_name*, pokud nekončí validní příponou – *valid_exstention*.

constants.py

Obsahuje *globální konstanty* programu:

- **FRACTION_OUTPUT:** **bool** – Řídí podobu výstupních dat, zdali se mají tisknout jako desetinné číslo nebo zlomek.
- **OUTPUT_PRECISION:** **int** – Nastavení zaokrouhlení výstupních hodnot (pouze v tisku, výpočet je prováděn v plné přesnosti typu Python **float**).
- **INPUT_FILES_DIR:** **string** – Název adresáře pro vstupní soubory (relativní k adresáři projektu).
- **OUTPUT_FILES_DIR:** **string** – Název výstupního adresáře pro ukládání soubory (relativní k adresáři projektu).

main_user_interface.py – MainUserInterface

Stará se o vykreslení a input “hlavních” uživatelských nabídek(jedná se o blokující funkce, které běží dokud uživatel nezadá validní vstup).

- **operation_selection() -> void:** Tisk nabídky volby operace a čtení volby ze standardního vstupu.
- **data_load_selection() -> void:** Nabídka pro zvolení práce s daty.
- **data_store_selection() -> void**: Nabídky pro uložení dat z výpočtu.

operation_execution.py – OperationExecution

Třída řídící průběh jednotlivých operací.

Atributy:

- **__operation** – odkaz na vnitřní metodu třídy zvolené operace. Inicializace pomocí konstruktoru parametrem – *mx_operation_method*
- **__read_mx_data** – odkaz na vlastní metodu způsobu načtení dat. Parametr v konstruktoru – *load_data_method*
- **input_reader** – instance třídy InputReader
- **output_writer** – instance třídy OutputWriter
- **current_operation** – private field zaznamenávající poslední proběhlou operaci
- **operation_result** – výsledek poslední operace
- **dims_check** – odkaz na funkce která slouží pro kontrolu vstupních dat při čtení(např. při operaci sčítání matic se po přečtení první matice nastaví na **Helpers.invalid_dims_addition** pro kontrolu při zadáván druhé matice)

Metody:

- **execute() -> void:** Hlavní metoda, kterou se spouští vykonání některé z operací.
- **write_result_to_file(file_name: str) -> void:** Zápis dat do souboru. Pokud je *file_name* prázdný, tak nastane běhová chyba s tiskem: "Název souboru je prázdný!". Používá atribut *self.operation_result*. Pokud je tento atribut None, tak nastane běhová chyba: "Výsledek operace je neplatný!"
- **__get_mx_dims_console() -> tuple:** Spustí čtení rozměrů matice z konzole.
- **load_data_from_file() -> Matrix:** Stará se o uživatelské rozhraní čtení dat ze souboru. Spouští samotné čtení *InputReader.read_matrix_data_from_file()*. Pokud jsou rozměry druhé matice neplatné, zavolá rekurzivně samu sebe.
- **load_data_from_console() -> Matrix:** Uživatelské rozhraní získání hodnot z konzole.

Metody uvedené níže se starají o hlavní uživatelské rozhraní a průběhu konkrétních operací. Metody mají velmi podobnou strukturu, která lze rámcově vystihnout:

1) vytiskne název operace,
2) načtení dat matic(e) a zbylých hodnot potřebných pro výpočet (včetně případného nastavení kontrolní funkce *self.dims_check*),
3) výpis počítané formule,
4) vyhodnocení výpočtu,
5) tisk výsledku.

- **mx_add() -> void:**
- **mx_sub() -> void:**
- **mx_scalar_multiply() -> void:**
- **mx_multiply() -> void:**
- **mx_transpose() -> void:**
- **mx_ref() -> void:**
- **mx_rref() -> void:**
- **mx_inverse() -> void:**
- **mx_rank() -> void:**
- **mx_determinant() -> void:**

input_output.input_reader – InputReader

Metody:

- **read_matrix_dimensions(validity_condition: lambda) ->list**: Blokuje program dokud uživatel nezvolí validní vstup. Funkce *validity_condition* poté slouží ke kontrole, zdali jsou rozměry pro danou operaci platné.
- **__try_parse_matrix_dims(dim_string: str)->list:** Pokusí se parsovat dimenze matice zadané uživatelem. V případě neúspěchu vrátí False.
- **read_matrix_user_input(mx: Matrix)-> void**: Zablokuje program a bude číst po řádcích hodnoty matice zadané uživatelem(dokud úspěšně nebudou vložena všechna data dle rozměrů matice). Při neplatné volbě vrátí chybovou hlášku, vytiskne zatím úspěšně vložená data a pokračuje v čtení.
- **__try_parse_matrix_row_input(input_data: str) -> list:** Pokusí se parsovat řádek matice zadaný uživatelem v konzoli. Při neúspěchu vrátí False.
- **read_scalar() -> float**: Blokující funkce – čtení skaláru vloženého uživatelem.
- **read_matrix_data_from_file(file_name: str) -> list:** Pokusí se přečíst soubor zadaný uživatelem. V případě neúspěchu vypíše chybovou hlášku a vrátí False. Jinak vrátí list dat.

input_output.output_writer – OutputWriter

Stará se o ukládání dat do souboru po provedení operace.

Metody:

- **store_output(data_to_print, file_name: str) -> void:** Public metoda zprostředkovávající zápis dat.
- **__write_data_to_file(data_to_print, path_to_file: str)-> void:** Private metoda sloužící k samotnému zápisu. Pokud jsou *data_to_print* typu Matrix získá jejich string reprezentaci pomocí, jinak se je pokusí převést na string. Data uloží na dle cesty v *path_to_file*. Pokud dojde k nějaké chybě vypíše pouze chybovou hlášku.

matrix_print.matrix_print_object.py – MatrixPrintObject

Pomocný objekt, který na rozdíl od objektu Matrix uchovává data zformátovaná ve formátu string a připravená pro tisk do konzole.

Atributy:

- **mx_m**: int – počet řádků matice
- **mx_n**: int – počet sloupců matice
- **round_to**: int – počet platných desetinných míst
- **use_frac**: bool – převést na zlomky
- **row_filter**: lambda – pokud všechny hodnoty v řádku splňují nějakou podmínku, tak řádek vynechat (defaultně pokud jsou všechny hodnoty None)
- **data**: list – 2D list stringů obsahující formátovaná data

Metody:

- **__convert_data_to_string(matrix_data:list) -> list:** Hlavní funkce konverze a formátování dat. Volá se v konstruktoru při vytvoření objektu.

- **__format_row(row:list)->list:** Formátuje řádek matice. Kontrola jestli je splněna podmínka *self.row_filter*. Pokud ne, funkce vrátí False.
- **__format_cell_value(matrix_data:float) -> str:** Formátování buňky – Zaokrouhlení a případný převod na zlomek.
- **__dec_to_frac(value:float, denominator_limit:int = 100):** Pomocná funkce pro převod desetinných čísel na zlomek.
  denominator_limit – určuje maximální možnou hodnotu ve jmenovateli zlomku.

matrix_print.column_padding.py – ColumnPadding

Pomocná třída která určí padding pro jednotlivé sloupce matice při tisku.

Metody:

- **get_padding_for_columns() -> list:** Vrací list nejdelší(podle počtu symbolů) hodnoty v každém sloupci.
- **__get_longest_val_in_row(column_index:int)->int:** Pomocná funkce pro nalezení nejdelší hodnoty v sloupci matice. (Pomocí lineárního vyhledávání)

matrix_print.Matrix_console_printer.py – MatrixConsolePrinter

Třída obsahující statické metody určené k tisku dat matic do konzole.

Metody:

- **print_default(matrix: Matrix)->void:** Vytiskne matici formátovanou po řádcích do konzole s krajním ohraničením.
- **print_simple(matrix:Matrix, get:bool = False)->str/void:** Prostý tisk po řádcích do konzole. Volitelný parametr *get* umožňuje místo tisku data vrátit jako string.
- **matrix_to_bracket_string(matrix: Matrix)->void:** Vytiskne matici jako jeden řetězec v složených závorkách. (V kódu se nepoužívá, pouze jako další možnost výstupních dat např. pro nějaký matematický software apod.)

elementary_operations.py – ElementaryOperations:

Elementární operace s maticemi.

- **check_if_matrix_dims_equal(mx1: Matrix, mx2: Matrix)->bool:**
- **exchange_rows(mx, row1: int, row2: int)->void:**
- **multiply_row_by_scalar(mx: Matrix, row: int, scalar: float)->void:**
- **add_two_rows(mx: Matrix, row1: int, row2: int)->void:** Pokud jsou souřadnice mimo rozsah, tak vyhodí běhovou chybu.
- **find_first_most_left_value(mx: Matrix, start_row: int, start_col: int)->tuple:** Pokud jsou hodnoty mimo rozsah vyhodí běhovou chybu. Pokud nenajde vrátí (0, 0).
- **expand_for_identity_matrix(mx: Matrix)->Matrix:**

operations:

Balíček *operations* je určený především pro třídy samotných algoritmů operací. Všechny třídy mají jistým způsobem předepsanou následující strukturu(ale není to jakkoliv pevně definováno):

- V konstruktoru se předává objekt matice a ostatní data potřebná pro operaci.
- Obsahuje *public metodu* začínající prefixem **calculate_**, vracející výsledek operace.

Moduly a Třídy:

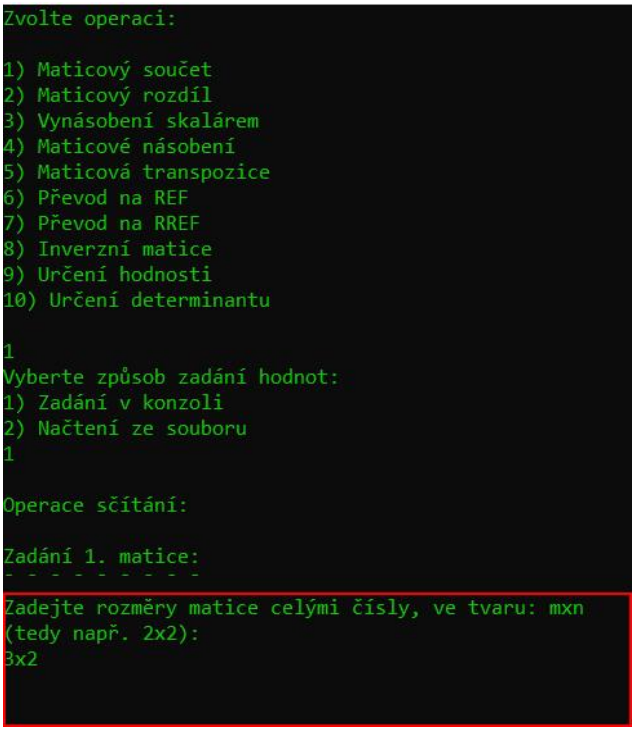
- **matrix_addition.py – MatrixAddition:**
  Používá se zároveň jako operace pro odčítání – volitelný parametr *substract:*
  calculate_sum(substract: bool = False)
- **matrix_determinant.py – MatrixDeterminant**
- **matrix_inversion.py – MatrixInversion**
- **matrix_multiplication.py – MatrixMultiplication**
- **matrix_ref.py – MatrixREF**
- **matrix_rref.py –MatrixRREF**
- **matrix_scalar_multiplication.py – MatrixScalarMultiplication**
- **matrix_transposition.py – MatrixTransposition**

Vstupní data

Poté co si uživatel zvolí jednu z deseti operací v hlavní nabídce, zobrazí se nabídka způsobu práce s daty. Pro čistou práci v konzoli stačí zvolit 1) v případě práci se soubory možnost 2).

Zadání v konzoli:

Po zvolení této možnosti se spustí vybraná operace a vykreslí se text vyzývající uživatele k zadání první matice.



Rozměry matic:

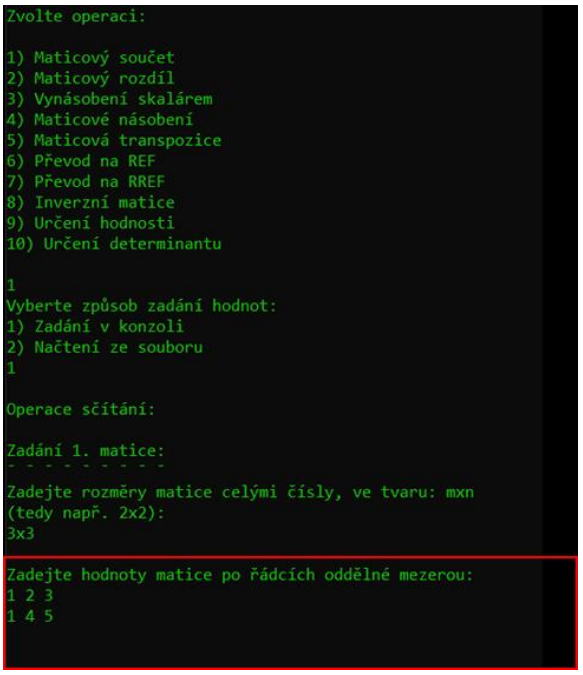
Při volbě zadání v konzoli je nutné při vkládání dat matice nejprve specifikovat její rozměry. Ty se zadávají ve specifickém formátu. Syntax začíná zadáním prvního celého čísla určující počet řádků (obecně značeno *m*). Poté následuje symbol *x* (lze psát i velkým písmenem), značící kartézský součin, za nímž poté následuje celočíselná hodnota počtu sloupců. Zápis tedy poté vypadá(bez závorek): *(počet řádků)x(počet sloupců)*.

Příklad: 3x2

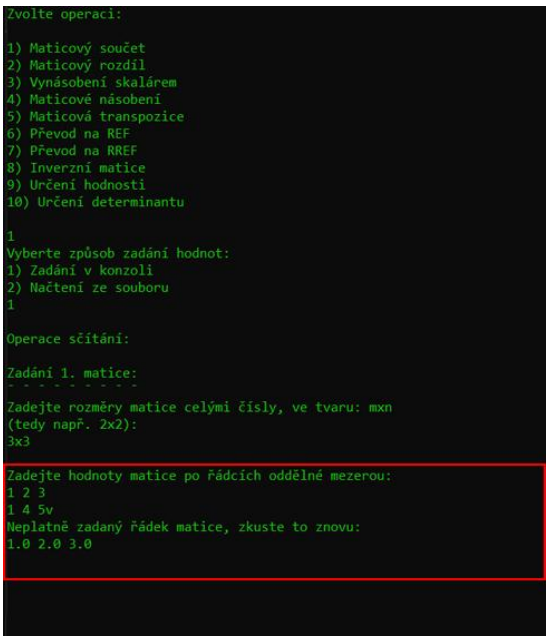
Mezi jednotlivé znaky je také možné vkládat libovolný počet mezer. Na obrázku výše je poté v červeném rámečku vyznačený tisk této operace. Pokud jsou rozměry neplatné, aplikace vybídne k opětovnému zadání.

Vložení dat:

Poté co uživatel vloží rozměry matice se zobrazí tisk vybízející k vložení dat po řádcích jak je vidět níže na obrázku. Jednotlivé hodnoty se oddělují na řádku mezerou. Ty mohou nabývat podoby celých nebo desetinných čísel. Jako delimiter desetinného čísla je použita tečka(tedy např. 1.5 nikoli 1,5!). Uživatel musí vložit na řádek přesně takový počet hodnot, jaký specifikoval v rozměrech matice.



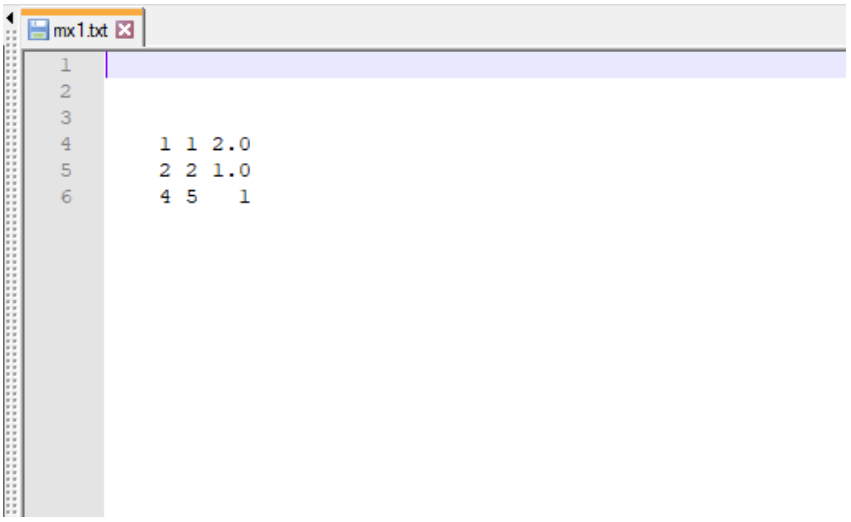
Pokud z nějakého důvodu nejsou data řádku akceptována, aplikace vytiskne větu vybízející k opětovnému pokusu vložení dat řádku a zároveň pod sebe vytiskne všechny dosavadně úspěšně vložené řádky. Taková situace je vidět na obrázku níže.



Načtení dat ze souboru

V druhém módu se uživateli na místo zadávání rozměrů matice vypíše rovnou text vybízející k zadání názvu souboru. Formát vstupního souboru musí být .txt, přičemž samotný název je možné zadat bez přípony. Veškeré vstupní soubory jsou poté defaultně v adresáři *matrix_files*. Výchozí adresář lze případně změnit v souboru *constants.py*: *INPUT_FILES_DIR*.

Veškeré hodnoty v souboru musejí být zadány jako *reálná*(nebo *celá*) čísla, oddělená jednou nebo více mezerami(nebo i *tabulátorem*). Příklad validního vstupu:



Program tento vstup přečte jako:

1.0 1.0 2.0
2.0 2.0 1.0
4.0 5.0 1.0

Při zadání neplatných hodnot(např. nenumerické znaky), nebo při rozličných délkách řádků dojde k chybě a aplikace vybídne uživatele, aby zkusil opakovat operaci zadání souboru.

Také je nutné poznamenat, že načítání dat ze souboru se vztahuje pouze na matice. Např. skalární násobek se zadává v konzoli jako jedna číselná hodnota.

Zpracování výstupu operací

Po provedení výpočtu je výsledek operace vytištěn v konzoli. Uživatel je poté u každé operace tázán, zdali chce data uložit do textového souboru.V případě, že nechce, tak stačí nechat input prázdný a stisknout Enter. V opačném případě, uživatel zadá název souboru(stačí bez přípony *.txt*). Soubor se zadaným jménem obsahující výsledek operace se následně uloží do výstupní složky definované v *constants.py: OUTPUT_FILES_DIR*.

Je nutné, aby adresář zadaný v constants byl již předem vytvořený!

Správná interpretace dat

Protože se kalkulačka „snaží“ pracovat v oboru *reálných čísel*, tak při výpočtech je možné občas narazit na zaokrouhlovací chyby. Je tedy podstatné mít tento faktor při výpočtech na paměti a správně volit OUTPUT_PRECISION konstantu. Se zaokrouhlovacími chybami se váže i fakt, že pokud bude matice tzv. *špatně podmíněná*, tak mohou vznikat velké zaokrouhlovací chyby.

Závěr

S finálním výsledkem kalkulačky jsem víceméně spokojený. Kalkulačka zvládá většinu základních operací, které si myslím, že dokáží značně ulehčit manuální zdlouhavé výpočty na papíře. Ukázalo se, že rozsah této práce byl trošičku větší, než jsem původně zamýšlel, ale zase na druhou stranu se nejednalo o nic zvlášť těžkého.

Původně stanovené cíle v zadání jsem sice splnil, ale i tak je zde několik věcí co by šlo určitě vylepšit. Jednou z nich je zcela bez pochyby nějaké rozšíření, které by zajišťovalo lepší stabilitu při výpočtu *špatně podmíněných matic* nebo které by řešilo *řídké soustavy*. K tomuto účelu jsem mohl například použít *parciální pivotizací,* která jak jsem až po implementaci REF zjistil má obecně lepší výsledky ve stabilitě. Samotná architektura také není v některých místech ještě ideální a chtělo by to provést jistý refactoring.

Nicméně z běžného uživatelského hlediska je kalkulačka použitelná, dle mého názoru, v rámci možností konzolového rozhraní i relativně přívětivá a pro většinu běžných výpočtů dostačující.