Univerzita Pardubice
Dopravní fakulta Jana Pernera

Softwarová konverze dat parametrů železničních tratí pro účely trakčních výpočtů v prostředí MATLAB

Tomáš Hering

Bakalářská práce
2015

PROHLÁŠENÍ

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.
Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/200 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a tím, že pokud dojde k užití této práce mnou nebo poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladu, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 28.5.2015.

Tomáš Hering

ANOTACE

Práce je zaměřena na tvorbu programu, jenž načítá výchozí data o směrových a sklonových poměrech, polohách zastávek a traťových rychlostech vybrané trasy. Dále vybírá pouze užitečné údaje a seřadí je dle zvoleného směru do navazujícího celku. Ke všem údajům je přiřazena relativní kilometrická poloha vztažená k začátku trasy. Směrové poměry jsou přepočteny na redukovaný jízdní odpor, přičteny ke sklonům a následně sloučeny dle pravidel pro trakční výpočty.

KLÍČOVÁ SLOVA

MATLAB, železnice, třídění, software

TITLE

Software for data conversion of parameters of railway tracks for the traction calculations in MATLAB environment.

ANNOTATION

The thesis is focused on the development of a program, which loads the input data of profile, alignment, halts and speed limits of a chosen route. The data are filtered for the relevant values and sorted in correspondence to the desired direction. A relative stationing position based on the distance from the beginning of the route is computed for each bit. The alignment data are reduced and added to those of profile resulting in a merged resistence information following the rules of traction calculations.

KEYWORDS

MATLAB, railways, sorting, software

Univerzita Pardubice
Dopravní fakulta Jana Pernera
Akademický rok: **2013/2014**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Tomáš Hering**
Osobní číslo:	**D11345**
Studijní program:	**B3709 Dopravní technologie a spoje**
Studijní obor:	**Dopravní infrastruktura: Elektrotechnická zařízení v dopravě**
Název tématu:	**Softwarová konverze dat parametrů železničních tratí pro trakční výpočty v prostředí MATLAB**
Zadávající katedra:	**Katedra elektrotechniky, elektroniky a zabezpečovací techniky v dopravě**

Zásady pro vypracování:

Navrhněte algoritmy pro načítání parametrů reálných železničních tratí (sklonů, oblouků, rychlostí, zastávek) pro trakční výpočty do matic programu MATLAB.
Zapracujte správné staničení jednotlivých tratí a relativní kilometrickou polohu pro definované trasy.
Úvod práce by měl obsahovat teoretický rozbor vztahů používaných výpočty.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování bakalářské práce: **tištěná**

Seznam odborné literatury:

Drábek J.: Dynamika a energetika elektrické trakce. VŠDS Žilina 1987
Jansa F.: Trakční mechanika a energetika kolejové dopravy. Dopravní nakladatelství Praha 1959
Zaplatílek K., Doňar B.: MATLAB pro začátečníky. BEN - technická literatura 2003

Vedoucí bakalářské práce: **Ing. Ladislav Mlynařík, Ph.D.**
Katedra elektrotechniky, elektroniky a zabezpečovací techniky v dopravě

Datum zadání bakalářské práce: **29. listopadu 2013**
Termín odevzdání bakalářské práce: **30. května 2014**

prof. Ing. Bohumil Culek, CSc.
děkan

L.S.

doc. Ing. Radovan Doleček, Ph.D.
vedoucí katedry

V Pardubicích dne 21. února 2014

Obsah

Úvod

Práce je koncipována jako popis základních pravidel trakčních výpočtů a dále mnou vytvořeného programu v aplikaci MATLAB společnosti Mathworks.

Motivací pro vznik práce je začlenění do projektu, jehož cílem je analýza vhodnosti nasazení tzv. hybridních vozidel[1] na vybrané trasy v síti SŽDC. Jestliže má být tato analýza relevantní, je nutné vyjít ze skutečných a přesných dat o dopravní cestě. Ta jsou sice k dispozici, nikoliv však ve vhodné formě. Jejich ruční zadávání do simulačního programu by bylo pracné a zdlouhavé, je tudíž žádoucí vytvořit program pro pohodlné a pokud možno automatizované načtení dat z různých zdrojů s unifikovaným výstupním formátem.

Trasy byly vybírány s ohledem na sklonové poměry, přítomnost trakčního vedení, dosavadního objednávaného objemu dopravy a nasazených vozidel. Některé z nich nejsou dnes obsluhovány přímým spojem například z důvodu nutnosti přepřahu hnacího vozidla. Celkem jich je 20 a jejich kompletní seznam je uveden v příloze A.

Využití prostředí MATLAB je logickou volbou z několika důvodů. Výstupní data v něm budou dále zpracována, takže je nebude potřeba importovat odjinud, jako programátor můžu navíc využít obsáhlou knihovnu funkcí a algoritmů, jakož i podporu grafického výstupu. Celý kód je s výjimkou komentářů psán anglicky. Z programátorského hlediska je úloha řešena funkcionalistickým přístupem s agregováním proměnných do struktur.

[1] Vozidla využívající více zdrojů energie. Preferovaným typem v projektu je kombinace napájení z trakčního vedení a ze superkapacitoru nabíjeného proudem ze sběrače nebo rekuperací.

1. Názvosloví

Program je psán pokud možno dle konvencí jazyka C++[1]. MATLAB má však mnohá specifika, kterým jsem musel pojmenování přizpůsobit. Vybraná konzistentní pojmenování jsou uvedena v Tab. 1.1.

Tab. 1.1 Význam vybraných pojmenování proměnných a funkcí

Pojmenování	**Význam**
num-	počet
col-	sloupec
row-	řádek
is-	logický datový typ (může být i vektor)
eq-	totožný
cur-	současný (ve smyslu průchodu smyčkou)
fresh-	nově vytvořený
-ref	referenční (týkající se struktury sklonu)
-new	týkající se struktury přiřazované k referenci
-x	proměnná vytvořená ve vnořené funkci, jejíž název by bez znaku „x“ jinak překrýval proměnnou ve funkci hlavní
první písmeno velké	datový typ struktura
vše kapitálkami	konstanta

[1] I těchto konvencí je několik, vycházím ze zdroje [1].

2. Teorie trakčních výpočtů

Trakční výpočty slouží k analýze nároků na provozování hnacího vozidla na železniční dopravní cestě. Klíčovým grafickým znázorněním je tzv. tachogram. Ten ukazuje závislost okamžité rychlosti na čase nebo dráze. Ve svém důsledku např. pomáhají s výběrem nebo návrhem vozidla pro konkrétní trasu a zátěž, k výpočtu jízdních dob apod. Je zřejmé, že výsledek ovlivňují jak vlastnosti vozidla, tak trasy. Konkrétními parametry jsou:

- trakční charakteristika,
- brzdící procenta,
- jízdní odpory,
- traťová rychlost (platná pro typ vozidla),
- poloha zastávek (případně dalších dopravně význačných bodů vyžadujících pravidelnou změnu rychlosti),
- specifika tratě z hlediska provozu elektrické trakce (oddělovací úseky, úseky bez trakčního vedení).

Z hlediska této bakalářské práce jsou důležité výhradně parametry na straně dopravní cesty. Poslední z nich také není předmětem načítání z důvodů zmíněných v úvodu o motivaci projektu – nejde o analýzu tratě z hlediska provozování čistě elektrických vozidel, naopak hybridní vozidlo by úseky nezávislé trakce mohlo pojíždět s výkonem.

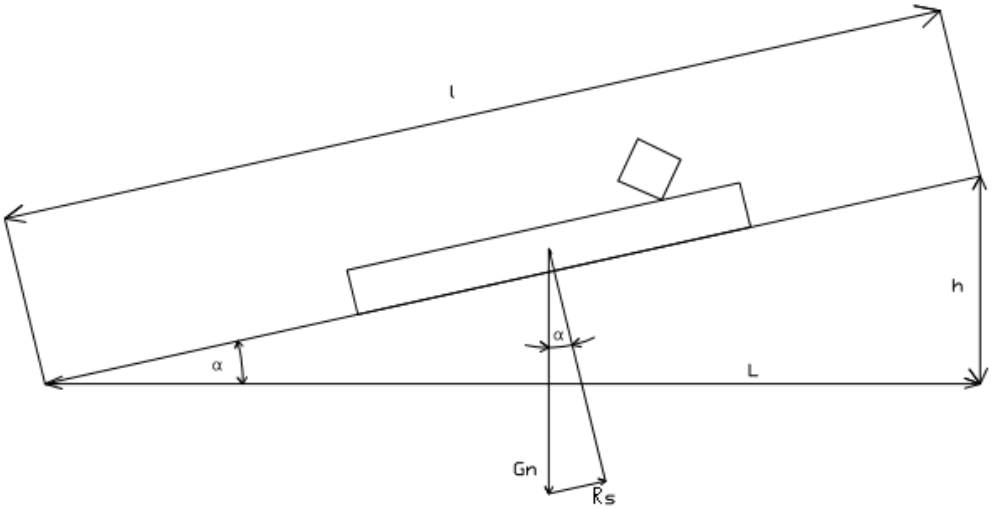
2.1. Traťové odpory

Odpory v zásadě dělíme mezi vlastní odpory vozidla a odpory traťové. Odpory vozidla z výše uvedených důvodů nejsou v mé práci podstatné. Základní traťové odpory, které se uplatňují v našich podmínkách jsou způsobeny jízdou:

- ve stoupání (nejvýznamnější vliv),
- v oblouku,
- v tunelu,
- ve větru.

S účinky větru samozřejmě nepočítáme, jelikož jeho výskyt a vlastnosti jsou ze své podstaty spíše stochastickým jevem.

2.1.1. Odpor ze stoupání



Obr. 2.1 Rozklad tíhy drážního vozidla ve sklonu Chyba! Nenalezen zdroj odkazů..

Tíha vozidla na sklonu je rozložena na dvě složky. Složku kolmou na rovinu koleje nemusíme uvažovat, protože vyvolá stejně velkou složku opačného směru a navzájem se vyruší. Složka R_s rovnoběžná s rovinou koleje působí v podélné ose vlaku. Při pohybu vlaku do stoupání působí jako odpor. V opačném případě se jedná o urychlující sílu. Její absolutní hodnotu můžeme vyjádřit následujícím vztahem:

$$R_s = 1000G \sin \propto$$

Protože je vhodné odpory vyjadřovat v poměrných jednotkách, můžeme rovnici podělit tíhou soupravy:

$$r_s = \frac{R_s}{G} = 1000 \sin \propto \left[\frac{N}{kN}\right]$$

Je-li zároveň sklon trati (s) vyjádřen v jednotkách promile, pak platí celá rovnice:

$$s = 1000\frac{h}{l} = r_s$$

můžeme říci, že měrný odpor ze stoupání (r_s) je číselně roven udávanému sklonu v promile. Matematickým vyjádřením odporu v negativním sklonu (při jízdě s kopce) je hodnota se záporným znaménkem. (Mlynařík, 2011)

2.1.2. Odpor z průjezdu obloukem

Při průjezdu kolejového vozidla obloukem vznikne nezanedbatelná odporová síla. Tato síla je závislá na mnoha faktorech, především pak na poloměru oblouku, rozvoru vozidla a podvozku, rozchodu koleje a velikosti převýšení. Pro jednoduchost jsou používány tzv. Röcklovy vzorce, respektující především poloměr projížděného oblouku. (Mlynařík, 2011)

Všechny načítané tratě mají standardní rozchod 1435 mm. Vztah pro ně v našich podmínkách nabývá podoby:

$$r_0 = \frac{650}{R - 55}\left[\frac{N}{kN}\right]$$

kde R tentokrát značí poloměr oblouku v metrech.

2.1.3. Odpor z jízdy v tunelu

Při průjezdu tunelem je čelem vlaku vytlačován sloupec vzduchu. Tím je zvýšen odpor jízdy vlaku. Aby bylo možné projíždět tunely konstantní tažnou silou, dochází v delších tunelech k redukci sklonu o 2‰ oproti okolním sklonovým poměrům. Z tohoto důvodu není nutné odpor z průjezdu tunelem v trakčních výpočtech uvažovat. (Mlynařík, 2011)

Předpis ČD – V7 uvádí, že by se v tunelu měla k redukovanému sklonu 2‰ přičíst, nicméně údaje o parametrech tunelů ani jejich kilometrické polohy nejsou v současné sadě vstupních dat. Pokud by s nimi v budoucnu mělo být počítáno, bude potřeba tuto funkcionalito naimplementovat.

2.2. Redukce odporu

Jelikož se parametry tratě mění poměrně často (moje empirická hodnota průměrného konstantního úseku je asi 200 m), byla v minulosti, kdy nebyla k dispozici výpočetní technika, přijata pravidla pro jejich redukci. Ta jsou používána dodnes, byť už je důvod jejich existence fakticky překonán.

Pravidla byla předepsána v předpise V7 takto (vybrané body):

- nesmí být slučován spád se stoupáním (může se však slučovat rovina se stoupáním nebo rovina se spádem) a nesmí se slučovat úseky s rozdílem sklonů větším 2,5 ‰.
- není-li rozdíl mezi největším a nejmenším sklonem větší než 1 ‰, lze sloučit jednotlivé úseky na libovolnou délku,
- činí-li rozdíl největšího a nejmenšího sklonu 1,1 až 2,5 ‰, lze sloučit jednotlivé úseky až na délku 3000 m,
- úseky kratší 100 m lze sloučit se sousedním úsekem i v případě, že rozdíl obou sklonů je věští než 2,5 ‰.
- odpor oblouku se nahrazuje fiktivním stoupáním (viz. kapitola 2.1.2),

- velikost výsledného redukovaného odporu (r_r) jednoho úseku se vypočte jako:

$$r_r = \frac{\sum r_s \cdot l_s + \sum r_o \cdot l_o}{\sum l_s}$$

kde $\sum l_s$ je délka celého sloučeného úseku a jednotlivé členy sumace vždy odpovídají jednomu konstantnímu elementu (v tomto případě sklonu respektive oblouku),

- zároveň musí být splněna podmínka, že:

$$\sum l_s \leq 2{,}5 \sum l_o$$

neboli: celková délka sloučeného úseku nesmí být větší než dvouapůlnásobek délek v něm zahrnutých oblouků.

3. Analýza vstupních dat

Vstupní data jsou uložena v tabulkách editoru MS Excel. Název každého souboru odpovídá trase včetně některých významných nácestných stanic a dále typu dat, jež uchovává. Jednotlivé stanice jsou odděleny mezerami a pomlčkami, obsah navazuje podtržítkem bez mezer na poslední z nich. Například *brno -tišnov - nedvědice - žďár nad sázavou_sklon.xls*, *liberec - česká lípa - děčín - ústí nad labem_zastavky.xls*.

První řádek je vždy záhlavím se zkráceným označením daného sloupce. Pokud se vyskytují na trase dvoukolejné traťové úseky, údaje pro další kolej jsou uvozeny návěštím „2. kolej“ v prvním sloupci a odděleny od parametrů 1. koleje prázdným řádkem. Všechny tabulky obsahují sloupce s kilometrickou polohou začátku a konce a názvy nejbližších dopraven. Pokud se daný údaj vyskytuje přímo v dopravně, název je uveden v prvním sloupci vždy i se zkratkou jejího typu (např. žst., dD3) a druhý sloupec je buď prázdný, nebo upřesňující (v tom případě je uzavřen v závorkách). Další sloupce jsou specifické pro jednotlivé tabulky (např. poloměr, jméno zastávky).

Řádky na sebe z hlediska staničení velmi často nenavazují, můžeme pozorovat následující deformace koheze:

- přesunuté řádky,
- duplikované řádky,
- překrývající se řádky (staničení řádku začíná ještě před koncem řádku předchozího),
- chybějící řádky,

Lze narazit rovněž na chybné údaje, např. řádek se sklonem 999 promile. Tento údaj, jak jsem zjistil, se vztahuje k částem dopraven, na které, nebo ze kterých se souprava nemůže přímou cestou (bez manipulace) dostat. V tabulce s traťovými rychlostmi je v mnohé dopravně více variant – rozdíl mezi průjezdní kolejí a ostatními staničními kolejemi (pojížděnými při jízdě do odbočky).

Další nespojitosti jsou dány přirozeně ze strany dopravní cesty:

- hybridní hektometrovníky,
- přechod na jinou trať,
- alternativní trasa (např. Praha – Beroun přes Loděnici / Karlštejn).

4. Jádro programu

Uživatel zadá cestu k souboru se sklonem, ostatní tabulky jsou nalezeny automaticky, nebo je opět vyzván uživatel. Pak jsou z dat vytvořeny automaticky souvislé bloky. Konkrétně dochází k mazání duplicitních a nepotřebných řádků, dále k připojení řádků zjevně souvisejících.

Bloky sklonu setřídí uživatel v grafickém rozhraní. Zároveň je může jednotlivě prohlížet respektive upravovat jako tabulku. Je vytvořena referenční relativní kilometráž. Bloky ostatních parametrů jsou samočinně přiřazeny k hotové sestavě sklonu a uživatel je tak zpravidla jen zkontroluje, v případě potřeby přeřadí nebo edituje v tabulce. Při každém přiřazení je sklon synchronizován tak, aby jeho data byla nepřetržitá po celé délce trasy. Ve chvíli, kdy už jsou seřazeny všechny parametry, program jim dopočítá relativní kilometráže dle hotové reference – sklonu. Z údajů o sklonu a obloucích se na závěr sloučí nová struktura redukovaného odporu a data pak již mohou být převedena do žádaného výstupního formátu.

4.1. Základní proměnné

Každá z původních tabulek obdrží svou strukturu[1] zapouzdřující údaje v maticích či vektorech o totožném počtu řádků. Matice obsahují vždy logicky k sobě patřící sloupce (např. počáteční a koncová kilometrická poloha, předchozí a následující dopravna atd.). Řádky všech matic a vektorů si v rámci jedné struktury navzájem odpovídají – dohromady tvoří kompletní informaci o jednom elementu (např. sklonu, oblouku, zastávce atd.). Základní složení všech struktur je graficky znázorněno na Obr. 4.1. Z obrázku je též zřejmé, proč je koncept jednotlivých struktur výhodný. Některé matice jsou ve strukturách stejné a díky stejnému pojmenování pak lze vytvářet univerzální funkce.

Například matice *km*, která informuje o počátku a konci staničení patříčného řádku je všude významově totožná. Podobně je to s maticí *opPoints*, která drží informace o dopravnách, mezi kterými se ten který řádek nachází. Rovněž dva sloupce má matice *kmRel*, zaplněná počáteční a koncovou relativní distancí od počátku trasy. Vektor *gradient* ve struktuře sklonu informuje o sklonu v jednotkách promile. Téměř totožný je vektor *resistance*, do něj jsou na konci běhu programu uloženy sloučené redukované jízdní odpory [N/kN]. Logický vektor *isRight* ve struktuře oblouků doplňuje položku *radius* s poloměry o směřování oblouků. Na řádcích, kde *isRight* nabývá hodnoty „true“, jsou uloženy oblouky směrující doprava od směru

[1] Struktury jsou pojmenovány ve shodě s celým programem anglicky, k překladu jejich názvů byl využit zdroj **Chyba! Záložka není definována.Chyba! Nenalezen zdroj odkazů.Chyba! Nenalezen zdroj odkazů.**.

jízdy. Rychlosti jsou číselným vektorem *speed* v téměř stejnojmenné struktuře a konečně v *name* nalezneme názvy všech zastávek na trase.

Obr. 4.1 Klíčový obsah hlavních datových struktur

Profile	Alignment	Resistance	Halts	Speed
• **gradient** • km • kmRel • opPoints	• **radius** • isRight • km • kmRel • opPoints	• **resistance** • kmRel	• **name** • km • kmRel • opPoints	• **speed** • km • kmRel • opPoints

Všechny struktury zapouzdřují ještě další pomocné matice, s nimiž uživatel přímo nepracuje. Ty jsou však pro funkčnost zcela esenciální a jejich popisu se nelze vyhnout.

4.1.1. Vektor „isClosing“

Tento logický vektor nabývá hodnoty *true* na pozicích odpovídajících řádkům, jež nejsou v tu chvíli spjaty s řádkem následujícím. Sílou této proměnné je, že umožňuje rozčlenění řádků na souvislé bloky dat. Tyto bloky využívám v programu jako základní rozlišovací jednotku, protože do nich zpravidla není nutno zasahovat (kromě viz kapitolu 5.2.2). Převážná většina úprav dat (třídění, mazání, otáčení směru, …) probíhá na jejich úrovni. Pro konstrukci vektoru je zpočátku využívána funkce *makeIsClosing*.

4.1.2. Vektor „isBothOpPoints“

Vektor je stejného typu s předchozím, nicméně tentokrát informuje o přítomnosti obou názvů na patřičných řádcích v matici dopraven. To je důležité jednak pro tvorbu uživatelské nabídky a také pro přiřazování údajů právě na základě shody názvů dopraven.

4.1.3. Řetězce „key“ a „content“

V případě těchto položek nejde o matice s daty, nýbrž o osamocené řetězce. *Key* nese název charakteristického vektoru dané struktury. Pro názornost jsou tyto vektory zvýrazněny tučně na Obr. 4.1. *Content* pak charakterizuje jedním slovem (česky) danou strukturu. Tento údaj je využit zejména v generických funkcích pro tvorbu grafického rozhraní.

4.2. Algoritmus načtení

Vzhledem k objemu dat bylo v mém původním plánu nenačítat všechny tabulky současně na začátku, nýbrž postupně až ve chvíli, kdy jsou kompletně zpracována data z tabulky předchozí. Tímto opatřením jsem sledoval snížení nároků programu na operační paměť při zpracováních. Avšak během tvorby jsem i bez využití měřicího softwaru zaznamenal složitost příkazu načtení dat z výchozích tabulek a rozhodl se vyřešit tento fenomén pomocí paralelního programování. Každá paralelní úloha je i v samotném kódu graficky oddělena, aby bylo zřejmé, kde začíná a končí.

4.2.1. Zjištění názvu

Uživatel musí zadat cestu k souboru, jmenovitě k tabulce s daty o sklonu. Zavoláním funkce *uigetfile* se zobrazí výběrové okno průzkumníka. Implicitně se zobrazují pouze soubory s koncovkou „xls“ a „xlsx“. Lze označit pouze jednu položku. Funkce dialogového okna vrací cestu (*pathname*) a název (*filename*) výběru.

Pro ostatní matice je postup odlišný, protože již znám cestu a název tabulky sklonů. Funkce *getFilename* přebírá spolu s nimi ještě textový řetězec *content*, který odpovídá českému popisu obsahu tabulky, resp. části jejího pojmenování za podtržítkem. Ve funkci je pak tato část nahrazena a výsledek je testován, zda existuje systematicky pojmenovaný soubor. Pokud ne, dochází k záměně přípony „xls“ za „xlsx“, nebo naopak a pokus se opakuje. V případě, že ani takový soubor v umístění, kde byl nalezen soubor se sklonem, neexistuje, musí být vyzván k jeho nalezení uživatel, tudíž je opětovně volána vestavěná funkce *uigetfile*.

4.2.2. Konverze z xls

MATLAB nabízí výkonný nástroj *xlsread*, který je možné použít několika způsoby. V mnou vytvořené funkci *readExcel* ji parametrizuji tak, aby byl návratovou hodnotou datový typ pole buněk, které obsahují čísla i texty v nezměněné poloze proti originálu. To je důležité zejména pro zachování vazby mezi záhlavím a sloupci. Nutným předpokladem načtení je, že data jsou uložena na prvním listě sešitu, což všechny výchozí soubory splňují.

Zmínil jsem, že jde o výkonný nástroj, zároveň je však jedním z hlavních důvodů, proč jsem přikročil k paralelizaci úloh – pro svou časovou náročnost na jedné straně a autonomii ve smyslu interakce s uživatelem na straně druhé. Tento příkaz je vpravdě ideálním kandidátem pro mandát procesu na pozadí. Zatímco je (bezobslužně) vykonáván, může uživatel ve stejném okamžiku pracovat v grafickém prostředí, aniž by docházelo k časovým prodlevám daným

nevylučitelným lidským faktorem (pomalá interakce, vyhledávání v podpůrných informacích, nerozhodnost apod.).

Tento příkaz je v případě každé tabulky vykonáván na pozadí, Tab. 4.1 uvádí odpovídající si načítanou tabulku a zobrazované grafické rozhraní z hlediska časové (nikoli významové) sounáležitosti. Většina zmíněných uživatelských rozhraní je rozebrána v kap. 5.2.

člověk / stroj	***readExcel* na pozadí**
zadání cesty k dalším souborům[1]	načtení sklonů
výběr navazujících bloků sklonů	načtení rychlostí
výběr navazujících bloků rychlostí	načtení sklonů
výběr navazujících bloků sklonů	načtení zastávek

Tab. 4.1 Přehled paralelních úloh při načítání datových struktur

4.2.3. Výběr sloupců a řádků

Z hlavního programu je v tomto okamžiku vždy volána specifická funkce pro každý typ dat zvlášť (např.: *loadAlignment, loadHalts*, …). Jejím prvním úkolem je zjištění užitečných sloupců. Do *findCol* jsou předávány následující parametry: načtené pole buněk (*array*), typ načítaných dat (*content*) a konečně výčet textových řetězců, které by hledaný sloupec měly charakterizovat, a jsou proto postupně v poli hledány. Program by pro zadaná vstupní data vůbec nemusel výčet přebírat, ale v případě, že se jejich označení v budoucnu změní, bude jednoduché tuto změnu implementovat. Pokud funkce nenajde žádnou buňku odpovídající popisům, spouští původní soubor přímo v MS Excel pro ruční výběr obsluhou. Návratovou hodnotou je číslo požadovaného sloupce v načteném poli.

Jelikož excelovská tabulka obsahuje prázdné řádky a případně údaje o 2. TK, je nutné najít indexy pouze těch řádků, které nesou informaci – volána funkce *findValidRows*. V ní se po nalezení čísla řádku s návěštím „2. kolej“ (*rowTrack2*) vyhodnotí sloupec počátečních kilometrických poloh, neboť ten musí přítomen ve všech tabulkách trasy. Nejprve je sestrojen vektor řádků s číselným typem buněk (*isNumericRow)*, jako jeho podmnožina je vytvořen vektor řádků s hodnotou (*isValueRow*). Modifikací logického součinu (průnik podmnožiny s množinou) dostáváme výsledné indexy platných řádků (logický vektor *isValidRow*). Nalezené řádky jsou potom dle významu sloupců nahrány do jednotlivých matic výstupní struktury a takto předány hlavnímu skriptu.

[1] Rozhraní se zobrazuje pouze v případě potřeby, tedy pokud selže automatický algoritmus popsaný v kapitole 4.2.1.

Poněkud odlišná je od ostatních pouze funkce *loadHalts*, protože se mezi zadanými trasami vyskytla (a v budoucnu jich může přibýt) taková, na níž nejsou žádné zastávky. Program by se snažil s tabulkou nakládat obvyklým způsobem a po několikeré interakci s uživatelem by došlo k výskytu výjimky. Modifikace proto počítá s touto variantou a vrací proměnnou *Halts* jako prázdnou. Samozřejmě je na takový výstup patřičně reagováno v hlavním skriptu.

4.3. Automatické úpravy

4.3.1. Nesmyslné a duplicitní údaje

Nesmyslné údaje jsou vyřazovány přímo v hlavní funkci ze dvou důvodů. Jednak jde o mazání unikátních původních dat, takže by měl mít programátor o tomto fenoménu povědomí a případně jednoduchý přístup. Druhým a důležitějším důvodem pak je fakt, že funkce není univerzální pro všechny struktury (není generická). Nesmyslným údajem se rozumí taková hodnota, která nemá souvislost s okolními daty a to znamená u každého typu dat něco jiného. Bylo by nejspíš zbytečné a také pomalé řešení vytvořit takto jednoduchou specifickou funkci pro pouhé jedno použití. Tyto řádky, jak bylo uvedeno v kapitole 3, nemají vůbec být brány v úvahu, a proto jsou vymazány. Nalezený logický vektor s indexy (*isPointlessRow*) je proto předán funkci *delRow* (viz. 5.1.1).

Naopak v případě duplicitních řádků je řešení zcela generické a čisté. Funkce *delEqRows* najde a odstraní duplicitní řádky podle následujícího algoritmu:

- nalezení ukončujících řádků (volání *makeIsClosing*, *getBlocksBoundaries*),
- porovnání každého ukončujícího řádku se všemi ostatními řádky dle počátečního a ukončujícího staničení a hodnoty uložené v charakteristické matici,
- pokud jsou nalezeny řádky shodující se s aktuálním řádkem v obou podmínkách uvedených v předchozím bodě, je zachován pouze jeden z nich, a sice první nalezený, který má obě dopravny, všechny ostatní jsou předány funkci delRow,
- pokud byl alespoň jeden řádek smazán, program se opakuje, jinak skončí.

4.3.2. Chybějící řádky

Tato funkcionalita je do určité míry pouze pro uživatelské pohodlí. Eliminuje totiž nespojitosti, které by po přímém dosazení libovolného sklonu byly s velkou pravděpodobností v rámci metod slučování zanedbány. Program doplňuje sklony pouze za těchto podmínek:

- řádky spolu sousedí,
- obě dopravny se shodují,
- koncové staničení prvního řádku je menší než staničení začátku druhého řádku,

- rozdíl mezi těmito staničeními, tedy celková délka doplněného úseku, je menší konstanty *ACCEPTABLE_LENGTH_FOR_AUTOCONNECTION* definované v úvodu skriptu (implicitně 100 m).

K vytvoření nového řádku je použita funkce *addRow* (kap. 5.1.1). Dopravny jsou zkopírovány beze změny, počáteční staničení se doplní tak, aby navazovalo na předchozí řádek a koncové na následující. Sklon je vypočten jako aritmetický průměr z obou obklopujících řádků.

4.3.3. Rozdělené související řádky

Funkce *connectByStationing* prochází postupně všechny ukončující řádky a hledá k nim komplementy v řádcích začínajících. Kritériem pro připojení jsou 2 podmínky. Řádky musí přesně kilometricky navazovat a alespoň jeden název dopravny musí být totožný. Pokud byl následovník nalezen, je pro spojení volána funkce *connectBlocks* a explicitně smazán indikátor ve vektoru *isClosing*. Smyčka přes ukončující řádky musí být spuštěna znova, protože byly přesunuty bloky a vektor posledních řádků už není aktuální.

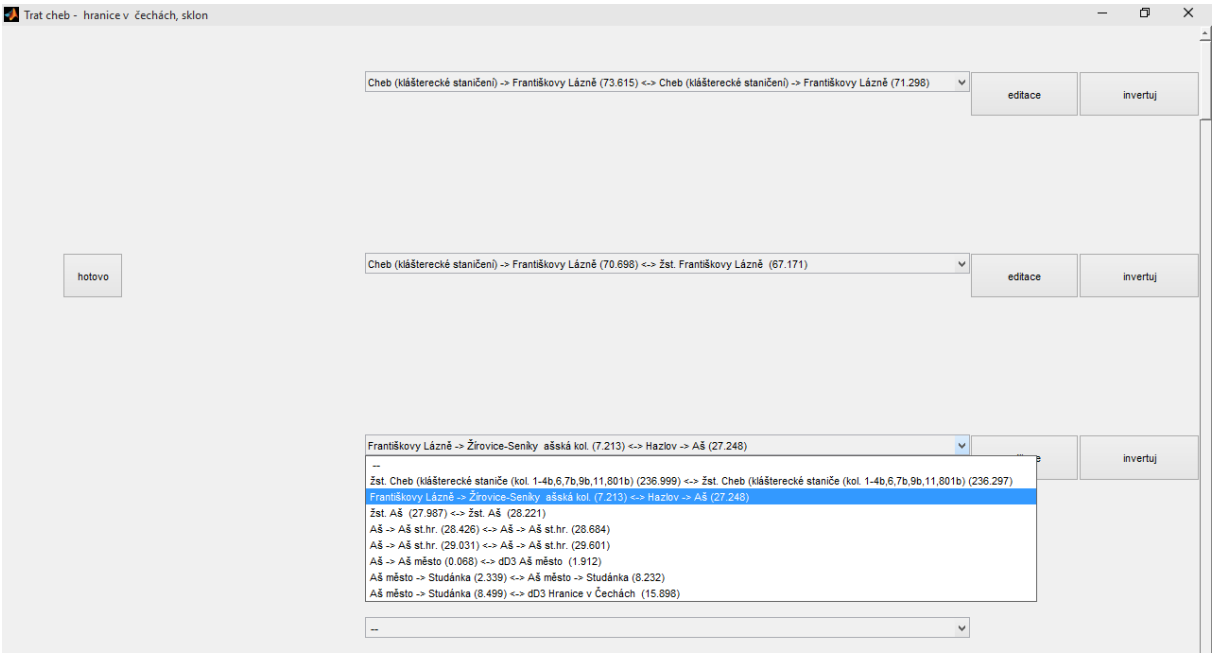
Algoritmus se opakuje, dokud dochází ke spojení, jinak je funkce opuštěna a vrací se upravená struktura.

4.4. Výběr a editace bloků

Z analýzy vstupních dat vyplývá, že nelze v každém případě programově rozhodnout o seřazení dat, aniž by hrozil výskyt systémové chyby. Je potřeba, aby uživatel vždy měl kontrolu nad průběhem řazení trasy a nestalo se, že bude například zanedbáno hybridní staničení, nebo že se spojení dvou tratí uskuteční v nesprávném kilometru a vznikne datová disproporce.

Vytvořil jsem dvě funkce, které tuto interakci zprostředkují. Rozdíl mezi nimi je takový, že *connectProfileByUser* funguje pouze s údaji o sklonu, zatímco *connectByUser* pracuje kromě sklonu ještě s jednou libovolnou strukturou. Sklon je totiž první seřazovanou sadou dat, a všechny další z něj tak mohou vycházet jako z reference. V této kapitole se budu věnovat pouze funkci *connectProfileByUser*. Rozbor jejího generického protějšku je k nalezení v části 5.2.1.

Obr. 4.2 Grafické rozhraní funkce *connectProfileByUser*



4.4.1. Uživatelská část

Bloky jsou vybírány v roletkových nabídkách. Pořadí nabídek (shora dolů) odpovídá výstupnímu pořadí zvolených bloků. Na začátku je ve všech nabídkách prázdná možnost. Nově zvolený blok se vymaže ze všech ostatních nabídek, aby nemohl být vybrán vícekrát. Zároveň se uživateli objeví dvojice tlačítek: „editace“ a „invertuj“. Tlačítko editace otevírá nové okno, v němž je možno patřičný blok prohlížet či editovat, zatímco invertování zkrátka obrátí směr za zachování celistvosti bloku.

Ve chvíli, kdy jsou všechny bloky vybrány, může uživatel okno zavřít tlačítkem „hotovo“ nebo standardně křížkem. Všechny rolety s vybranou prázdnou možností jsou skrečovány, podobně jsou smazány bloky, jež nebyly vybrány v žádném menu. Ostatní data jsou záhy seřazena a pokračuje se k přiřazení bloků ostatních dat.

4.4.2. Princip činnosti

Příkaz *figure* vytvoří obecné dialogové okno. V horní liště je zobrazena informace o trase (využití globálních proměnných *firstTitle* a *lastTitle*) a typu dat (položka *content* přebírané struktury směru). Je zřejmé, že na obrazovku se nemusí vejít všechny roletkové nabídky. Aby mohly být všechny prvky dostupné, je zapracováno tzv. „scrollování“.

Stacionárními prvky okna jsou: tlačítko ukončení (prvek *uicontrol* typu *button*) a postranní posuvník (prvek *uicontrol* typu *slider*). Ukončovací tlačítko při obsluze volá vnořenou funkci *closeWindow*, jež prostě zavírá celé okno. Obsluha změnou hodnoty

posuvníku volá funkci *scroll*. V ní je měněna pozice všech nestacionárních prvků, tedy rolovacích nabídek (*pops*), tlačítek editace (*buttonsEdit*) a invertování (*buttonsInv*). Všechny nestacionární prvky jsou uloženy jako pole buněk, stejně tak i jejich základní pozice: *popPos*, *buttonEditPos*, *buttonInvPos*.

Ve výše zmíněné funkci *scroll* je vektorizován příkaz *set* tak, abych nastavil pozice (*position*) vždy celému poli prvků podle nové hodnoty posuvníku (*slider*). Konkrétně je měněna třetí hodnota v parametru *position*, neboť ta udává vertikální pozici prvku v rámci okna.

Roletkové nabídky všechny vycházejí ze základní soupisky bloků (*defaultMenu*), jíž vytvářím s pomocí své funkce *blockToString* (kap.: 5.3.4). Zvolené bloky jsou průběžně ukládány ve vektoru *choices*. Při výběru kterékoli možnosti v roletkové nabídce je volána funkce *evaluateChoice*. Ta zjistí, zda došlo ke změně volby oproti předchozímu stavu. Je-li volba nová, musí být upraveny všechny okolní nabídky tak, aby mimo jiné nebylo možné vybrat aktuální blok znovu, a je také aktualizováno patřičné políčko vektoru *choices*. Posledním krokem je zobrazení/zneviditelnění dvojice tlačítek, které by pro prázdnou možnost neměly smysl, nicméně u zvoleného bloku musí být přítomny.

Na stisk tlačítka editace čeká vnořená funkce *editBlock*. Ta jen vypočte počáteční a koncový řádek příslušného bloku a předá je s celou strukturou do funkce *editInTable*. Po jejím ukončení musí být volána vnořená funkce *refreshDefaultMenu*. Ta zjistí nové označení bloku (fce. *blockToString*), které vloží na své místo v soupisce *defaultMenu*, aby korespondovalo s upravenými daty, neboť se mohly změnit hraniční body bloku. Poslední procedurou je aktualizace roletkové nabídky vnořenou funkcí *refreshPop*, jež vychází z upravené základní soupisky.

Na obsluhu tlačítka „invertuj“ je analogicky mapována vnořená funkce *invertBlock*, v jejímž těle dojde k předání dat algoritmu *invertDirection* popsanému v části 5.1.4. Dále proběhne stejná rutina synchronizací jako v předchozím odstavci.

Průběh funkce po vytvoření všech prvků narazí na příkaz *uiwait*, jenž zastaví další běh, dokud nebude zavřeno hlavní okno. Vykonávání obslužných vnořených funkcí grafického rozhraní tímto dotčeno není, aby v něm mohl uživatel pracovat. Jakmile je však okno zavřeno, posledním prováděným úkonem, který v těle funkce *connectProfileByUser* zbývá, je předání hotového vektoru *choices* do funkce *executeConnection* (5.3.5).

4.5. Výpočet referenční relativní kilometráže

Funkce *computeRelKm* přebírá seřaděnou strukturu *Profile*. Vytvoří v ní matici *relKm* o dvou sloupcích, počet řádků je totožný s ostatními maticemi. Do prvního políčka je vložena nulová

hodnota. Poté je sestrojen pomocný vektor *lengthsRow* s délkami jednotlivých řádků. Aplikací nativní funkce *cumsum* (kumulativní suma) na tento vektor jsou vygenerovány koncové polohy každého řádku. Počáteční polohy (kromě první – nulové) jsou totožné s koncovými předchozích řádků.

4.6. Tvorba redukovaného jízdního odporu

Na začátku funkce *mergeResistance*, jež byla pro tento účel vytvořena, jsou definice důležitých konstant. *ACCEPTABLE_GRAD_DIFFERENCE* je hodnota maximální diference odporu slučitelných bez omezení. *RESTRICTED_GRAD_DIFFERENCE* je hodnota stejného typu, pouze označuje diferenci pro odpory slučitelné maximálně do celkové délky uvedené v *RESTRICTED_LENGTH*. Posledním zástupcem je konstanta *NEGLIGIBLE_LENGTH*, tedy maximální délka krátkého úseku slučitelného i v případě nedodržení podmínky omezené diference. Hodnoty jsem přiřadil podle pravidel v kapitole 2.2, nicméně uživatel si je může případně přizpůsobit.

Do lokální proměnné *gradVirtual* je uložen výstup z funkce *reduceCurve*, v níž jsou všechny oblouky struktury *Alignment* podrobeny Röcklovu vzorci. Dočasná struktura *Temp* inkasuje matici *relKm* a *gradient* ze struktury sklonů a jsou procházeny její jednotlivé řádky. Ke každému z nich jsou nalezeny řádky oblouků, jež nějakým způsobem odpovídají relativní kilometráží. Mohou v aktuálním řádku končit, začínat, přesahovat ho, nebo v něm být vnořeny. Pro jakýkoli překryv sklonu s obloukem je spočítán redukovaný odpor tohoto úseku jako součet hodnoty sklonu a odpovídajícího řádku vektoru *gradVirtual*, což je virtuální sklon vzniklý přepočtem z oblouku. Tímto způsobem vzniknou ve většině případů nové řádky.

V tuto chvíli je ve struktuře *Temp* zcela komplexní a nejpodrobnější možná informace o redukovaném jízdním odporu. Řádky jsou slučovány tak, že se počítá průměrný odpor vážený délkou každého přibíraného řádku (*averageGrad*). Zároveň je evidována maximální a minimální hodnota sloučeného odporu (*minGrad, maxGrad*), i celková délka aktuálně slučovaného úseku (*mergedLength*). Na základě těchto údajů je podle kritérií uvedených v kapitole 2.2 rozhodnuto o přijetí každého dalšího řádku. Byl-li by některý z limitů překročen nebo pokud by znaménko nově slučovaného odporu nesouhlasilo se znaménkem dosavadního průměru, je celý slučovaný blok uzavřen a zkopírován do výstupní struktury *Resistance*. Hodnota z *averageGrad* se ukládá do vektoru sloučeného odporu. Příslušné řádky z *Temp* jsou ihned vymazány, aby nebyly použity znova. Tímto způsobem se sloučí postupně všechny řádky. Poslední operací je úprava velikosti *Resistance*, která je vždy před-alokována tak, aby se do ní vešly všechny řádky, kdyby nedošlo k žádnému sloučení.

5. Generické funkce

Následujicí funkce jsou generické tím, že dokáží pracovat s různými vstupními proměnnými. Nejsou vázány na konkrétní algoritmus a často jsou volány z různých míst celého programu. Právě na nich jsou nejlépe patrny výhody agregace klíčových proměnných do větších struktur.

5.1. Funkce pracující se strukturami

5.1.1. Funkce „delRow“, „addRow“, „copyRow“

Tyto funkce mají volitelné předání výčtu názvů upravovaných matic, který se zadává v podobě textových řetězců za povinnými parametry. Pokud není výčet použit, funkce upravuje všechny matice předané struktury.

Všechny popisované algoritmy pracují na úrovni řádků, slouží k jejich základním úpravám. Jako parametr je přebírána struktura, v jejíchž maticích chci řádky editovat a indexy těchto řádků. Funkci *delRow* lze indexy předávat numericky i logickým vektorem, ostatní pracují vždy pouze s jedním indexem. Vracenou proměnnou je vždy upravená struktura. *DelRow* řádky maže, *addRow* naopak jeden přidává. Principem přidání je replikace řádku, jehož index je předáván. Vytvoří se tedy jeho věrná kopie na následující pozici a všechny ostatní řádky jsou pochopitelně o jedno místo v pořadí posunuty.

Funkce *copyRow* zkopíruje řádek z matic struktury *SourceStruct*, jehož pořadí je předáno číselným parametrem *sourceRow*, do řádku *targetRow* struktury *TargetStruct*.

5.1.2. Funkce „sortStruct“

Funkce přebírá strukturu, v jejíchž maticích chci seřadit řádky podle nového indexu. Ten je dalším přebíraným parametrem. Vektor indexů byl použit záměrně, protože je výstupem z nativního třídícího algoritmu *sort*. Zvolený přístup je zároveň velmi pohodlný a odpovídá celkové koncepci prostředí MATLAB.

Tělem funkce je smyčka jako průchod přes vybrané matice. Jejich názvy získávám voláním *getChosenNamesMat*, jíž předávám nepovinné parametry uložené v poli *varargin*[1]. Pouhým adresováním vektorem *index* v řádcích pak získávám kýžené pořadí a celou strukturu vracím.

[1] MATLAB nepodporuje přetěžování funkcí, jak je známe z objektových jazyků. Stejného efektu však lze dosáhnout jednak tím, že argumenty nejsou implicitně typově charakterizovány ani kontrolovány a jednak pomocí fenoménu nepovinných parametrů. Je-li v deklaraci funkce uvedeno na poslední pozici mezi argumenty klíčové slovo *varargin*, lze této funkci předat libovolné množství dalších parametrů. Ty se uloží do stejnojmenného pole buněk.

5.1.3. Funkce „connectBlocks“

Jak už bylo deklarováno (v kapitole 4, základním stavebním prvkem jsou pro aplikaci spíše bloky než jednotlivé řádky. Neuspořádanost vstupních dat si žádá možnost jejich přeřazení. Funkce *connectBlocks* přebírá libovolnou datovou strukturu a s ní ještě ukončující řádek bloku, který zůstane na svém místě (*closingRow*), a počáteční řádek bloku, jenž k němu má být přesunut (*openingRowMov*). V konečném důsledku tak budou tyto řádky novými sousedy.

Z vektoru *isClosing* vypočítávám počet přesouvaných řádků (*numRowsMov*) a číslo posledního z nich (*lastRowMov*). Klíčovou proměnnou je nyní *index*. Jde o vektor s čísly řádků. Díky němu budu moct pomocí číselného indexování řádky přesunout právě jednou bez dalších mezioperací[1]. Jeho tvorba je závislá na vzájemném pořadí stacionárního a přesouvaného bloku. Pořadí řádků je pro obě varianty uvedeno v Tab. 5.1:

Tab. 5.1 Sestava indexu řádků podle původního pořadí spojovaných bloků

Stacionární blok před přesouvaným	Přesouvaný blok před stacionárním
počátek až konec stacionárního bloku	od počátku po přesouvaný blok
přesouvané řádky	mezi přesouvaným blokem a koncem stacionárního
mezi stacionárním a přesouvaným blokem	přesouvané řádky
zbytek za přesouvaným blokem	zbytek mezi stacionárním a koncem

Hotový *index* je předán funkci *sortStruct*, jež přestaví řádky vybraných matic (*varargin*) do finálního stavu.

5.1.4. Funkce „invertDirection“

Změna směru (invertování) je skloňována téměř v celém projektu. Funkce *invertDirection* ji zajišťuje. Opět je přebíráná struktura, za ní poté 2 čísla řádků. Prvním je počáteční (*openingRow*) a druhým ukončující řádek (*closingRow*). Tyto by měli, ale nemusí odpovídat začátku a konci jednoho bloku.

Obracení směru je citlivé na typ struktury, resp. konkrétní matici jednak z hlediska významu a jednak z hlediska datového typu. Je zřejmé, že jsou obraceny pouze matice a nikoli

[1] Pro zajímavost jsem nechal v souboru funkce také svou původní implementaci v podobě neaktivního kódu. Seřaďování v ní bylo realizováno přímo s daty, což vyžadovalo jejich několikeré kopírování a mazání. Šlo o jeden z prvních vytvářených algoritmů. Srovnání s novou (pochopitelně rychlejší) verzí chápu jako pomyslný vektor změny mé obratnosti v prostředí MATLAB a také částečnou demonstraci jeho výhod.

například osamocené řetězce. Proto je volána fce. *getChosenNamesMat*, díky níž je možno explicitně zvolit zpracovávané matice. S těmi je poté manipulováno jednotlivě.

Má-li matice 2 sloupce (žádná s větším počtem se ve strukturách nevyskytuje), musí být mezi sebou vyměněny. Jedinou výjimkou jsou řádky matice *opPoints*, v nichž je dokumentována pouze jedna dopravna. Takové případy chci nechat v původním formátu (viz kap. 4.1.2). Všechny ostatní řádky jsou standardně vyměněny s využitím pomocné proměnné (*temp*), která byla před-alokována na patřičný datový typ.

Má-li však matice pouze jeden sloupec (jde tedy o vektor), je v mnohém případě vyžadován specifický přístup popsaný v Tab. 5.2.

Tab. 5.2 Popis specifických úprav vektorů ve strukturách při invertování

Název vektoru	Úprava
gradient	změna znaménka (mění se smysl sklonu)
isRight	negace (mění se směr oblouku)
isClosing	rotace celého vektoru o jednu pozici (z počátečních řádků se stávají koncové)

Poslední krok, který je pro všechny matice společný, je samotné obrácení – přeskupení řádků. Toho je dosaženo velmi jednoduše a elegantně přes interval s negativním krokem. Výsledný vektor *index* předávaný funkci *sortStruct* tedy obsahuje všechny řádky matice v původním pořadí s výjimkou rozmezí mezi *openingRow* a *closingRow*, ty jsou seřazeny v opačném pořadí.

5.1.5. Funkce „makeIsClosing“ a „getBlocksBoundaries“

První funkce vloží do přebírané struktury vektor *isClosing*. Je založená na jednoduchém algoritmu zjišťujícím návaznost staničení prvního řádku s druhým, druhého s třetím a tak dále. Je zřejmé, že funkce nevyhodnocuje poslední řádek (ten je ukončující za všech okolností) a lze ji volat kdykoli v průběhu programu, pokud mají hranice bloků respektovat původní navazující bloky.

Komplementární funkcí je *getBlocksBoundaries*, jež je využívána téměř ve všech mnou vytvořených algoritmech. Ta totiž na základě hotového logického vektoru ukončujících řádků zpracovávané struktury vrací dva číselné vektory číselných indexů – počáteční a koncové řádky jednotlivých bloků. De facto jde jen o vhodnou aplikaci nativního příkazu *find*, nicméně při četném použití je každé zjednodušení výhodné.

5.1.6. Funkce „makeIsBothOpPoints“

Z názvu lze odvodit, že do výstupní struktury je přidán vektor *isBothOpPoints*. Tvorba tohoto vektoru je založena na analýze dat. Je potřeba označit ty řádky, v nichž matice *opPoints* informuje ve svém druhém (koncovém) sloupci o dopravně, k níž je směřováno. Tento sloupec tedy musí být neprazdný a zároveň nesmí být obsah uzavřen v závorkách, to by znamenalo, že jde o pouhé upřesnění dopravny v prvním sloupci. Výsledek je získán za pomocí vestavěných metod *strcmp* a *strncmp*. První porovnává celý řetězec, zatímco druhá varianta pouze zadaný počet znaků – pro naši aplikaci je to porovnání prvního znaku se znakem začátku závorky.

5.1.7. Funkce „getNamesMat“, „getNamesRelMat“, „getNamesOther“, „getChosenNamesMat“

U generických funkcí je téměř vždy potřeba zajistit, aby byly zpracovány pouze vybrané typy položek struktury. V kapitole 4.1 jsou tyto blíže projednány. Pro zjištění jmen všech položek sice existuje funkce *fieldnames*, ta však nikterak neodliší jejich typy. Vzhledem k mizivé typové kontrole prostředím MATLAB jsou všechny brány rovnocenně a musí být odebrány explicitně. Funkce *getNamesMat* vrací jména těch položek, které jsou maticemi, tedy obsahují řádky s jednotlivými úseky trasy. Tuto podmínku jsem původně zjišťoval právě přes počet řádků, jenž musel odpovídat počtu řádků charakteristické matice. Nešlo však o jednoznačnou identifikaci, proto jsem přidal výčet dvou řetězců („key“ a „content“), které každopádně nesmí být zpracovány.

Funkce *getNamesRelMat* vychází z té předchozí (také ji ve svém těle volá). Dá se říct, že její výstup jen filtruje na ty matice, jež jsou zajímavé pro uživatele, tedy přímo se týkají tratě, od matic systémových, pomocných (např. *isClosing*). Implementace využívá porovnání názvů s konstantními řetězci definovanými přímo v těle funkce. Pokud by tedy při úpravách programu přibyly další systémové matice, je nezbytné sem jejich názvy přidat.

Poslední zástupkyně umožňuje konfrontovat vybrané názvy položek (odtud také její název *getChosenNamesMat)* s výstupem funkce *getNamesMat*. Konkrétně vrací ty názvy z přebírané množiny, které jsou maticemi. Je potřeba si uvědomit, že názvy jsou předávány jako pouhé řetězce, nikoliv výčtovým typem, tudíž tato verifikace je nutná ve všech funkcích, které umožňují výběr zpracovaných položek struktury, mimo jiné proto, aby nedošlo k výjimce způsobené přístupem na neexistující položku. Jednoduše shrnuto; z přebíraných názvů jsou odstraněny ty, které nejsou maticemi nebo se vůbec ve struktuře nevyskytují.

5.2. Funkce uživatelských rozhraní

5.2.1. Funkce „connectByUser“

Účelem tohoto rozhraní je výběr bloků nové struktury (*StructureNew*) a jejich přiřazení ke struktuře referenční (*StructureRef*), jejíž bloky jsou seřazeny. Ve svém kódu používám jako referenci vždy údaje o sklonech. Uživatel listuje mezi jednotlivými referenčními bloky a individuálně k nim přiřazuje nové v roletových nabídkách. Je evidentní, že funkce vychází z varianty dedikované pouze struktuře sklonu (*connectProfileByUser*), nicméně doznala několika zásadních změn.

Především je využito funkce *assignByStationing*, která navrhuje nové bloky k referenčním na základě staničení. Získaný přehled je ještě upravován, protože při tomto přiřazení by mohlo docházet k částečné mystifikaci uživatele, pakliže by se v trase vyskytly dva úseky z různých tratí s překrývajícími se kilometrickými polohami, mohly by být nové bloky přiřazeny ke špatné referenci. Proto zpřesňuji výběr díky názvům dopraven. Každý nový blok porovnávám po řádcích s jeho referenčními kandidáty (dle funkce *assignByStationing*). Jsou sečteny všechny shody (proměnná *numHits*) a vyděleny počtem řádků reference. Vznikne poměr shodných řádků dle dopraven pro každý z vybraných referenčních bloků (*ratiosHits*). Je-li hotov poměr u všech bloků, vyhledá se mezi nimi maximální člen, jenž tudíž odpovídá referenčnímu bloku s nejvyšší pravděpodobností správného přiřazení. Při otevření je tak již převážná většina bloků správně přiřazena, což uživateli ušetří signifikantní množství práce.

Pro zobrazení dvojice editačních tlačítek v pravé části platí opět pravidlo s neprázdnou zvolenou možností (tu nemá význam editovat) a pro tlačítko invertování přibývá ještě podmínka, že se zobrazuje pouze u těch bloků, které nebyly přiřazeny automaticky (nejsou v *defaultChoices*), neboť u nich je garantováno, že jsou srovnány v souladu s referencí.

Související změnou je stránkování. Každá stránka náleží jednomu referenčnímu bloku, jehož popis je v levé části okna mezi tlačítky přesunu na jinou stránku. Tato tlačítka jsou analogicky popsána sousedními bloky (na něž odkazují), aby měl uživatel přehled o podobě předchozího i následujícího bloku. Číslo cílové stránky je zároveň uloženo v položce *userData* obou tlačítek. Stisk tlačítek spouští vnořenou funkci *showPage*, která na základě informace v *userData* zjistí nová čísla první a poslední roletkové nabídky a změní viditelnost prvků takto: Všechny nestacionární prvky dosavadní stránky jsou zneviditelněny. Poté jsou zviditelněny všechny roletkové nabídky patřící na novou stránku a k nim příslušná tlačítka. Tlačítka jsou však ještě individuálně posuzována podle pravidel uvedených v předchozím odstavci.

Pro prostředí stránek je modifikována vnořená funkce *scroll*. Je zbytečné (a pomalé) měnit pozici všech nestacionárních prvků, tudíž nyní pracuje pouze s výčtem těch, jež jsou zobrazitelné na aktuální stránce.

Ve chvíli, kdy je uživatel hotov a okno uzavře, musí být vyhodnoceny polohy vybraných nových bloků pro každou referenci. Program automaticky tyto vztahy vyhledává a dále spolupracuje s uživatelem, pokud dochází k nejednoznačnosti, nebo vznikne podezření na uživatelskou chybu. Nejprve je kontrolováno, zda se nový blok překrývá s referencí staničením. Pokud ano, je kontrolována shodnost směru. Jestliže blok některou z těchto podmínek nesplňuje, lze předpokládat, že nový blok nemá být přímo zahrnut do rozsahu vybraného bloku sklonů a je potřeba pro něj referenční strukturu upravit. Stejný předpoklad platí pro ty z nových bloků, které se částí staničení s referencí překrývají, ale nejsou v ní vnořeny, nýbrž ji přesahují. V těchto případech je kontrolováno, zda nový blok nepřekrývá přilehlé referenční bloky a je na uživateli, aby z nich vybral ty, které skutečně mají být sloučeny v jeden celek. Čísla těchto bloků jsou uložena ve vektoru *coveredBlocks*.

Slučování referencí je v gesci vnořené funkce *connectCoveredBlocksRef*. Ta zacelí mezery mezi všemi bloky uvedenými ve vektoru *coveredBlocks*, čímž vznikne po aplikaci funkce *makeIsClosing* jeden souvislý blok. Celý tento blok je i na konci a na začátku upraven tak, aby nebyl přesahován zakrývajícím přiřazeným blokem, tedy aby celková délka nového i sloučeného bloku byla minimálně stejná. Takové opatření je zcela nutné, neboť potřebuji zaručit, aby údaje o sklonu byly nepřetržité po celé délce trasy bez disproporcí. Do nových kilometrických poloh sklonu nemůže být implicitně uložena žádná hodnota gradientu, jinak by mohlo snadno dojít k systémové chybě při následném zpracování redukovaného odporu. Proto je v nových řádcích uložena hodnota *NaN* („not a number“), která sice splňuje podmínku číselného datového typu, ale jakýkoli výpočet s ní vyvolá automaticky výjimku. Názvy dopraven jsou doplněny z okolních řádků.

Pokud však nový přiřazovaný blok odpovídá staničením i směrem a nepřekrývá zároveň předchozí přiřazený nový blok, mohou být jeho řádkům vypočteny relativní kilometrické polohy ve funkci *computeRelKmFromRef*. Tento blok je brán jako uspěšně přiřazený.

Je-li však jakkoli upravena referenční struktura, musí se přestavit celé grafické rozhraní. Pro zachování vazeb mezi všemi bloky je nejjednodušší funkci restartovat. To ve skutečnosti zajišťuje hlavní skript na základě indikátoru *isReady*, který funkce vrací spolu s oběma přebíranými strukturami. Pokud má být funkce restartována, je příznak nastaven na logickou hodnotu *false*. Až když jsou všechny nové bloky správně přiřazeny a je jim vypočtena patřičná relativní kilometráž, vrátí funkce hotové struktury a v proměnné *isReady* hodnotu *true*.

5.2.2. Funkce „editInTable“

MATLAB umožňuje grafické zobrazení dat v podobě tabulky. Toho využívám k implementaci funkce pro přímou editaci dat na elementární úrovni. Jako parametr je přebírána struktura a program z ní vybere ty matice, které jsou důležité pro uživatele (volána fce. *getNamesRelMat*). Nepovinnými parametry jsou ještě počáteční a koncový řádek zobrazený v tabulce. Toho lze využít pro umožnění editace pouze jednoho souvislého bloku.

Aby byla celá funkce generická, je nutno sloupce všech matic jednotně seřadit dle univerzálního mustru. Tento mustr si ukládám do proměnné *colAssign*, jíž tvoří vektor lokálních struktur s položkami *matrix* (název odpovídající matice) a *col* (číslo sloupce v této matici). Každé políčko vektoru tak náleží jednomu sloupci finální tabulky. Z Tab. 5.3 je zjevné, jakým způsobem jsou data do těchto proměnných ukládána. Matice, které jsou u všech typů základních struktur stejné (viz kap.: 4.1), zobrazuji pro přehlednost na začátku tabulky. Hned za nimi následuje charakteristická matice. Další sloupce jsou seřazeny na základě výstupu z funkce *getNamesRelMat* v pořadí, v jakém byly matice v průběhu programu vytvářeny. Samozřejmě nejsou roztrženy sloupce těchto matic.

Tab. 5.3 Fixní přiřazení počátečních sloupců v tabulce zobrazované uživateli

Číslo sloupce tabulky	Název matice	Číslo sloupce v matici
1	*km / relKm*	1
2	*km / relKm*	2
3	*opPoints*	1
4	*opPoints*	2
5	*key*	1

Příkaz *uitable* vytvoří tabulku zobrazující data nahraná příznačně do proměnné *data*. Obsahem musí být pole buněk, které vytvářím ve vnořené funkci *prepareData*. Do záhlaví sloupců jsou zkopírovány názvy odpovídajících matic (viz Tab. 5.3), což je sice velmi elegantní a jednoduché řešení, jenže vnáší určitý nešvar do lexikální stránky grafického prvku. Pokud by měly být zobrazovány jiné (české) řetězce, bylo by nutné vytvořit mapu těchto názvů pro všechny možné matice a tu případně udržovat v případě jejich příbytku.

Výše zmíněná funkce *prepareData* jen připojuje sloupce za sebe dle mustru v proměnné *colAssign* a ty, které obsahují číselné nebo logické hodnoty jsou zde zapouzdřeny do buněk. Mimochodem logické hodnoty jsou poté v tabulce automaticky zvýrazněny binárním grafickým prvkem zaškrtávacího políčka.

Tabulka neslouží pouze k zobrazení dat. Jejím kruciálním posláním je umožnění přímé editace. Kdykoli je některé políčko editováno, dochází k volání vnořené funkce *editData*. V té musí být nejprve rozklíčována odpovídající matice a její sloupec (samozřejmé využití mustru *colAssign*). Dále je zjišťováno, zda je nový obsah buňky neprázdný. Všechna numerická data musí být vyplněna, je zobrazena varovná hláška, pokud tomu tak není, zatímco u textových je tolerováno též prázdné pole. Každá nová neprázdná hodnota je uložena (odpovídá-li její datový typ) a celá tabulka je aktualizována.

Úprava buněk ještě není vyčerpávajícím výčtem možné interakce. Uživatel může také chtít přidávat nové řádky, nebo je mazat. Toto je umožněno přes označení některé buňky daného řádku a výběru příslušné možnosti v kontextové nabídce (zobrazené při stisku pravého tlačítka). Implementace obsluhy tkví v ukládání indexu řádku označené buňky do proměnné *rowSelected*, což zajišťuje velmi jednoduchá vnořená funkce *evalSelection*. Metody *addRowTable* a *delRowTable* spouštěné z kontextového menu již jen přepošlou obsah proměnné *rowSelected* do univerzální fce. *addRow*, resp. *delRow* a zajistí aktualizaci tabulky, aby byla hned synchronizována a nedošlo k přetržení vazby mezi ní a strukturou.

5.3. Ostatní pomocné funkce

5.3.1. Funkce „iif“

Název je spojením a zkrácením výrazu „inline if“. Jde o implementaci tzv. ternárního operátoru z rodiny jazyků C, ovšem je navíc adaptován na maticové prostředí MATLAB. Pořadí parametrů je analogické;

- podmínka,
- výraz vracený při splnění podmínky,
- výraz vracený při nesplnění podmínky.

Nejen návratové hodnoty mohou být pole, nýbrž také podmínka nemusí být pouhým skalárem. Bude-li podmínka definována jako logická matice, výrazy se poskládají do jakési mozaiky podle logických hodnot pravda / nepravda na odpovídajících pozicích. Budou-li například výrazy pole o rozměrech 3x2 a podmínka bude logickou maticí o velikosti 3x1, potom celková velikost výstupní proměnné bude 9x2. Mozaikovou replikaci zajišťuje nativní funkce *repmat* v kombinaci s logickým indexováním.

Tato funkce má díky svému charakteru určitá omezení. Jsou-li podmíněné výrazy typu řetězec, musí být v jejím těle kvůli replikaci převedeny na datový typ buňka, jinak by došlo k výskytu výjimky. Další záludnost spočívá v samotném volání této funkce. Výrazy jsou totiž

před předáním standardně vyhodnoceny bez ohledu na podmínku – na rozdíl od konvenčního větvení rozhodovacím blokem. Je tak nutno dbát zvýšené opatrnosti při rozhodování např. na základě datového typu, velikosti pole a podobně.

5.3.2. Funkce „getBeginningBlocks“, „getEndingBlocks“, „getExceedingBlocks“

Pro zjišťování vztahů mezi bloky co do staničení jsem vytvořil tuto sadu funkcí. Společným rysem jsou vstupní a výstupní parametry. Posloupnost argumentů je následující: referenční struktura (vše s ní spjaté je pojmenováno s doložkou *Ref*), počáteční řádek referenčního bloku, koncový řádek referenčního bloku a konečně přiřazovaná struktura (doložka *New*). Na výstupu lze očekávat pole s čísly bloků – dle konkrétní funkce. *GetBeginningBlocks* vrací ty, jež v rozmezí referenčních řádků svým staničením začínají, *getEndingBlocks* naopak vrací končící. Jejich logickým součinem lze jednoduše získat bloky vnořené, nicméně je potřeba najít také bloky z obou stran přesahující, a to právě nabídne funkce *getExceedingBlocks*.

Implementace všech funkcí je velmi podobná. Nejprve je vyhodnocen směr referenčního bloku (zda staničení stoupá, či klesá), na základě čehož jsou přiřazeny indexy sloupců s počátky a konci staničení referenčních řádků (*colLoRef* a *colHiRef*). Dále je vytvořen logický vektor, v němž pozice „true“ označují tentokrát přiřazované bloky s rostoucím staničením (*isRisingNew*). Využitím logického indexování vytvořím vektory, označující řádky s počátky (*rowsLoNew*) nebo konci (*rowsHiNew*) staničení přiřazovaných bloků; totéž analogicky pro sloupce (*colsLoNew* a *colsHiNew*). Nuže budu-li hledat úplný počátek staničení třeba 2. přiřazovaného bloku, jeho řádkovou souřadnici najdu na pozici *rowsLoNew*(2), sloupcovou pak v *colsHiNew*(2).

Zjištěné indexy je ještě potřeba převést na tzv. lineární souřadnice[1], jinak by je MATLAB zpracovával jako oblast, nikoli diskrétní body. Jsou za tímto účelem předány funkci *sub2ind*. S hotovými vektory lineárních indexů (*indicesLo* a *indicesHi* – doložku *New* v názvu vynechávám, jelikož tato proměnná nemá v referenci protějška, a tudíž nehrozí záměna) již provádím operace srovnání patřičných staničení. Podmínky jsou uvedeny přehledně v Tab. 5.4. V jejím prvním sloupci uvádím označení funkce, ze které bude blok vrácen, splní-li obě podmínky uvedené v daném řádku.

[1] Lineární index je skalární a jeho hodnota odpovídá pořadí prvku, kdyby se celé pole po jednotlivých sloupcích rozvinulo v jeden dlouhý vektor. Tohoto indexování lze použít v libovolném poli bez jakýchkoli přechozích úprav. Vestavěná funkce *sub2ind* indexy vypočítá na základě rozměrů cílového pole.

Tab. 5.4 Podmínky pro zařazení bloků do jednotlivých kategorií překryvu staničení

	Podmínka začátku staničení	Podmínka konce staničení
getBeginningBlocks	>= začátek reference, < konec reference	žádná
getEndingBlocks	žádná	<= konec reference, > začátek reference
getExceedingBlocks	< začátek reference	> konec reference

Výsledky srovnání jsou logické vektory, jež, podrobivše se funkci *find*, poskytnou kompletní žádaný soupis odpovídajících bloků dané kategorie.

5.3.3. Funkce „assignByStationing“

Tento algoritmus de facto aplikuje funkce popsané v předchozí kapitole. Vstupními parametry jsou 2 struktury – referenční (*StructureRef*) a přiřazovaná (*StructureNew*). Pro každý referenční blok (vymezený řádky uloženými v proměnných *currentOpening* a *currentClosing*) hledám odpovídající bloky začínající (*beginningBlocks*), končící (*endingBlocks*) a přesahující (*exceedingBlocks*). Všechna čísla těchto bloků jsou následně sloučena do jednoho vektoru (*corresp*) a seřazena podle začátku jejich staničení. Je-li v aktuálním referenčním bloku staničení se stoupajícím trendem, seřadí se nové bloky vzestupně a naopak. Pokud je některý z bloků potřeba invertovat, aby odpovídal svým směrem referenci, přidá se k jeho označení záporné znaménko. Výsledný vektor *corresp* tedy obsahuje čísla bloků například: 5 -2 -3 -4 17, kde 1. v pořadí podle trendu staničení reference je blok číslo 5 nové struktury, bloky 2 až 4 nejsou ve stejném směru jako reference a 17. blok výčet odpovídajících uzavírá.

Zdůrazňuji, že vektor *corresp* platí pro jeden blok reference, takže dohromady jsou všechny postupně uloženy do před-alokovaného pole buněk *correspBlocks*, jež je návratovou proměnnou funkce.

5.3.4. Funkce „blockToString“

Spojení „to string“ je velmi často používané zejména u metod v objektových jazycích. Popisovaná funkce je, řekněme, značně aplikovaná. Ze zadané struktury a počátečního a koncového řádku určitého bloku vytvoří krátký řetězec, který má daný blok shrnout pro uživatele. Jako nejvýmluvnější jsem vyhodnotil informaci o počátečním a koncovém bodu, přičemž se bude uživateli zobrazovat kilometrická poloha a názvy dopraven na těchto řádcích. Samozřejmě je respektován aktuální směr bloku bez ohledu na trend staničení.

Při výpisu záleží, zda jsou v koncových bodech uvedeny dopravny obě, nebo nikoli. Proto využívám svojí „inline“ implementace *iif*, díky níž mohu rozhodovat lokálně, zda se bude mezi sloupce dopraven vkládat ukazatel („->“), nebo mezera – to pro případ jedné dopravny a jejího upřesnění v druhém sloupci (viz kap. 4.1.2).

Výsledný výpis tak vypadá například takto: Pacov -> Obrataň (39.350) <-> žst. Obrataň (45.159).

5.3.5. Funkce „executeConnection“

Již velmi aplikovanou funkcí je *executeConnection* volanou výhradně z funkcí uživatelského rozhraní. Jejími argumenty jsou struktura, jejíž bloky budou přerovnány a vektor voleb v podobě, jak je vybral uživatel (*choices*) – viz kap. 5.2.

Na začátku jsou odfiltrovány prázdné možnosti a volby jsou přepočteny na konkrétní bloky. Dále je ve dvou krocích vytvořen vektor řádků *index*. Pro jeho syntézu musí být vypočteny všechny řádky jednotlivých bloků dle vektoru *choices*. Je volána funkce *arrayfun*, protože operátor dvojtečky je definován pouze pro skalární operandy, zatímco já potřebuji vytvořit intervaly mezi všemi počátečními a ukončujícími řádky najednou. Funkce *arrayfun* vrací výsledek jako pole buněk s jednotlivými intervaly, takže je ve druhém kroku vyjímám a spojuji za sebe do jednoho výsledného vektoru seřazených indexů.

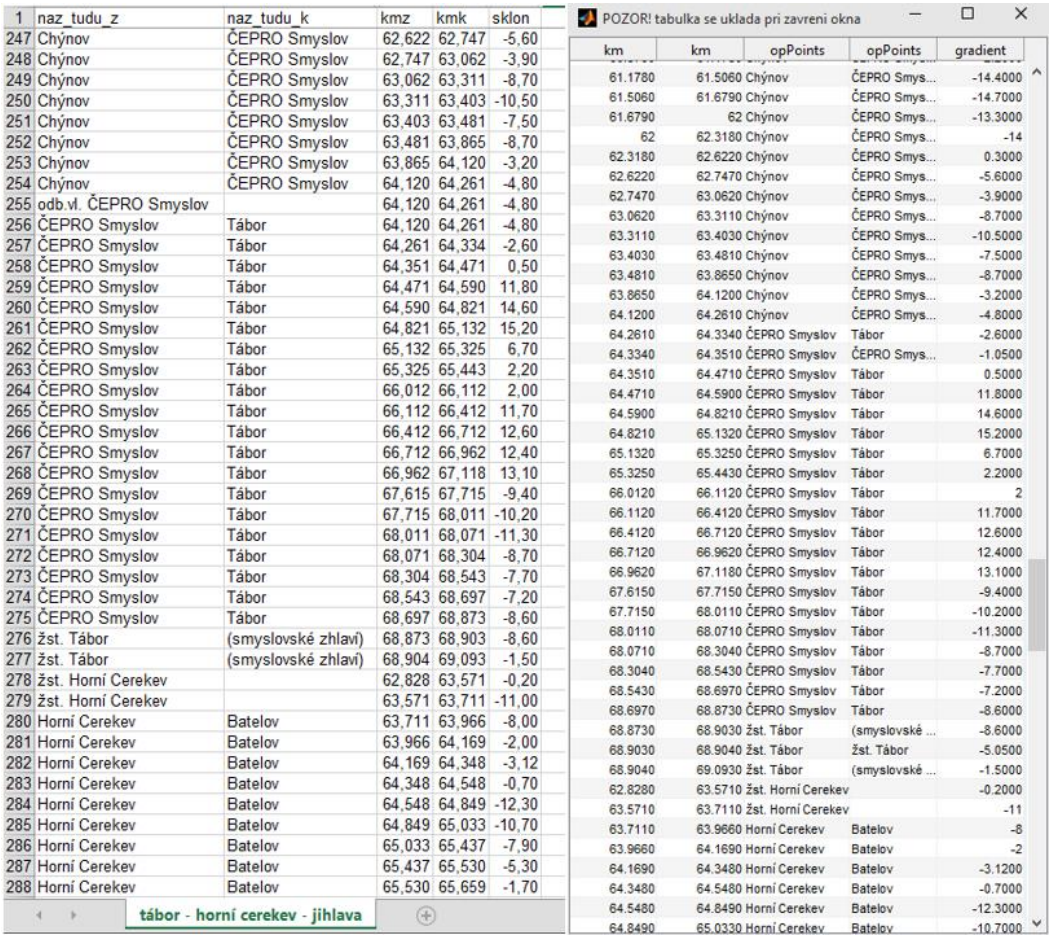
5.3.6. Funkce „computeRelKmFromRef“

Je aplikací funkce pro výpočet relativního kilometru. Je přebírána referenční a nová struktura a v nich ještě řádky vymezené navzájem si korespondující bloky. Návratovou hodnotou je celá nová struktura. Je potřeba upozornit, že funkce předpokládá správně přiřazené (a směrově srovnané) bloky. Pokud by směr neodpovídal, je vyvolána výjimka. Dále se má za to, že v referenci jsou již relativní kilometry vypočteny a lze z nich vycházet. Princip výpočtu je zachován z původní funkce *computeRelKm* až na určení počátečního políčka. To není implicitně nastaveno na nulovou hodnotu, nýbrž bude vypočteno podle vzorce:

$$relKm_{počátek} = relKm_{reference} + km_{počátek} - km_{reference}$$

6. Testování

Testování je nedílnou součástí tvorby jakéhokoli výrobku. Nejinak je tomu u softwaru. Program netestuji z hlediska časové náročnosti. Nepředpokládá se jeho nepřetržité ani velmi pravidelné využití, naopak bude spouštěn nárazově v případě potřeby. Stejně tak mi dnešní průměrná úroveň výpočetního výkonu a velikosti operační paměti umožňuje nesoustředit se na optimalizaci z hlediska čerpání systémových prostředků. Důraz je proto kladen na eliminaci systémových chyb.



1	naz_tudu_z	naz_tudu_k	kmz	kmk	sklon
247	Chýnov	ČEPRO Smyslov	62,622	62,747	-5,60
248	Chýnov	ČEPRO Smyslov	62,747	63,062	-3,90
249	Chýnov	ČEPRO Smyslov	63,062	63,311	-8,70
250	Chýnov	ČEPRO Smyslov	63,311	63,403	-10,50
251	Chýnov	ČEPRO Smyslov	63,403	63,481	-7,50
252	Chýnov	ČEPRO Smyslov	63,481	63,865	-8,70
253	Chýnov	ČEPRO Smyslov	63,865	64,120	-3,20
254	Chýnov	ČEPRO Smyslov	64,120	64,261	-4,80
255	odb.vl. ČEPRO Smyslov		64,120	64,261	-4,80
256	ČEPRO Smyslov	Tábor	64,120	64,261	-4,80
257	ČEPRO Smyslov	Tábor	64,261	64,334	-2,60
258	ČEPRO Smyslov	Tábor	64,351	64,471	0,50
259	ČEPRO Smyslov	Tábor	64,471	64,590	11,80
260	ČEPRO Smyslov	Tábor	64,590	64,821	14,60
261	ČEPRO Smyslov	Tábor	64,821	65,132	15,20
262	ČEPRO Smyslov	Tábor	65,132	65,325	6,70
263	ČEPRO Smyslov	Tábor	65,325	65,443	2,20
264	ČEPRO Smyslov	Tábor	66,012	66,112	2,00
265	ČEPRO Smyslov	Tábor	66,112	66,412	11,70
266	ČEPRO Smyslov	Tábor	66,412	66,712	12,60
267	ČEPRO Smyslov	Tábor	66,712	66,962	12,40
268	ČEPRO Smyslov	Tábor	66,962	67,118	13,10
269	ČEPRO Smyslov	Tábor	67,615	67,715	-9,40
270	ČEPRO Smyslov	Tábor	67,715	68,011	-10,20
271	ČEPRO Smyslov	Tábor	68,011	68,071	-11,30
272	ČEPRO Smyslov	Tábor	68,071	68,304	-8,70
273	ČEPRO Smyslov	Tábor	68,304	68,543	-7,70
274	ČEPRO Smyslov	Tábor	68,543	68,697	-7,20
275	ČEPRO Smyslov	Tábor	68,697	68,873	-8,60
276	žst. Tábor	(smyslovské zhlaví)	68,873	68,903	-8,60
277	žst. Tábor	(smyslovské zhlaví)	68,904	69,093	-1,50
278	žst. Horní Cerekev		62,828	63,571	-0,20
279	žst. Horní Cerekev		63,571	63,711	-11,00
280	Horní Cerekev	Batelov	63,711	63,966	-8,00
281	Horní Cerekev	Batelov	63,966	64,169	-2,00
282	Horní Cerekev	Batelov	64,169	64,348	-3,12
283	Horní Cerekev	Batelov	64,348	64,548	-0,70
284	Horní Cerekev	Batelov	64,548	64,849	-12,30
285	Horní Cerekev	Batelov	64,849	65,033	-10,70
286	Horní Cerekev	Batelov	65,033	65,437	-7,90
287	Horní Cerekev	Batelov	65,437	65,530	-5,30
288	Horní Cerekev	Batelov	65,530	65,659	-1,70

tábor - horní cerekev - jihlava

POZOR! tabulka se uklada pri zavreni okna

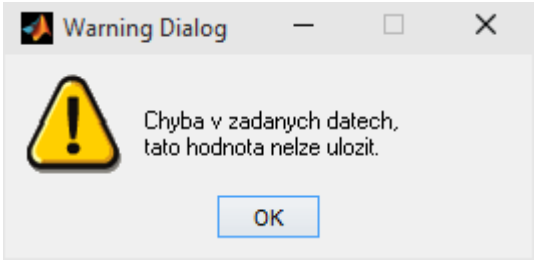
km	km	opPoints	opPoints	gradient
61.1780	61.5060	Chýnov	ČEPRO Smys...	-14.4000
61.5060	61.6790	Chýnov	ČEPRO Smys...	-14.7000
61.6790	62	Chýnov	ČEPRO Smys...	-13.3000
62	62.3180	Chýnov	ČEPRO Smys...	-14
62.3180	62.6220	Chýnov	ČEPRO Smys...	0.3000
62.6220	62.7470	Chýnov	ČEPRO Smys...	-5.6000
62.7470	63.0620	Chýnov	ČEPRO Smys...	-3.9000
63.0620	63.3110	Chýnov	ČEPRO Smys...	-8.7000
63.3110	63.4030	Chýnov	ČEPRO Smys...	-10.5000
63.4030	63.4810	Chýnov	ČEPRO Smys...	-7.5000
63.4810	63.8650	Chýnov	ČEPRO Smys...	-8.7000
63.8650	64.1200	Chýnov	ČEPRO Smys...	-3.2000
64.1200	64.2610	Chýnov	ČEPRO Smys...	-4.8000
64.2610	64.3340	ČEPRO Smyslov	Tábor	-2.6000
64.3340	64.3510	ČEPRO Smyslov	ČEPRO Smys...	-1.0500
64.3510	64.4710	ČEPRO Smyslov	Tábor	0.5000
64.4710	64.5900	ČEPRO Smyslov	Tábor	11.8000
64.5900	64.8210	ČEPRO Smyslov	Tábor	14.6000
64.8210	65.1320	ČEPRO Smyslov	Tábor	15.2000
65.1320	65.3250	ČEPRO Smyslov	Tábor	6.7000
65.3250	65.4430	ČEPRO Smyslov	Tábor	2.2000
66.0120	66.1120	ČEPRO Smyslov	Tábor	2
66.1120	66.4120	ČEPRO Smyslov	Tábor	11.7000
66.4120	66.7120	ČEPRO Smyslov	Tábor	12.6000
66.7120	66.9620	ČEPRO Smyslov	Tábor	12.4000
66.9620	67.1180	ČEPRO Smyslov	Tábor	13.1000
67.6150	67.7150	ČEPRO Smyslov	Tábor	-9.4000
67.7150	68.0110	ČEPRO Smyslov	Tábor	-10.2000
68.0110	68.0710	ČEPRO Smyslov	Tábor	-11.3000
68.0710	68.3040	ČEPRO Smyslov	Tábor	-8.7000
68.3040	68.5430	ČEPRO Smyslov	Tábor	-7.7000
68.5430	68.6970	ČEPRO Smyslov	Tábor	-7.2000
68.6970	68.8730	ČEPRO Smyslov	Tábor	-8.6000
68.8730	68.9030	žst. Tábor	(smyslovské ...	-8.6000
68.9030	68.9040	žst. Tábor	žst. Tábor	-5.0500
68.9040	69.0930	žst. Tábor	(smyslovské ...	-1.5000
62.8280	63.5710	žst. Horní Cerekev		-0.2000
63.5710	63.7110	žst. Horní Cerekev		-11
63.7110	63.9660	Horní Cerekev	Batelov	-8
63.9660	64.1690	Horní Cerekev	Batelov	-2
64.1690	64.3480	Horní Cerekev	Batelov	-3.1200
64.3480	64.5480	Horní Cerekev	Batelov	-0.7000
64.5480	64.8490	Horní Cerekev	Batelov	-12.3000
64.8490	65.0330	Horní Cerekev	Batelov	-10.7000

Obr. 6.1 Srovnání původní tabulky a stavu dat po automatických úpravách

Zkoušení jsem prováděl průběžně během tvorby po dílčích krocích. Při tomto množství funkcí je však riziko skryté chyby opravdu nezanedbatelné, takže jsem procházel programem za použití většiny tras. Nejprve jsem kontroloval načtení a automatické úpravy. Obr. 6.1 ukazuje rozdíl mezi původními údaji v aplikaci Excel a daty v mém programu zobrazenými pomocí funkce *editInTable*. Obrázek sice nezachycuje všechny varianty, nicméně zjevně jsou všechny změny v pořádku, především nedochází k žádnému úniku nebo deformaci dat.

Dalším krokem je kontrola seřazení bloků sklonu dle uživatele. Ta může být provedena bez využití tzv. „breakpointů“, neboť výsledek je hned vidět v následujícím grafickém rozhraní,

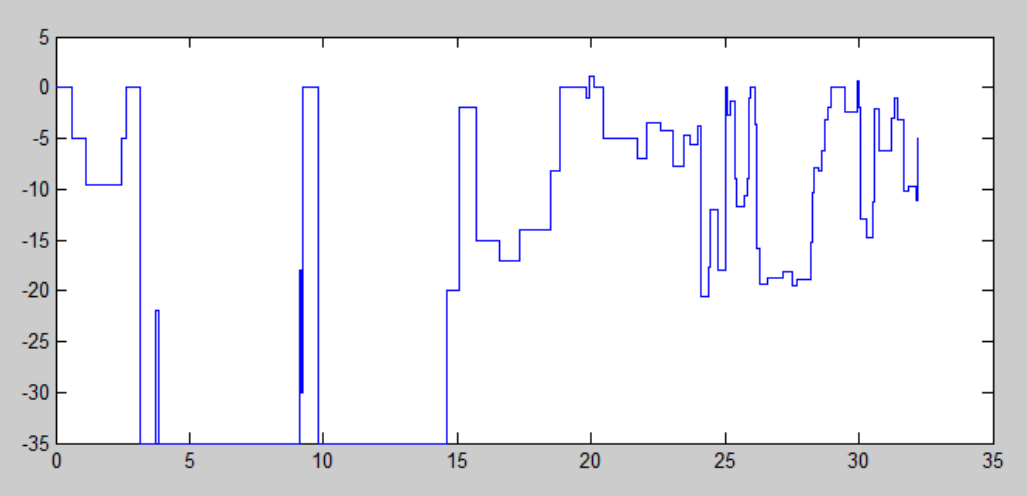
kde seřazené bloky představují jednotlivé stránky. V rámečku jsou vidět správně oddělené názvy první a poslední dopravny. Při stisku tlačítka editace se objeví odpovídající blok řádků. Změna v buňkách tabulky se ukládá s určitou prodlevou, avšak spolehlivě. Číselné sloupce jsou kontrolovány, aby do nich nebyl zapsán jiný datový typ – objeví se chybová hláška. Opačné omezení by bylo problematické, neboť číslo v rámci názvu je zcela legitimní.



Obr. 6.2 Chybová hláška ve funkci *editInTable*

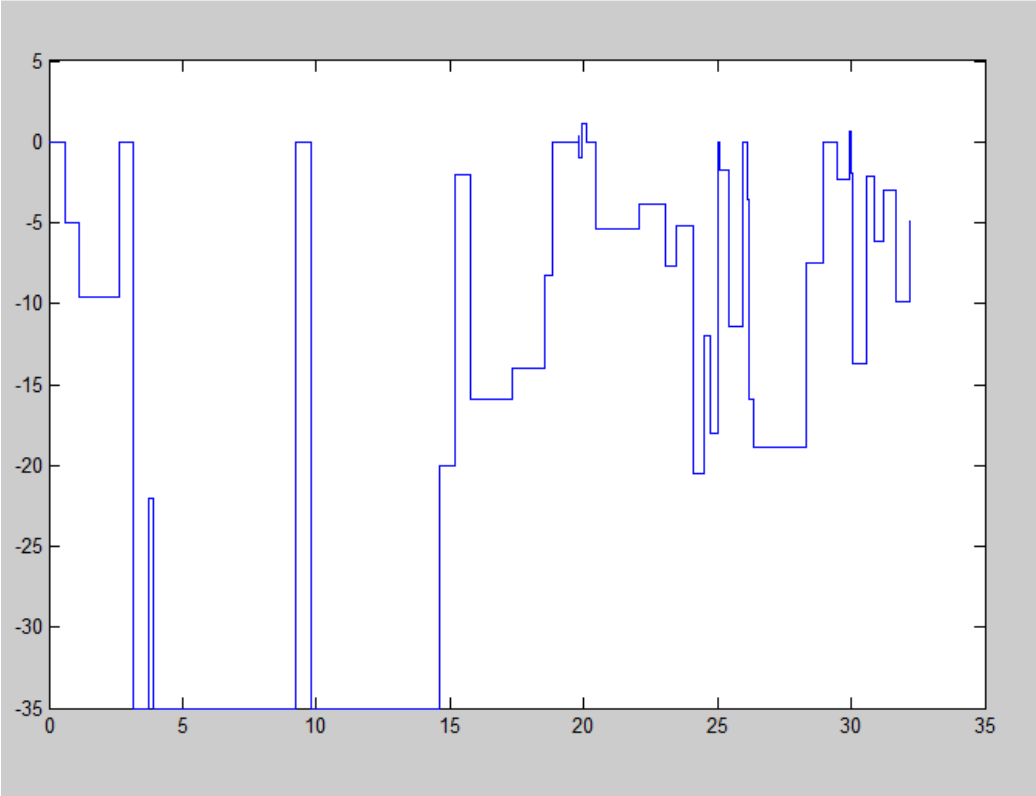
Jako kontrolu správnosti výpočtu relativní kilometráže jsem porovnával vzdálenost mezi zastávkami na trase Litvínov – Teplice v Čechách. Ve vstupních datech jsou uvedeny pouze zastávky, a sice Lom u Mostu a Háj u Duchcova. Jejich vypočtená relativní distance od Litvínova vyšla 3,055 a 8,632 km. V knižním jízdním řádě Českých drah je vyznačena tarifní vzdálenost 2 a 9 km. Uvážíme-li, že můj nultý kilometr je kdesi na vzdálenějším záhlaví litvínovské stanice, lze prohlásit, že výpočet odpovídá realitě.

Že mj. funguje správně invertování směru, dokládají Graf 6.1 a Graf 6.2. Na horizontální osu je v obou případech vynesena relativní kilometráž. V prvním grafu je vyznačena hodnota sklonu v promile. Jde vidět, že trať prudce klesá, což odpovídá reálnému stavu, takže znaménka v struktuře sklonu byla správně upravena při změně směru.



Graf 6.1 Průběh sklonů na trase Moldava – Most

V druhém grafu lze kromě vlivu oblouků také pozorovat, že základní pravidla pro slučování jsou dodržena (kapitola 2.2). Testování považuji za úspěšné.



Graf 6.2 Průběh sloučených redukovaných jízdních odporů (Moldava – Most)

7. Závěr

Můj původní záměr vytvořit autonomní algoritmy načítání dat o železničních trasách, které proběhnou bez participace uživatele nebyl dosažen. Při analýze dat vyšlo najevo, že řazení musí být řízeno a téměř s jistotou se neobejde bez přístupu k dalším informačním zdrojům o trase. Nicméně výsledkem práce je hotový program pro pohodlnou obsluhu. Má snaha o zjednodušení takového procesu, myslím, nevyšla naprázdno. Uživatel pracuje v přehledných grafických prostředích a interaguje prakticky po celou dobu chodu programu. To je umožněno zejména díky paralelnímu programování, které především šetří čas.

Program byl vytvářen a je plně kompatibilní s aplikací MATLAB R2011b, umí reagovat na neočekávané názvy a rozložení vstupních souborů. Nutným předpokladem korektního chodu je umístění souborů společných pro trasu v jedné složce a funkční instalace aplikace Microsoft Excel. Pro využití paralelního programování musí být nainstalováno také licencované rozšíření Parallel computing toolbox a počítač musí disponovat procesorem s alespoň 2 logickými jádry.

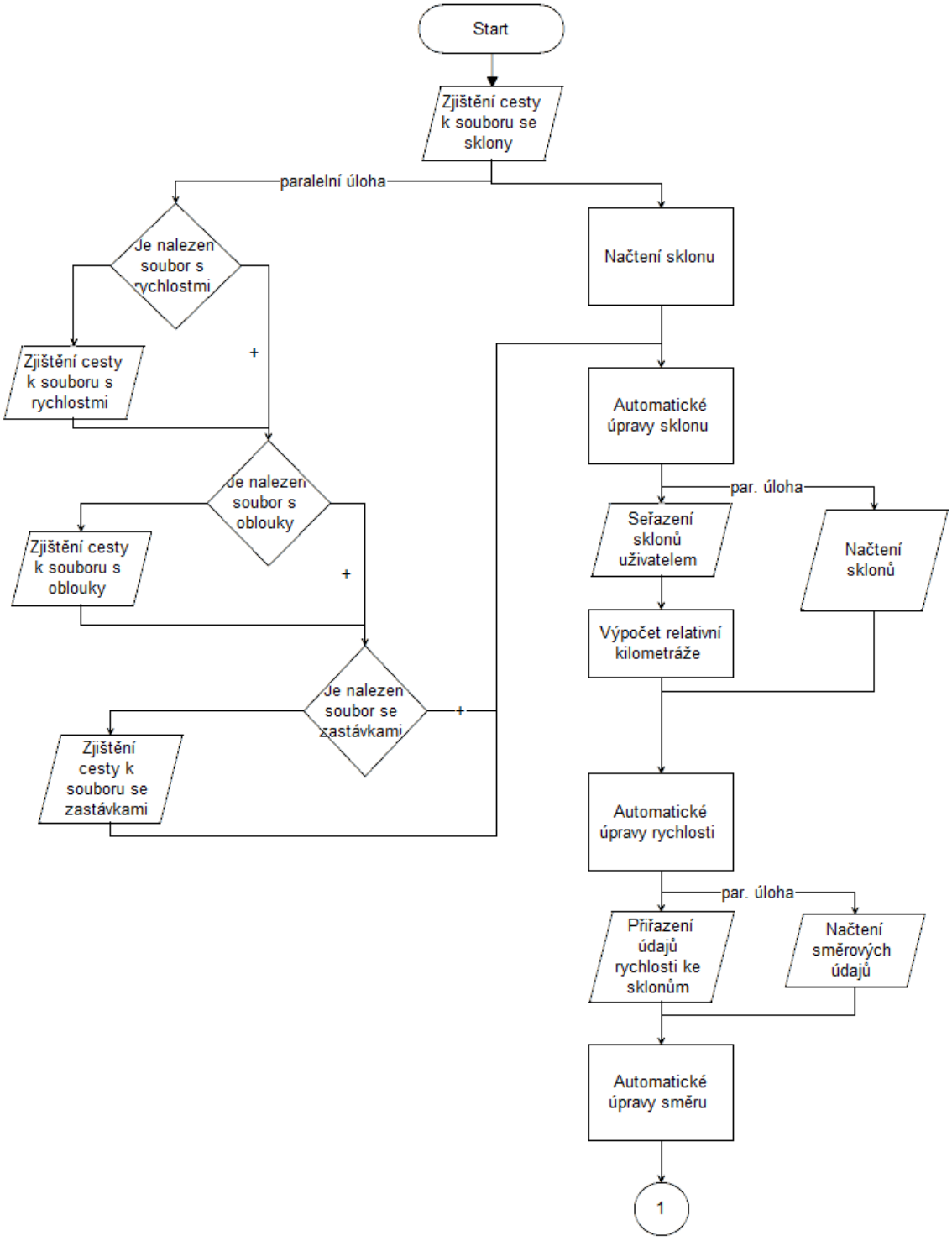
Příloha A

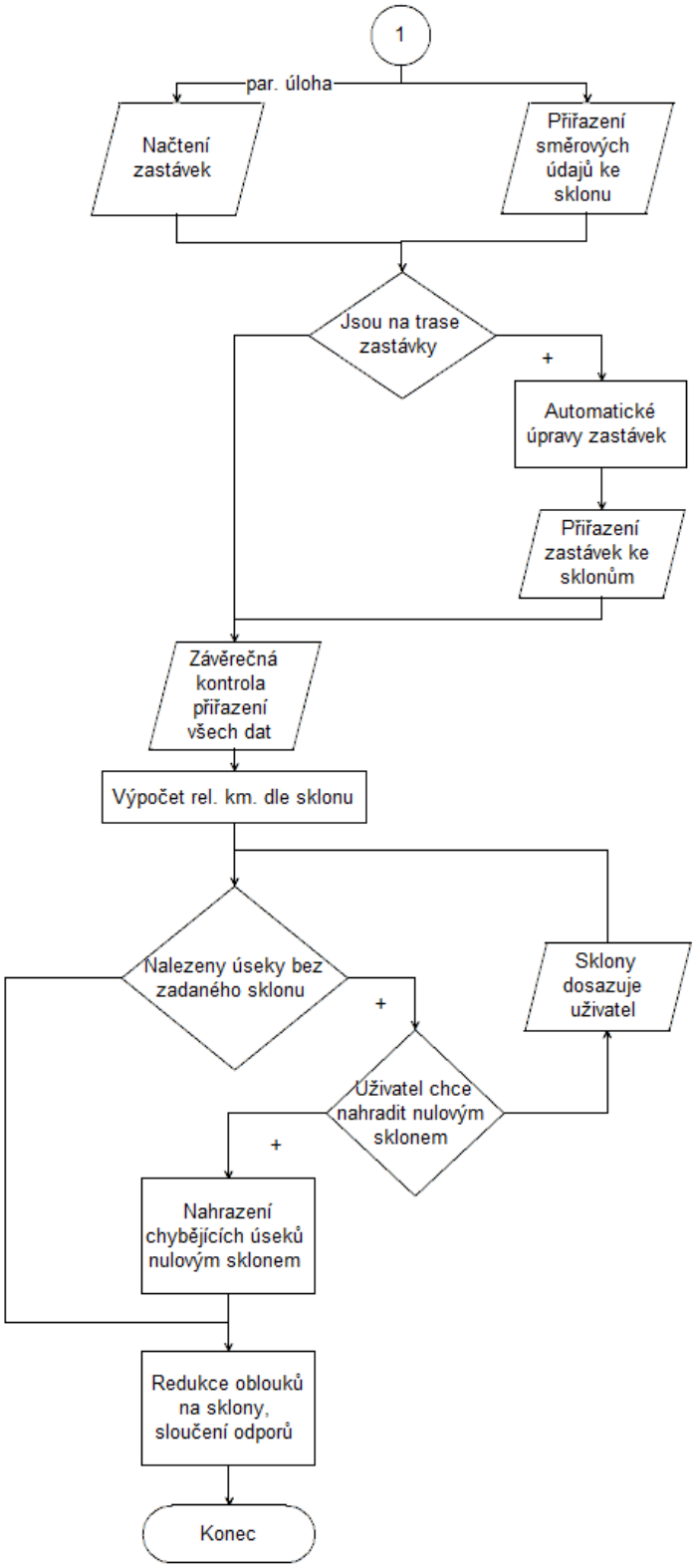
Seznam výchozích tras:

- Brno – Tišnov – Nedvědice – Žďár nad Sázavou,
- Česká Třebová – Rudoltice v Čechách – Lanškroun,
- Hradec Králové – Týniště nad Orlicí – Letohrad,
- Cheb – Františkovy Lázně – Hranice v Čechách,
- Liberec – Česká Lípa – Děčín – Ústí nad Labem,
- Litvínov – Louka u Litvínova – Teplice v Čechách,
- Most – Louka u Litvínova – Moldava v Krušných horách,
- Olomouc – Zábřeh na Moravě – Jeseník,
- Ostrava – Opava – Krnov,
- Pardubice – Jaroměř – Turnov – Liberec,
- Pardubice – Hradec Králové – Trutnov,
- Pardubice – Moravany – Holice – Borohrádek,
- Pardubice – Ústí nad Orlicí – Hanušovice,
- Plzeň – Chrást u Plzně – Radnice,
- Plzeň – Přovany – Bezdružice,
- Plzeň – Rokycany – Nezvěstice,
- Plzeň – Žatec – Chomutov,
- Plzeň – Železná Ruda,
- Praha – Karlštejn / Rudná u Prahy – Beroun – Březnice – Písek – České Budějovice,
- Tábor – Horní Cerekev – Jihlava.

Příloha B

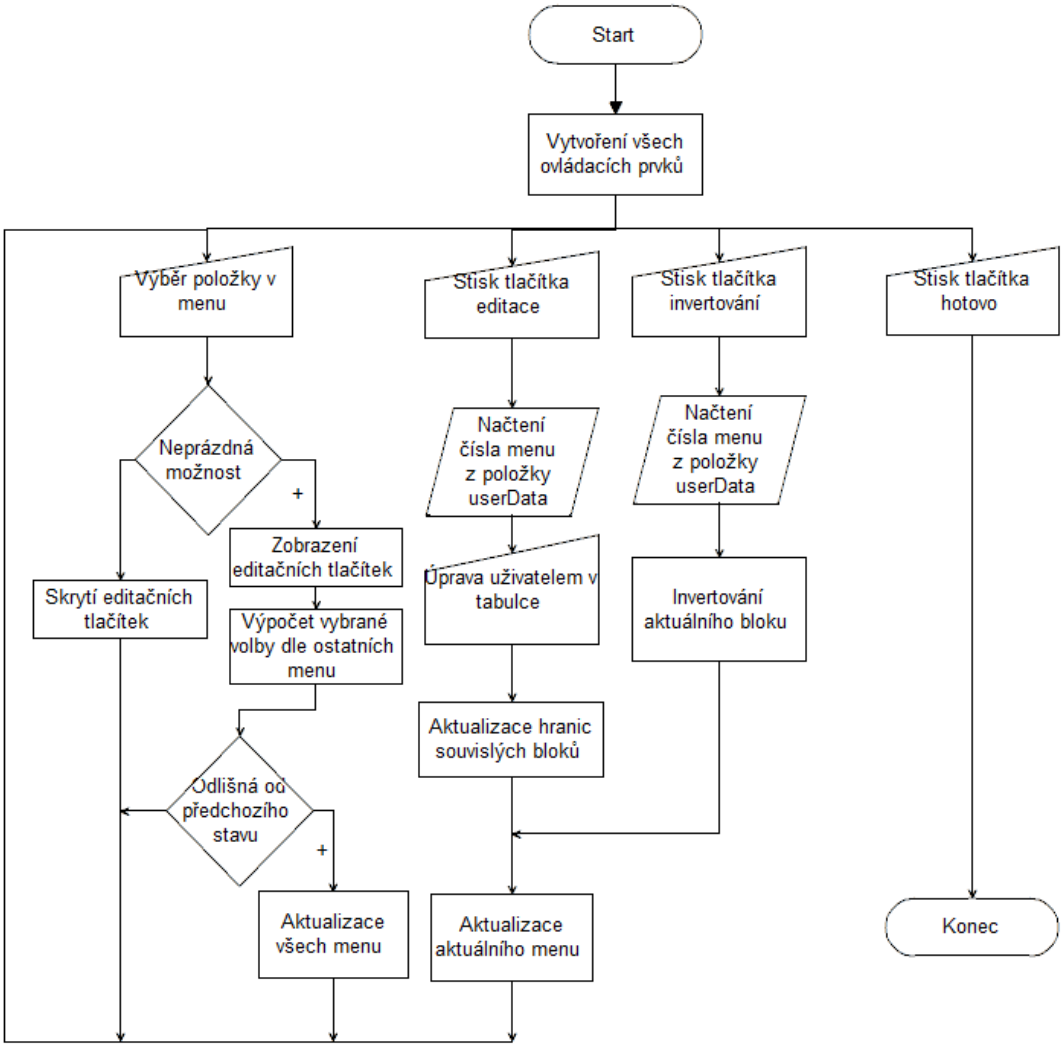
Vývojový diagram celého programu.




1
par. úloha
Načtení zastávek
Přiřazení směrových údajů ke sklonu
Jsou na trase zastávky
+
Automatické úpravy zastávek
Přiřazení zastávek ke sklonům
Závěrečná kontrola přiřazení všech dat
Výpočet rel. km. dle sklonu
Nalezeny úseky bez zadaného sklonu
+
Sklony dosazuje uživatel
Uživatel chce nahradit nulovým sklonem
+
Nahrazení chybějících úseků nulovým sklonem
Redukce oblouků na sklony, sloučení odporů
Konec

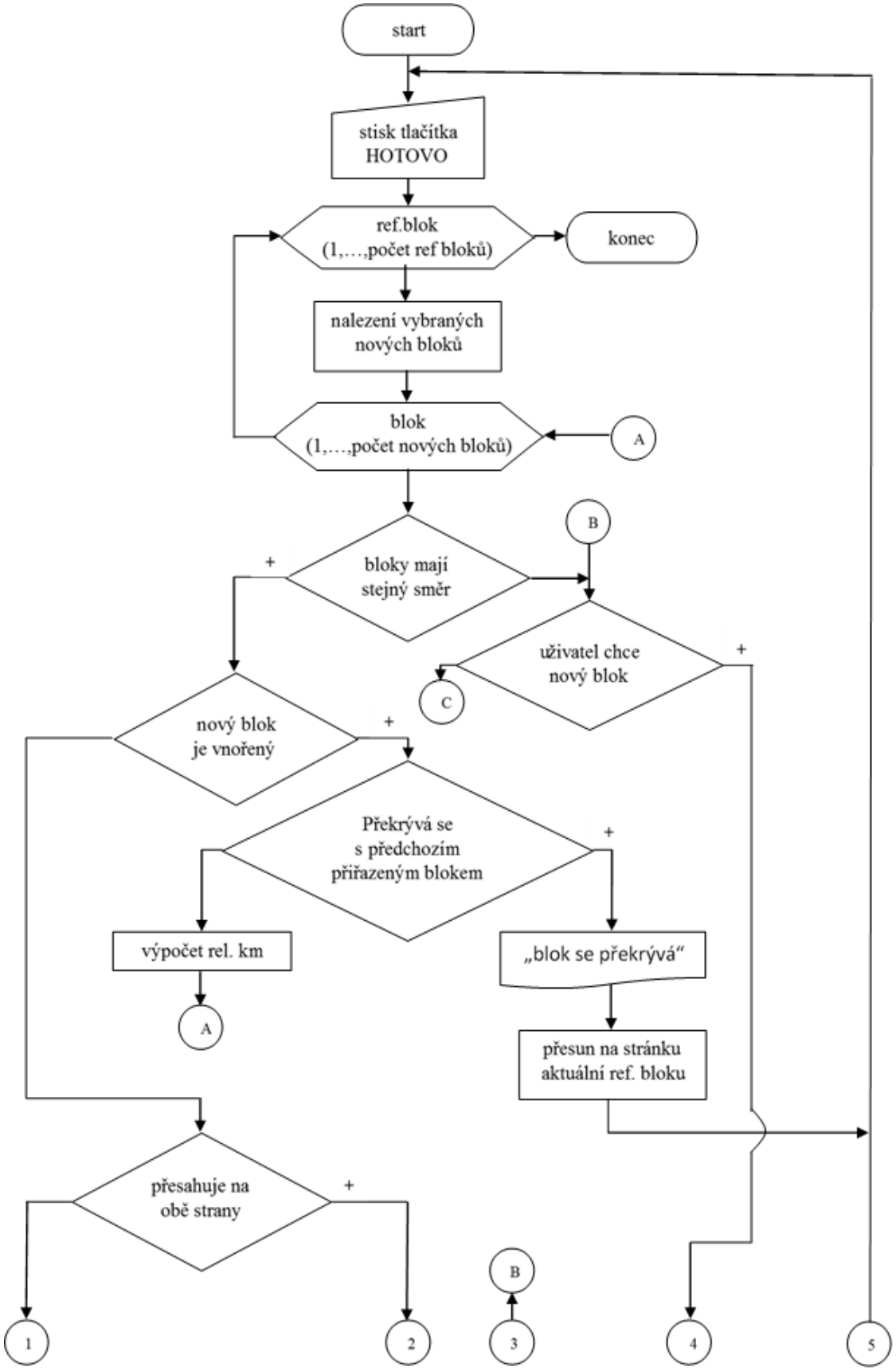
Příloha C

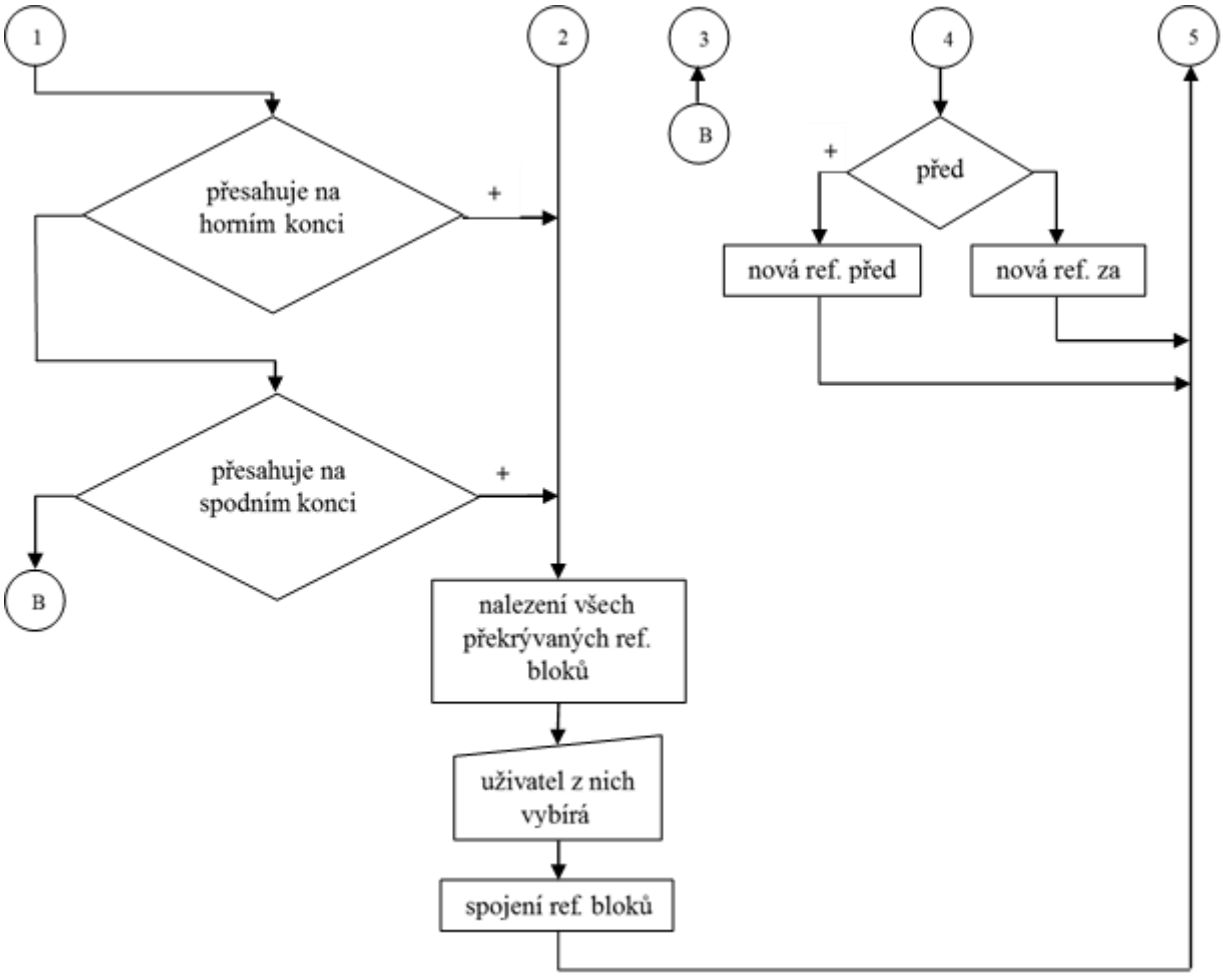
Vývojový diagram funkce *connectProfileByUser*.



Příloha D

Vývojový diagram vyhodnocení voleb ve funkci *connectByUser* – vnořená fce. *closeWindow*.





Příloha E

Obsah vloženého CD:

- soubory s funkcemi vytvořeného programu,
- soubory vstupních dat.

Seznam obrázků

Seznam grafů

Seznam tabulek

Seznam pramenů

[1] Federální ministerstvo dopravy. (1979). *Předpis ČSD - V7: Trakční výpočty.* Praha: Nakladatelství dopravy a spojů.

[2] Geotechnical Software Services. (Leden 2011). *C++ Programming Style Guidelines*, 4.9. Získáno Listopad 2013, z http://geosoft.no/development/cppstyle.html

[3] Mathworks, Inc. (1994 - 2015). *MATLAB documentation*. Získáno 2013 - 2015, z http://www.mathworks.com/help/

[4] Ministerstvo Dopravy ČR. (2009). *Slovník dopravní terminologie*. Získáno Listopad 2013, z http://www.slovnikdopravy.cz/

[5] Mlynařík, L. (2011). *Elektrická trakce 1.* Pardubice: Dopravní fakulta Jana Pernera.

[6] Stack Exchange, Inc. (2015). *Stack Overflow*. Získáno 2014 - 2015, z http://stackoverflow.com/

[7] SŽDC, s.o. (2014). *Jízdní řád 2015.* Praha: SŽDC, s.o.