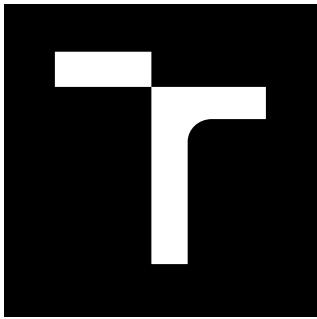


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky  
a komunikačních technologií

SEMESTRÁLNÍ PRÁCE



# VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

**FAKULTA ELEKTROTECHNIKY  
A KOMUNIKAČNÍCH TECHNOLOGIÍ**  
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

**ÚSTAV RADIOTELEKTRONIKY**  
DEPARTMENT OF RADIO ELECTRONICS

**TELEMETRICKÝ ARCHIV DRUŽIC**  
SATELLITE TELEMETRY ARCHIVE

**SEMESTRÁLNÍ PRÁCE**  
SEMESTRAL THESIS

**AUTOR PRÁCE** Bc. Péter Tóth  
AUTHOR

**VEDOUCÍ PRÁCE** Ing. Tomáš Urbanec, Ph.D.  
SUPERVISOR

**BRNO 2017**



## Semestrální práce

magisterský navazující studijní obor **Elektronika a sdělovací technika**

Ústav radioelektroniky

**Student:** Bc. Péter Tóth

**ID:** 100291

**Ročník:** 2

**Akademický rok:** 2017/18

### NÁZEV TÉMATU:

### Telemetrický archiv družic

### POKYNY PRO VYPRACOVÁNÍ:

MM2E: Seznamte se se strukturou vysílaných telemetrických dat PSK31 družic PSAT, BRICsat, PSAT-2. Realizujte odstranění dopplerova jevu a provedte demodulaci signálu z SDR IQ záznamu, vytvořte kostru archivu telemetrických dat a ověřte jeho funkčnost. K archivu vytvořte referenční údaje o poloze družice vůči Zemi a Slunci.

MMSE: Zpracujte celý archiv signálů družic PSAT, BRICsat, PSAT-2 a získaná data synchronizujte s referenčními údaji. Vytvořte vhodný nástroj pro zobrazení údajů z telemetrického archivu. Proveďte analýzu získaných dat všech dostupných veličin a jejich vzájemnou vazbu. Porovnejte získaná data z volně dostupných telemetrických archivů. Získejte maximum dat z obsahu transpondéru družic, doplňte o ně archiv a vyhodnoťte souvislosti s telemetrickými údaji. Vyhodnoťte získané údaje vzhledem k budoucím mísím.

### DOPORUČENÁ LITERATURA:

[1] Urbanec, T., Vágner, P., Kasal, M. P-sat Transponder WEB Specification [online]. 2015 [cit. 2017-10-5]. Dostupné z:

<http://www.urel.feec.vutbr.cz/esl/files/Projects/PSAT/P%20sat%20transponder%20WEB%20spec02.htm>

[2] Bruninga, B. PSAT - APRS plus a new PSK31 Approach [online]. 2017 [cit. 2017-10-5]. Dostupné z:

<http://aprs.org/psat.html>

**Termín zadání:** 18.9.2017

**Termín odevzdání:** 13.12.2017

**Vedoucí práce:** Ing. Tomáš Urbanec, Ph.D.

**Konzultant:**

**prof. Ing. Tomáš Kratochvíl, Ph.D.**  
*předseda oborové rady*

### UPOZORNĚNÍ:

Autor semestrální práce nesmí při vytváření semestrální práce porušit autorská práva třetích osob, zejména nesmí zasahovat nedovoleným způsobem do cizích autorských práv osobnostních a musí si být plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č.40/2009 Sb.

## **ABSTRAKT**

Táto práce se zaobírá zpracováním přijatých signálů z amatérských družic NO-83 a NO-84 ParkinsonSat na nízké oběžné dráze (Low Earth Orbit – LEO) vysílajících telemetrické údaje v pásmu 70 cm vln. které jsou postiženy dopplerovským posuvem kmitočtu. Kvůli povaze oběžné dráhy a kmitočtu vysílání, přijatý signál je znatelně poškozen dopplerovským posuvem kmitočtu, který se musí kompenzovat pro pozdější potřeby demodulace.

## **KLÍČOVÁ SLOVA**

korekce dopplerského posuvu, NO-83, BRICsat, NO-84, PSat, družice LEO, archiv dat TLE

## **ABSTRACT**

This project work is dealing with processing of received radio signals of LEO satellites NO-83 and NO-84 ParkinsonSat transmitting in the 70–centimeter band. The nature of this kind of setup makes the received signal bearing a large amount of Doppler shift, which needs to be compensated in order to demodulate it.

## **KEYWORDS**

doppler shift correction, NO-83, BRICsat, NO-84, PSat, LEO satellites, TLE data archive

TÓTH, Péter. *Název studentské práce*. Brno, 2017, 41 s. Semestrální projekt. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav radioelektroniky. Vedoucí práce: Ing. Tomáš Urbanec , Ph.D.

## PROHLÁŠENÍ

Prohlašuji, že svůj semestrální projekt na téma „Název studentské práce“ jsem vypracoval(a) samostatně pod vedením vedoucího semestrálního projektu a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor(ka) uvedeného semestrálního projektu dále prohlašuji, že v souvislosti s vytvořením tohoto semestrálního projektu jsem neporušil(a) autorská práva třetích osob, zejména jsem nezasáhl(a) nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom(a) následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno .....  
.....  
podpis autora(-ky)

## PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Ing. Tomášu Urbanci, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno .....  
.....  
podpis autora(-ky)

# OBSAH

<b>Úvod</b>	<b>10</b>
<b>1 Teorie</b>	<b>11</b>
1.1 Struční přehled amatérských družic . . . . .	11
1.2 Sledování a predikce pohybu těles na oběžné dráze Země . . . . .	11
1.3 Výpočet rychlosti těles na oběžní dráze Země . . . . .	12
1.4 Určení dopplerovského posuvu . . . . .	12
<b>2 Realizace skriptů v jazyce Python 3</b>	<b>14</b>
2.1 Vytvoření archivu dat TLE . . . . .	14
2.2 Korekce dopplerovského posuvu . . . . .	15
2.3 Tvorba spektrogramů . . . . .	16
2.4 Quo vadis moduly? . . . . .	17
<b>3 Prezentace výsledků</b>	<b>19</b>
<b>4 Závěr</b>	<b>23</b>
<b>Literatura</b>	<b>24</b>
<b>Seznam symbolů, veličin a zkratek</b>	<b>26</b>
<b>Seznam příloh</b>	<b>27</b>
<b>A Zdrojové kódy</b>	<b>28</b>
A.1 Skript TLE.py . . . . .	28
A.2 Skript get_undopplered.py . . . . .	34
A.3 Skript wav2spectrogram.py . . . . .	39

# SEZNAM OBRÁZKŮ

1.1	Výpočet vektoru relativní rychlosti [13] . . . . .	13
2.1	Vývojový diagram extrafování dat TLE . . . . .	15
2.2	Náčrt fungování modulu pro korekci dopplerovho posuvu . . . . .	18
3.1	Spektrogram signálu družice NO-84 bez korekce . . . . .	19
3.2	Spektrogram signálu družice NO-84 s korekcí . . . . .	20
3.3	Spektrogram signálu družice NO-83 bez korekce . . . . .	21
3.4	Spektrogram signálu družice NO-83 s korekcí . . . . .	22

## **SEZNAM TABULEK**

1.1	Tangenciální rychlosť pro rôzne obežné dráhy [14]	12
-----	---	----

# **SEZNAM VÝPISŮ**

```
/home/ptoth/Documents/Projekt/github/projekt17/Python3/TLE.py  . .  28
/home/ptoth/Documents/Projekt/github/projekt17/Python3/get_undopplered.py 34
/home/ptoth/Documents/Projekt/github/projekt17/Python3/wav2spectrogram.py 39
```

# ÚVOD

Umělé družice obíhající kolem Země jsou systémy, s kterými poslední fyzické kontakty lidí vznikají těsně před její vypuštěním. Prakticky to znamená praktickou nemožnost provádění jakékoli činnosti na zařízení v místě působení činnosti umělé družice. Proto hraje důležitou roli při provozování umělých družic dálkový sběr naměřených údajů senzorů, tzv. telemetrických dat, umělých družic za účelem vyhodnocení její stavu.

V této práci je vytvářena snaha o automatické zpracování telemetrických dat amatérských umělých družic Parkinson NO-83 a NO-84. Protože tyto družice mají svou dráhu na nízké oběžné dráze, je jejich signál silně postižen dopplerovským posuvem. Pro správní demodulaci signálu se tento posuv kmitočtu signálu musí korigovat, a kvůli číslicové povaze modulovaného signálu musí být táto korekce prováděna bez vzniku fázové diskontinuity.

Dělení dokumentu do kapitol a podkapitol je tvořen ze záměrem porozumění a sledování vzniku řešení zadaného a problému. Ke kompenzaci dopplerovského posuvu kmitočtu signálu musíme znát relativní rychlosť družice vzhledem ke pozemní stanici. Pro výpočet této rychlosti musíme znát naši polohu, predikovat pohyb umělé družice a tvar její dráhy. Stručný úvod do této problematiky na popsán v úseku 1.2. V následující úseku 1.3 se věnujeme výpočtu relativní rychlosti umělé družice vzhledem k pozemní stanici nutnou pro výpočet velikosti dopplerovského posuvu.

V kapitole 2 se prezentuje konkrétní řešení, včetně vývojových diagramů a popisu fungování jednotlivých skript v jazyce Python 3.

V příloze jsou uvedeny zdrojové kódy prezentovaných modulů.

# 1 TEORIE

## 1.1 Stručný přehled amatérských družic

Radiokomunikace prošla ohromným vývojem v průběhu druhé polovice 20. století. Vypuštění první umělé družice Sputnik-1 v roce 1957 datujeme začátek vesmírného věku. Od té doby se otevřeli nové možnosti radioamatérův věnovat se svému koníčku, nebo výzkumu.

Necelé čtyři roky po vypuštění první umělé družice Země se svět dočkal první amatérské umělé družice sestrojeného v rámci projektu Orbiting Satellite Carrying Amateur Radio (OSCAR) [1], kterého nástupnickou organizaci se stal Radio Amateur Satellite Corporation (AMSAT-NA) [2]. Družice OSCAR I byla vypuštěná na nízkou oběžnou dráhu jako druhotný náklad, který vyžíval rezervy nosnosti rakety Thor DM-21 Agena-B. Tento způsob dopravy byl zvolen z ekonomických důvodů a je dodnes používán.

Obecně se amatérské družice nasazují na nízkou oběžnou dráhu (Low Earth Orbit – LEO) [3]. Výhodou dráhy tohoto typu je jejich finanční nenáročnost, co je zčásti způsobená vlastností plynoucího z nazvu oběžné dráhy, výškou orbitu, který se pohybuje od 300 km do 2000 km. Nedostatkem nízké oběžné dráhy je vysoká relativní rychlosť družice vůči pozemní stanici, která přibližně  $7,8 \text{ km/s}$  [4] jehož důsledkem je rádiový signál značně postižen dopplerovským posuvem kmitočtu, kterého velikost dosahuje i 26 ppm.

## 1.2 Sledování a predikce pohybu těles na oběžné dráze Země

Objekty, které se pohybují vesmírem kolem Země jsou sledovány organizacemi [5]:

- United States Strategic Command (USSTRATCOM)(součást Department of Defense (DoD))
- European Space Agency (ESA)
- Fraunhofer-Institut fur Hochfrequenzphysik und Radartechnik (Fraunhofer-FHR)
- Jet Propulsion Laboratory (JPL)(součást National Aeronautics and Space Administration (NASA))
- Massachusetts Institute of Technology (MIT)
- European Incoherent Scatter Scientific Association (EISCAT)
- United States Air Force (USAF)

Ke sledování se používají pozemní radary, lidary, pozemní a vesmírné teleskopy. Mezi sledované objekty patří umělé družice, pozůstatky raket a jiný vesmírný odpad. Nejrozsáhlejší katalog stavu družic udržuje Ministerstvo obrany Spojených států (DoD) s názvem Space Object Catalog. Civilní varianta této databáze je provozována organizací NASA. Tyto databáze se udržují pomocí různých modelů orbitální mechaniky. Pohyby družic jsou analyticky vypočteny pomocí teorie všeobecných perturbací. Prvky dráhy této teorie jsou publikovány ve formátu NASA/NORAD two-line elements (TLE). [6].

Proč se zabývat přesnou polohou družice? Aby jsme byly schopní provést korekci dopplerovského posuvu, musíme splnit několik požadavek výpočtu:

- polohu pozemní stanice <sup>1</sup>
- polohu a rychlosť umělé družice

### 1.3 Výpočet rychlosti těles na oběžní dráze Země

Rychlosť těles na oběžné dráze lze precizně vypočítat pomocí vztahu [14]:

$$v = \sqrt{\mu - \left( \frac{2}{r} - \frac{1}{a} \right)} \quad (1.1)$$

kde  $\mu$  je standardní gravitační parametr,  $v$  je rychlosť tělesa,  $r$  je vzdálenost tělesa od místa pozorování,  $a$  je délka hlavní poloosy.

orbita [–]	výška orbitu [km]	rychlosť orbitu [km/s]
LEO	200 – 2000	6,9 – 7,8 pro kruhovou dráhu
Molnija	500 – 39900	6,5 – 8,2 pro eliptickou dráhu
Geostacionární	35768	0,97 – 1,08

Tab. 1.1: Tangenciální rychlosť pro různé orbity [14]

### 1.4 Určení dopplerovského posuvu

Pozorovatel, který je v relativním pohybu vzhledem ke zdroji vlnění, bude vnímat vlnění se změněným kmitočtem. V případě, že relativní rychlosť se mění časem, mění se i velikost změny kmitočtu. Tuto změnu kmitočtu nazýváme dopplerovským posuvem a jev se nazývá Dopplerův jev, který byl publikovaný Christianem Dopplerem v roce 1842 v Praze.

---

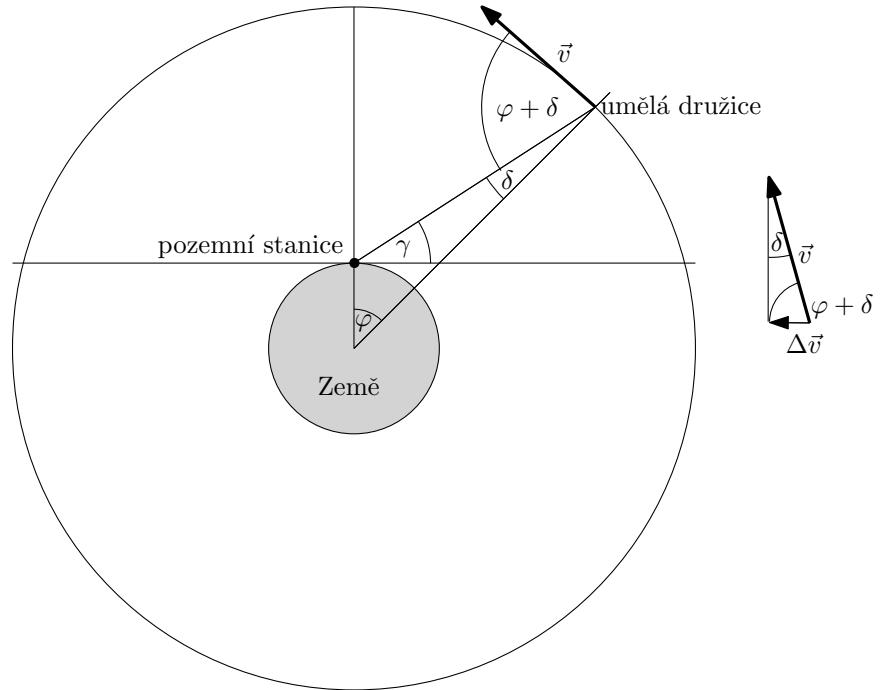
<sup>1</sup>pozemní stanici považujeme za stacionární vzhledem ke povrchu Země

Velikost dopplerovského posuvu lze přibližně určit pomocí vztahu [15]:

$$\Delta f = \frac{\Delta v}{c} f_0 \quad (1.2)$$

kde  $\Delta f$  je posuv kmitočtu,  $\Delta v$  je relativní rychlosť,  $c$  rychlosť vln v prostredí,  $f_0$  je pôvodný kmitočet.

Pro výpočet dopplerovského posuvu signálu vysílaného družicí na oběžné dráze Země je potrebné znáť její relativnú rychlosť vzhľadom k pozorovateľovi na pozemnej stanici. Túto rychlosť lze vypočítať z oběžnej rychlosťi družice pomocí trigonometrických výpočtov. Princip je zobrazen na obrázku 1.1



Obr. 1.1: Výpočet vektoru relatívnej rychlosťi [13]

## 2 REALIZACE SKRIPTŮ V JAZYCE PYTHON 3

### 2.1 Vytvoření archivu dat TLE

Vytvoření lokální databázi dat TLE je nezbytné pro správnou korekci dopplerovského posuvu. Pro umělé družice NO-83 a NO-84 jsou veřejně přístupné data TLE na stránce organizace AMSAT-NA: <<http://amsat.org/pipermail/keps/>>. Tyto data jsou distribuované formou elektronického mailing listu, kterého archív se nachází na výše zmíněném URL v jedním souboru. Aby bylo možné použít údaje TLE obsahnutý v tomhle archivním souboru, je nutné provést extrakci dat TLE dle datu jejího vzniku.

Na obrázku 2.1 je uveden vývojový diagram skriptu pro extrahování dat TLE. Jako povinný vstupní parametr je název souboru staženého z webové stránky organizace AMSAT-NA. Data TLE jsou posílané v jednom emailu. Skript identifikuje začátek i konec balíků dat TLE. Na začátku každého balíku se hledají vzory:

```
SB\s+KEPS\s+@\s+AMSAT\s+$ORB\d5\. [A-Z]
```

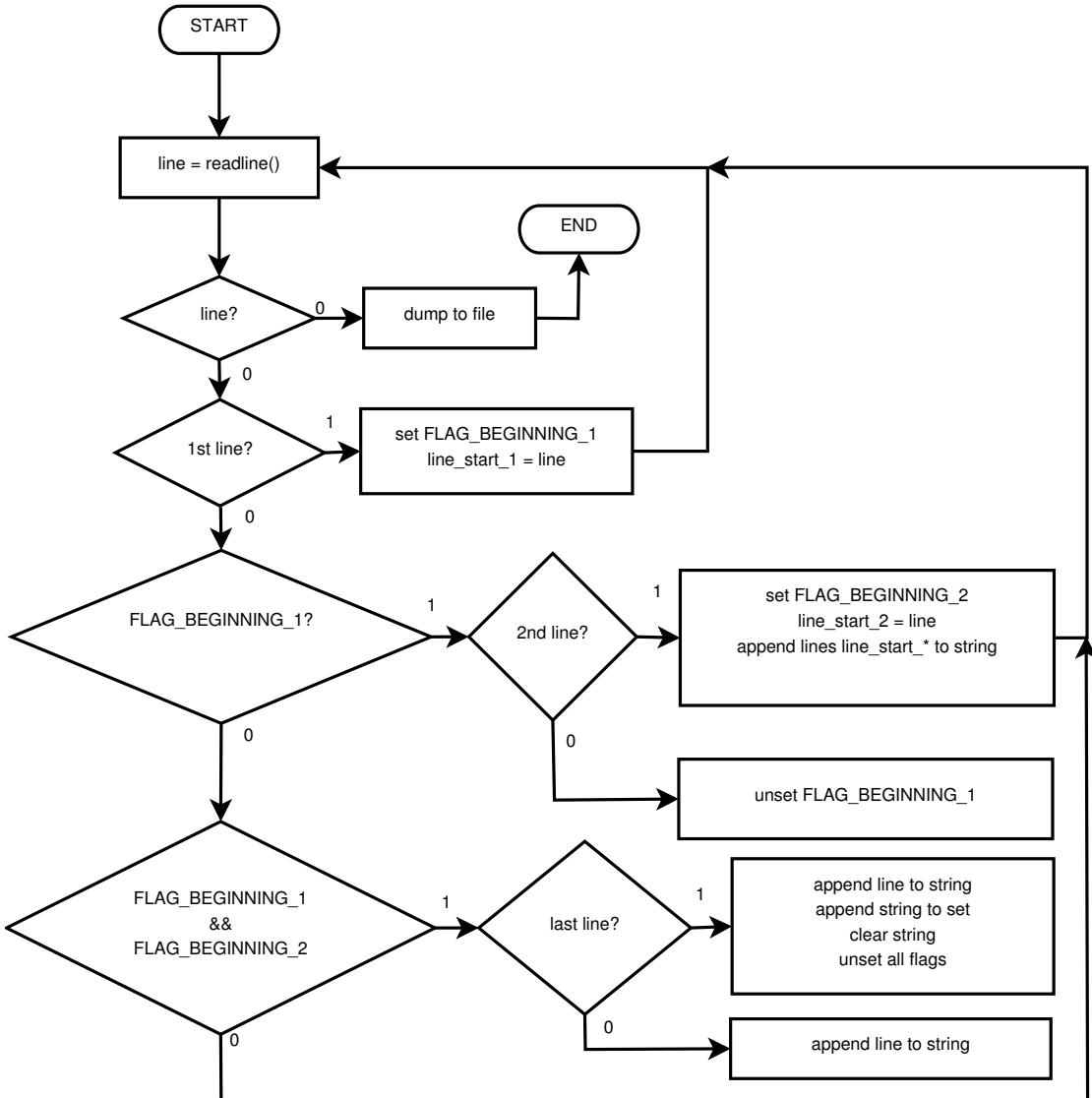
```
^2Line
```

pomocí nástroje na vyhledávání regulárních výrazů implementovaného modulem re ze standardní knihovny jazyka Python. Konec bálíka je značen řetězcem \EX.

V případě, že skript narazí na hledaný výraz, nastaví se příslušný příznak. Tyto příznaky zaručí, aby řádky jednoho balíku se připojili k jednomu řetězci. Když skript identifikuje poslední řádek, pospojovaný řetězec řádků se připojí k množině všech balíků TLE dat. Návratovou hodnotou je právě tato množina. Množina v jazyce Python 3 zaručuje jedinečnost všech prvků, obdobně jako množiny z teorie množin.

K výpisu souboru patří funkce `dump_to_file` skriptu `TLE.py`. Tato funkce má jeden povinný parametr, množinu dat TLE. Z této množiny se načte každý jeden prvek. V těchto prvcích se prohledává datum vzniku, což se použije jako název souboru. Následně se dle potřeby vytvoří adresář, kterého název je rok vzniku TLE dat. Před vytvářením adresáře se kontroluje přítomnost adresáře, nebo souboru stejného jména, jaký se chystá být vytvořit. Jestli adresář se stejným jménem existuje, začne se de ní zapisovat. V opačném případě předpokládáme, že v adresáři je soubor se stejným jménem musíme název složky, kterou se chystáme vytvořit změnit z důvodu nemožnosti koexistence souboru a adresáře stejného jména v rodičovském adresáři. K názvu složky se přidává znak podtržítka a číslo, které se iteruje inkrementací do doby, kdy v rodičovském se nebude nacházet soubor se stejným jménem. V případě adresáře se stejným jménem nalezeného po přidání dodatečných znaků k názvu souboru, se adresář začne používat k uložení souborů. Čtení z množiny je realizováno

pomocí iterací nad její prvky.



Obr. 2.1: Vývojový diagram extrahování dat TLE

## 2.2 Korekce dopplerovského posuvu

Po úspěšném získání nutných souborů s daty TLE pro umělé družice našeho zájmu, můžeme přistoupit ke korekci dopplerovského posuvu signálu, kterou provedeme pomocí programu doppler volně dostupného z repozitáře GIT týmu codehub dostupného na URL <https://github.com/cubehub>.

Program doppler je utilitou příkazového řádku, který ze standardního vstupu čte IQ (in-phase, quadrature) data a zpracovává dle zadaných parametrů. Rozlišujeme dva režimy korekce frekvenčního posuvu. V režimu *const* se kompenzuje konstantní

posuv kmitočtu, kým v režimu *track* se provádí korekce sledováním pohybu umělé družice a to i dodatečně v případě zpracování předem zaznamenaných dat IQ. V pozdním případě se programu doppler musí předát argument datu a času záznamu ve formátu ISO 8601 [8] [9] bez udání časového posuvu v čase UTC.

Korekce dopplerovského posuvu se provádí programem doppler za pomoci volně dostupné knihovny libgpredict, který je založen na predikčním kódu programu Gpredict. [10]

Aby se zjednodušilo zpracování velikého množství souborů, byl vytvořen skript v jazyce Python 3 pro automatické zpracování záznamů s IQ daty. Modul get\_undopplred má definovanou funkci undoppler\_it, který jako vstupní parametry má:

- název souboru IQ dat
- název družice v notaci OSCAR<sup>1</sup>
- kmitočet na kterém je z družice vysíláno
- adresář s IQ daty
- adresář s TLE daty
- lokace pozemní stanice

Pomocí těchto údajů se sestrojí řetězec obsahující příkaz shellu BASH, kde jednotlivé příkazy jsou řazeny do tzv. kolony. Jde o zřetězení příkazů oddělených metaznakem svislá čára '|'. Standardní výstup příkazu se předává standardnímu příkazu následujícímu. [11]

Parametry záznamu IQ dat se zjišťují pomocí modulu pymediainfo, který je tzv. wrapper function knihovny Mediainfo [12]. Pomocí tohoto modelu lze zjistit kmitočet vzorkování, bitovou hloubku, kódování, počet kanálů, kodek.

Aby jsme mohli soubory formátu Waveform Audio File Format (WAVE) použít jako vstupní data pro program doppler, je nutné provést změnu formátu dle očekávání programu. K této úloze se použije volný program Sound eXchange (SoX). Podobně na výstupu lze použít SoX pro převod z RAW Audio formátu na WAVE.

## 2.3 Tvorba spektrogramů

Kontrola správnosti korekci dopplerovského posuvu lze hrubě odhadnout pomocí spektrogramu korigovaných IQ dat. Pro automatizovaní tohoto procesu slouží modul wav2spectrogram.

Hlavní částí tohoto modulu je funkce IQ\_to\_spectrogram. Python 3 modul pro tvorbu spektrogramů. Jediným argumentem je název souboru IQ dat. Ostatní parametry jsou buď napevno dány, nebo automaticky zjištění ze souboru IQ dat. Spektrogram je uložen ve formátu Portable Network Graphics (PNG).

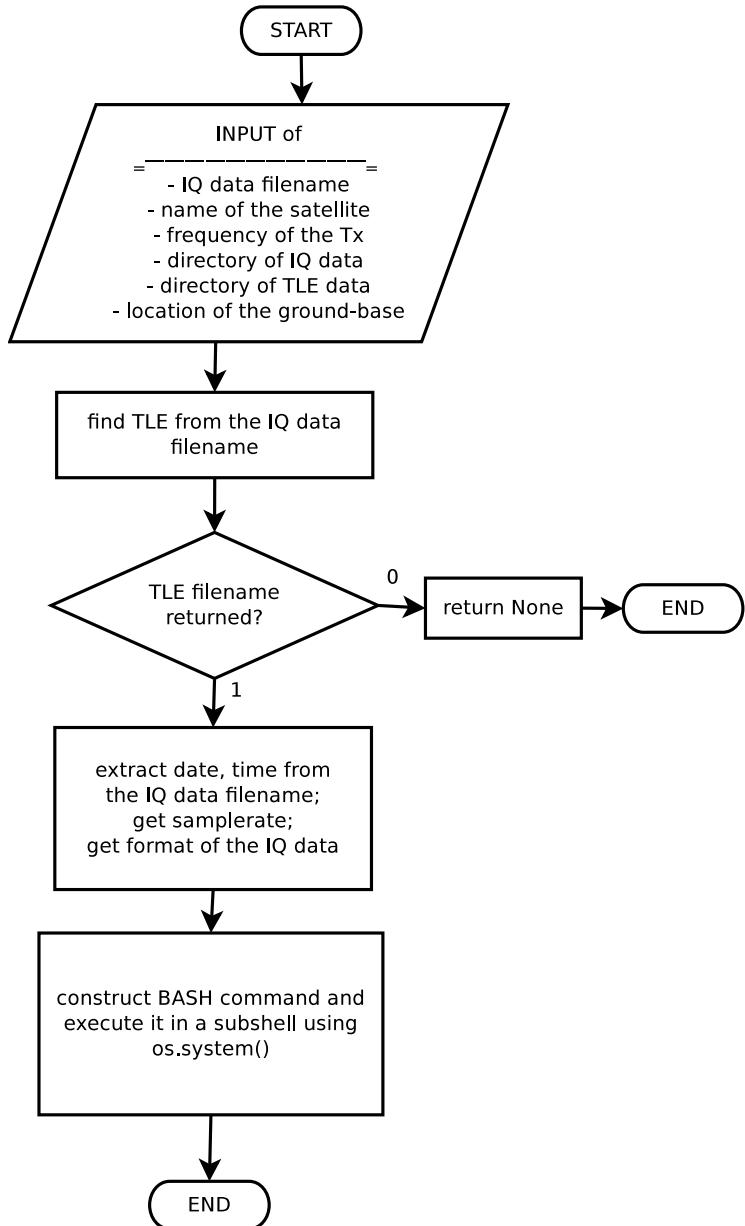
---

<sup>1</sup>v našém případě se jedná o NO-83, NO-84

## 2.4 Quo vadis moduly?

Kudy směrují tyto moduly? Je filozofická otázka, na kterou by se dalo napsat nespočetné množství stran. Lze říct, že sami o sobě tyto moduly, skripty, neprovádějí automatické zpracování signálů, avšak jako základ pro budoucí vývoj mohou být nápomocné pro svou vlastnost zjednodušeného volání jinou funkcí, skriptem. Jazyk Python umožňuje jednoduchou iteraci nad všemi soubory složky, což nám dává možnost dávkového zpracování souborů s IQ daty.

Do budoucna se počítá s dalším zpracováním frekvenčně zkorigovaných signálů – demodulace FM a následné dekódování přijatých zpráv telemetrie PSK31.



Obr. 2.2: Náčrt fungování modulu pro korekci dopplerovho posuvu

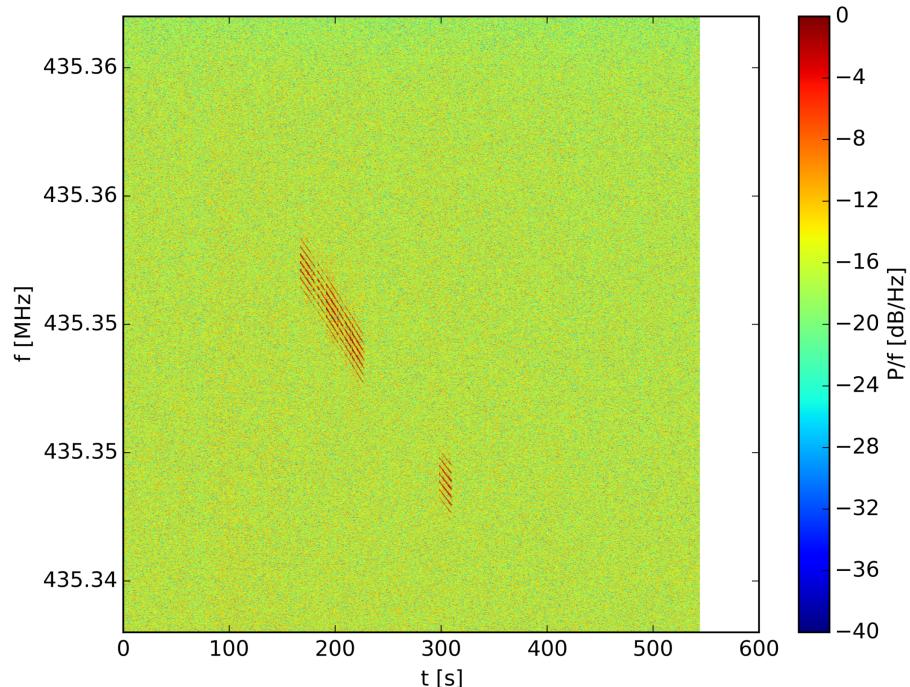
### 3 PREZENTACE VÝSLEDKŮ

Funkčnost skript bylo ověřeno na záznamech signálů umělých družic NO-83, a NO-84.

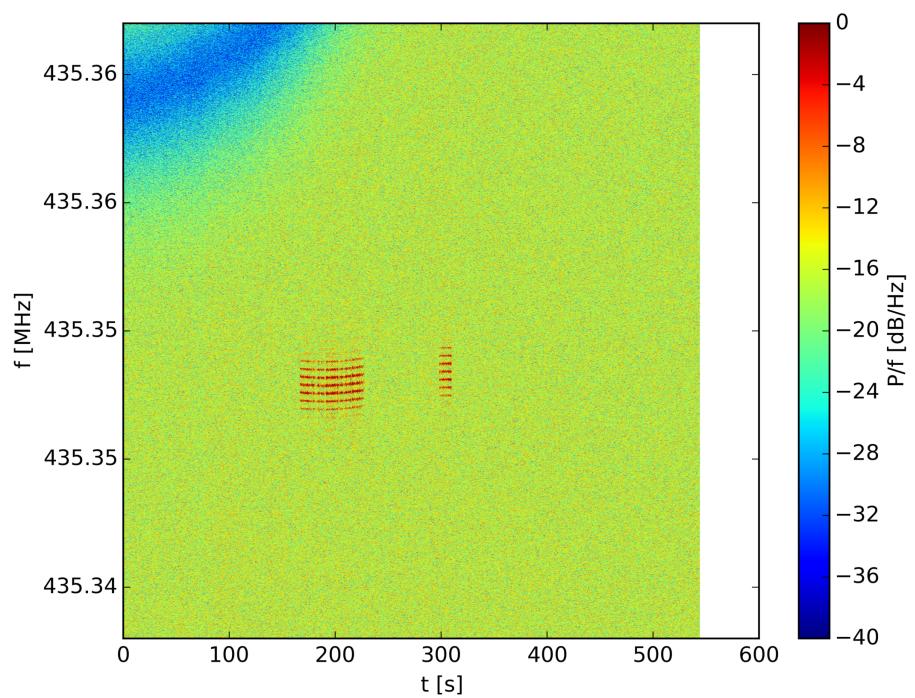
Na obrázku 3.1 je znázorněn spektrogram přijatého rádiového signálu umělé družice NO-84. Dopplerův efekt lze zpozorovat na základě zjevného klesání kmitočtu přijatého signálu v závislosti na čase. Za předpokladu konstantního kmitočtu nosné vysílače a kmitočtu lokálního oscilátoru směšovače radiového přijímače můžeme usoudit, že umělá družice se k nám přibližuje a její relativní rychlosť vzhledem ke místu příjmu se zmenšuje.

Po aplikování korekci dopplerovského posuvu je spektrogram signálu znázorněn na obrázku 3.2. Vidno, že drift kmitočtu od ideálního stavu přímky je minimální. Tyto odchylinky jsou dány:

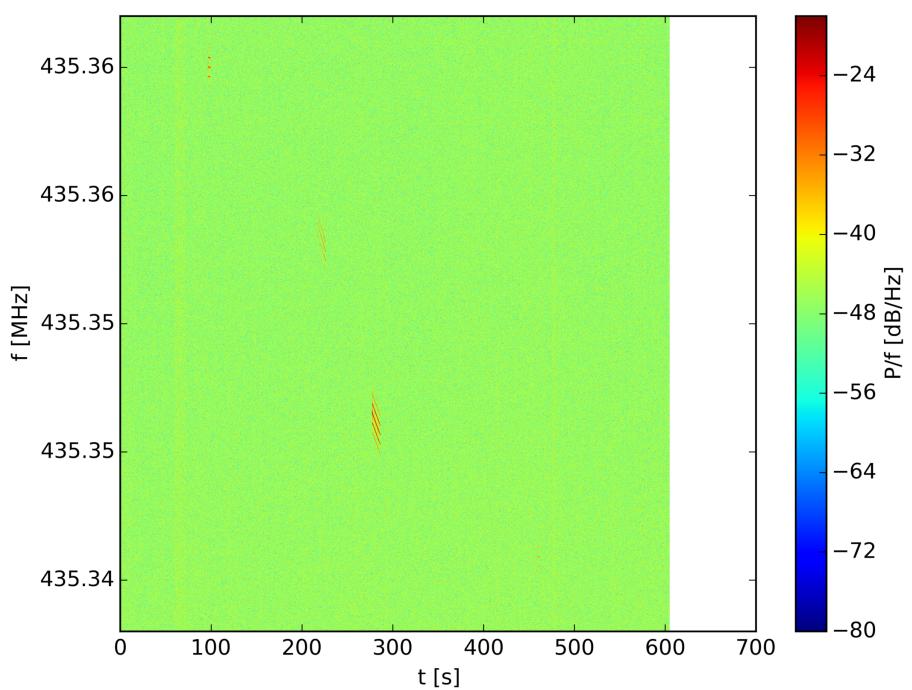
- nepřesnost lokálního oscilátoru družice
- korekce kmitočtového posuvu po částech signálu diskrétní časové délky
- nepřesnost modelu SGP4 při předvídání pohybu družice



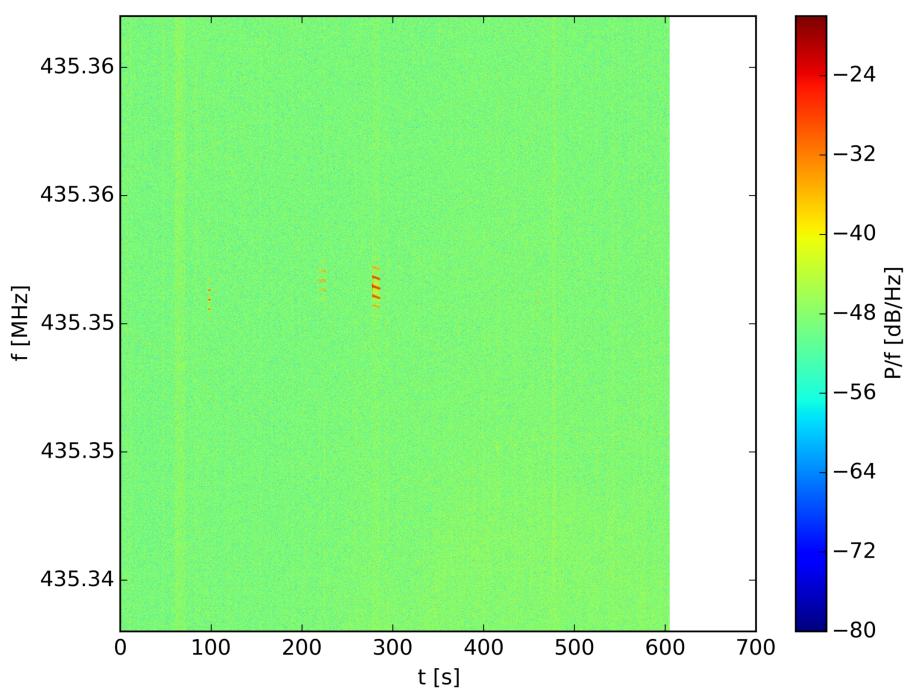
Obr. 3.1: Spektrogram signálu družice NO-84 bez korekce



Obr. 3.2: Spekrogram signálu družice NO-84 s korekcí



Obr. 3.3: Spektrogram signálu družice NO-83 bez korekce



Obr. 3.4: Spektrogram signálu družice NO-83 s korekcí

## 4 ZÁVĚR

Dopplerův jev je součástí našeho každodenního života, a to nejen v případě když vedle nás na železniční projede vysokorychlostní vlaková souprava dávající kontinuální výstražní signál, ale i v případě, že na ni sedíme a snažíme se naším mobilním komunikačním prostředkem provést datovou komunikaci prostřednictvím radiových signálů, avšak musí se poznamenat, že i rychlosť nejrychlejšího vlaku je zanedbatelná vzhledem ke rychlosti družice na nízké oběžné dráze<sup>1</sup>. Nejenom vysoká rychlosť pohybu umělé družice, ale i použitý kmitočet z pásmá 70 cm vln nás vede ke nutnosti korigování vzniklého dopplerovského posuvu kmitočtu signálu.

Táto část zadání byla provedená bez jakýkoli zjištěné chyby. Jako ukazují spektrogramy v kapitole 3 (obrázky 3.1 až 3.4), korekce dopplerovského posuvu je do značné míry úspěšně provedeno. Cesta vývoje telemetrického archivu však žádném případě tady nekončí, a před námi je ještě několik problémů, které se musí zdolat.

Pokračováním projektu je demodulace a vytvoření databáze. Obdobně jako dosavadní práci, i další části projektu jsou plánovány být implementovány pomocí volného softwaru jako GNU Radio a SQLite, který pod licencí public domain [17].

---

<sup>1</sup>viz. tabulka 1.1, nejrychlejší vlak pro osobní dopravu dle wikipedie dosahuje rychlosť  $603 \text{ km/s} = 0,1675 \text{ m/s}$ [16]

## LITERATURA

- [1] *Amateur radio satellite*. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2017-12-10]. Dostupné z URL: <[https://en.wikipedia.org/wiki/Amateur\\_radio\\_satellite](https://en.wikipedia.org/wiki/Amateur_radio_satellite)>.
- [2] *Amateur radio satellite*. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2017-12-10]. Dostupné z URL: <<https://en.wikipedia.org/wiki/AMSAT>>.
- [3] PUBLISHED BY AMERICAN RADIO RELAY LEAGUE. *The ARRL handbook for radio communications 2011*. 88th ed. Newington, CT: American Radio Relay League, 2010. ISBN 9780872590953.
- [4] *Low Earth orbit*. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2017-12-10]. Dostupné z URL: <[https://en.wikipedia.org/wiki/Low\\_Earth\\_orbit](https://en.wikipedia.org/wiki/Low_Earth_orbit)>.
- [5] *Space debris*. In: Wikipedia: the free encyclopedia [online]. Dostupné z URL: <[https://en.wikipedia.org/wiki/Space\\_debris](https://en.wikipedia.org/wiki/Space_debris)>.
- [6] *United States Space Surveillance Network*. In: Wikipedia: the free encyclopedia [online]. Dostupné z URL: <[https://en.wikipedia.org/wiki/United\\_States\\_Space\\_Surveillance\\_Network](https://en.wikipedia.org/wiki/United_States_Space_Surveillance_Network)>.
- [7] *Two-line element set*. In: Wikipedia: the free encyclopedia [online]. Dostupné z URL: <[https://en.wikipedia.org/wiki/Two-line\\_element\\_set](https://en.wikipedia.org/wiki/Two-line_element_set)>.
- [8] *ISO 8601*. In: Wikipedia: the free encyclopedia [online]. Dostupné z URL: <[https://en.wikipedia.org/wiki/ISO\\_8601](https://en.wikipedia.org/wiki/ISO_8601)>.
- [9] *Doppler documentation*. [online]. Dostupné z URL: <<https://github.com/cubehub/doppler>>.
- [10] *Libgpredict documentation*. [online]. Dostupné z URL: <<https://github.com/cubehub/libgpredict>>.
- [11] BRANDEJS, Michal. *UNIX - Linux: praktický průvodce*. Praha: Grada, 1996. ISBN 80-7169-170-4.
- [12] *Pymedainfo github repository*. [online]. Dostupné z URL: <<https://github.com/sbraz/pymedainfo>>.

- [13] Geroge P. Ah-Thew *Doppler compensation for LEO satellite communication systems*. [online] <<https://macsphere.mcmaster.ca/bitstream/11375/5713/1/fulltext.pdf>>.
- [14] *Orbital speed*. In: Wikipedia: the free encyclopedia [online]. Dostupné z URL: <[https://en.wikipedia.org/wiki/Orbital\\_speed](https://en.wikipedia.org/wiki/Orbital_speed)>.
- [15] *Orbital speed*. In: Wikipedia: the free encyclopedia [online]. Dostupné z URL: <[https://en.wikipedia.org/wiki/Doppler\\_effect](https://en.wikipedia.org/wiki/Doppler_effect)>.
- [16] *Land speed record for rail vehicles*. In: Wikipedia: the free encyclopedia [online]. Dostupné z URL: <[https://en.wikipedia.org/wiki/Land\\_speed\\_record\\_for\\_rail\\_vehicles](https://en.wikipedia.org/wiki/Land_speed_record_for_rail_vehicles)>.
- [17] *SQLite*. In: Wikipedia: the free encyclopedia [online]. Dostupné z URL: <<https://en.wikipedia.org/wiki/SQLite>>.

## **SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK**

AMSAT-NA	Radio Amateur Satellite Corporation
OSCAR	Orbiting Satellite Carrying Amateur Radio
USSTRATCOM	United States Strategic Command
DoD	Department of Defense
ESA	European Space Agency
Fraunhofer-FHR	Fraunhofer-Institut fur Hochfrequenzphysik und Radartechnik
TIRA	Tracking & Imaging Radar
NASA	National Aeronautics and Space Administration
JPL	Jet Propulsion Laboratory
GDSCC	Goldstone Deep Space Communications Complex
MIT	Massachusetts Institute of Technology
EISCAT	European Incoherent Scatter Scientific Association
USAF	United States Air Force
TLE	two-line elements
SGP4	Simplified perturbations models
UTC	Coordinated Universal Time
WAVE	Waveform Audio File Format
SoX	Sound eXchange
PNG	Portable Network Graphics

# SEZNAM PŘÍLOH

<b>A Zdrojové kódy</b>	<b>28</b>
A.1 Skript TLE.py . . . . .	28
A.2 Skript get_undopplered.py . . . . .	34
A.3 Skript wav2spectrogram.py . . . . .	39

# A ZDROJOVÉ KÓDY

## A.1 Skript TLE.py

Python 3 modul pro extrahování TLE dat.

```
1 #!/bin/python3
2
3 import re
4 import datetime
5 import os
6
7 #SB KEPS @ AMSAT $ORB06243.N
8 #2Line Orbital Elements 06243.AMSAT
9 PATTERN_START_1 = re.compile(r'SB\s+KEPS\s+@\s+AMSAT\s+\$ORB\d{5}\.[A-Z]')
10 PATTERN_START_2 = re.compile(r'^2Line')
11 #1 28897U 05043H 06249.26374724 .00000138 00000-0
12     38823-4 0 1152
13 #2 28897 098.1525 146.2174 0016539 282.7486 077.1854
14     14.59597696 37959
15 #/EX
16 PATTERN_END      = re.compile(r'^\//EX')
17 #FROM WA5QGD FORT WORTH, TX August 31, 2006
18 PATTERN_DATE     = re.compile(r'(January|February|March|
19     April|May|June|July|August|September|October|November|
20     December)\s+(\d{1}|\d{2})\,\s+(\d{4})') # Patter to
21     search the date inside the extracted TLE chunk. The
22     regular expresion is split to three groups as the
23     date is written in the US format: group 1: Month;
24     group 2: day (one or two digits); group 3: year (four
25     digits)
26 FLAG_BEGINNING_1 = 1 << 0
27 FLAG_BEGINNING_2 = 1 << 1
28 FLAG_ENDING      = 1 << 2
29
30
31 # Setting flags on a register passed
32 # register is the register to modify
```

```

24 # multiple flags can be passed
25 def set_flag(register, *flag):
26     """set_flag(register, flag_1, flag_2, ... )
27
28     Setting one or more flags in a register passed as an argument.
29     If no flag is passed the register is not going to be modified.
30 """
31     if flag:
32         for f in flag:
33             register |= f
34     return register
35 else:
36 # An empty list has been passed
37     return register
38
39 # Unsetting flags on a register passed
40 # register is the register to modify
41 # multiple flags can be passed
42 def unset_flag(register, *flag):
43     """unset_flag(register, flag_1, flag_2, ... )
44
45     Unsetting one or more flags in a register passed as an argument.
46     For multiple flags passed they all need to be unset in the registry
47     in order to achieve True to be returned.
48     If no flag is passed the register is not going to be modified and
49     it is being returned as it was passed.
50 """
51     if flag:
52         for f in flag:
53             register &= ~f
54     return register
55 else:
56 # An empty list has been passed.
57     return register

```

```

58
59 # Checking flags on a register passed
60 # register is the register to be checked.
61 # Multiple flags can be passed
62 def check_flag(register, *flag):
63     """check_flag(register, flag_1, flag_2, ... )
64
65     Checking one or more flags in a register passed as an
       argument.
66     If no flag is passed the register 0 is being returned
       .
67     """
68     flags = 0
69
70     if flag:
71         for f in flag:
72             flags |= f
73     return bool((register & flags) == (register))
74 else:
75     return 0
76
77 # Checking if the passed line matches the passed pattern
       .
78 def check_line(pattern, line_to_check):
79     """check_line(pattern, line_to_check)
80
81     Checking if the passed line matched the REGEX pattern.
82     """
83     return bool(re.search(pattern, line_to_check))
84
85 # Appending the line to a string.
86 def append_line(string, *lines_to_append):
87     """append_line(string, *lines_to_append)
88
89     Appending lines passed to the function to the string
90     """
91     for line in lines_to_append:
92         string += line
93

```

```

94     return string
95
96 # Dumping a set of TLEs files
97 def dump_to_file(set_of_TLEs, directory='../../keps/'):
98     '''dump_to_file(set_of_TLEs,[directory='../../keps/'])'
99
100    The function takes a set of TLEs extracted from the
101       AMSAT-e-mail list.
102    Each member of the set is written to a file, which name
103       contains the
104    date of the TLE file. In case of multiple files per day
105       , the last one
106    will be written out overwriting the files written
107       before.
108    The function checks the directory if it is available at
109       the CWD. If it
110       exists, than the file is written there. If a file named
111       that exists,
112       a new folder is being created.
113       Probably the function will be able to handle not just
114       set as parameter,
115       but another iterable types. It was not tested due to
116       shortage of available
117       time.
118       Optional directory parameter can be passed, which can
119       be either
120       relative or absolute.
121
122
123    ''
124    for element in set_of_TLEs:
125        date_string = re.search(PATTERN_DATE, element)
126        if date_string == None:
127            return None
128        else:
129            date = datetime.datetime.strptime(date_string.group
130                (0), '%B %d, %Y')
131            filename = date.strftime('%Y-%m-%d') + '.amsat.tle'
132            ,
133            directoryname = date.strftime('%Y')

```

```

122
123     i = 0          # iteration for the directory name
124
125     while True:
126         if os.path.isdir(directoryname):
127             full_path = os.path.join(directory, directoryname
128                                         , filename)
129             break
130         else:
131             if os.path.exists(directoryname):
132                 if directoryname.find('_') == -1:
133                     directoryname += '_'+ '%02d' %(i)
134                 else:
135                     directoryname[:directoryname.rfind('_')] += '_'
136                     i += 1
137             else:
138                 full_path = os.path.join(directory,
139                                         directoryname, filename)
140
141             os.makedirs(os.path.join(directory, directoryname),
142                         exist_ok=True)
143             tle_file = open(full_path, 'w')
144             tle_file.writelines(element)
145             tle_file.close()
146
147             # Function to look up TLE in the AMSAT mailing list
148             # archive.
149
150             #
151             #
152             def extract_amsat_TLE(AMSAT_maillist_filename, directory=
153                                     '../keps/'):
154                 '''extract_amsat_TLE(AMSAT_maillist_filename,[
155                     directory='../keps/'])'''
156
157                 The mail list file is from http://amsat.org/pipermail/
158                 keps/.
159
160                 The filename is just the year represented by four
161                 digits,
162
163                 a period and string "txt" (i.e. 2006.txt).

```

```

152  uuOptional directory parameter can be passed , which can
      be either
153  uurelative or absolute .
154  uu' '
155  f = open(os.path.join(directory ,
                           AMSAT_maillist_filename) , 'r')
156  f_register = 0
157  string_TLE = ''
158  set_of_string_TLE = set()
159  line = f.readline()
160
161
162  while line:
163      if check_line(PATTERN_START_1 , line):
164          f_register = set_flag(f_register , FLAG_BEGINNING_1)
165          line_start_1 = line
166          line = f.readline()
167          continue
168      elif check_flag(f_register , FLAG_BEGINNING_1):
169          if check_line(PATTERN_START_2 , line):
170              f_register = set_flag(f_register ,
                                     FLAG_BEGINNING_2)
171              line_start_2 = line
172              line = f.readline()
173              string_TLE = append_line(string_TLE , line_start_1
                                         , line_start_2)
174              continue
175      else:
176          f_register = unset_flag(f_register ,
                                     FLAG_BEGINNING_1)
177          line = f.readline()
178          continue
179  else:
180      if check_flag(f_register , FLAG_BEGINNING_1 ,
                     FLAG_BEGINNING_2):
181          if check_line(PATTERN_END , line):
182              string_TLE = append_line(string_TLE , line)
183
184      set_of_string_TLE.add(string_TLE)

```

```

185
186         string_TLE = ''
187         f_register = unset_flag(f_register,
188                         FLAG_BEGINNING_1, FLAG_BEGINNING_2,
189                         FLAG_ENDING)
190         line = f.readline()
191     else:
192         string_TLE = append_line(string_TLE, line)
193         line = f.readline()
194
195     return set_of_string_TLE

```

## A.2 Skript get\_undopplered.py

Python 3 modul pro korekci dopplerovského posuvu.

```

1 import re
2 import datetime, pytz
3 import os, pty
4 import subprocess, shlex
5 import pymediainfo
6 import wav2spectrogram as w2s
7
8 # Function to get some information from the IQ sample
9 # filename.
10 def extract_IQ_filename(filename, directory='../../NO-84'):
11     '''extract_IQ_filename(filename, [directory='../../NO
12     -84'])'''
13
14     """This function is extracting the filename of which
15     pattern
16     will be the same in the future, or the REGEX needs to
17     be changed.
18     The function will return a dictionary containing:
19     *date //YYYYMMDD//
20     *time //HHMMSS//
21     *frequency //CCCMMM// [kHz]
22     *extension //re.(\w{3})//e.g. .wav

```

```

19  uuIf the file is not ending with 'wav', then instead of a
     dictionary
20  uuNone is returned.
21  uu'
22 # How the regex is built -- hint from a filename.
23 #
24 pattern_named = re.compile(r'HDSDR_(?P<date>\d{8})_(?P<
     time>\d{6}).*_(?P<frequency>\d{6})kHz_(RF)\.(?P<
     extension>\w{3})$')      # REGEX pattern with named
     subgroups date, time, frequency, extension
25 filename_regexed_named = re.search(pattern_named,
     filename)
26 if filename_regexed_named != None:
27     file_parameters = filename_regexed_named.groupdict()
28
29     if file_parameters['extension'] == 'wav':           #
         filter out non-wav files
30     return file_parameters
31 else:
32     return None
33 else:
34     print(filename_regexed_named)
35
36 # A little 'key' function for sorting TLE files with
     sorted()
37 def key_function_TLE(date_IQ, date_TLEs):
38     return abs(date_IQ - date_TLEs)
39
40 def find_TLE_for_IQ_file(filename_IQ, directory_TLE='../
     keps/'):
41     '''find_TLE_for_IQ_file(filename_IQ,[directory_TLE
     ='../keps/'])'''
42
43 uuThis function is returning the best matching TLEs file
44 uubased on the IQ filename given and the date encoded in
     it.

```

```

45  uuThe uu best uu match uu means uu the uu least uu difference uu between uu the uu
     dates
46  uuindicated uu by uu the uu names uu of uu the uu TLEs uu and uu IQ uu files .
47  uu ''
48  parameters_IQ = extract_IQ_filename(filename_IQ)
49  if parameters_IQ != None:
50
51      date_IQ = datetime.datetime.strptime(parameters_IQ.
52          get('date'), '%Y%m%d')
53      directory_path_TLEs = os.path.join(directory_TLE, str
54          (date_IQ.year))
55      pattern_TLE_file = re.compile('%d-%02d-' % (date_IQ.
56          year, date_IQ.month))
57      TLEs = list()      # Empty list for TLEs file
58
59      for i in os.listdir(directory_path_TLEs):
60          if bool(re.search(pattern_TLE_file, i)):
61              TLEs.append(i)
62      TLEs = sorted(TLEs, key=lambda TLE: abs(date_IQ -
63          datetime.datetime.strptime(TLE[:10], '%Y-%m-%d'))))
64
65      return TLEs[0]
66  else:
67      return None
68
69  # Function used for undoing the doppler frequency shift
    using the doppler application called by os.system
70  def undoppler_it(filename_IQ, satellite_name='NO-84',
71      satellite_frequency='435350000', directory_IQ='../NO
72      -84', directory_TLE='../keps', location_SDR='lat
73      =49.173238,lon=16.961292,alt=263.73', offset_corr=
74      False):
75      '''undoppler_it(filename_IQ,[satellite_name='NO-84',u
76          satellite_frequency='435350000',udirectory_IQ='..uNO
77          -84',udirectory_TLE='..u/keps',ulocation_SDR='lat
78          =49.173238,lon=16.961292,alt=263.73']):u
79
80  uuOnly uu one uu parameter uu is uu required uu not uu to uu get uu an uu error , uu but
     uu not uu all uu time

```

```

70    will make sense. It is due to the fact that the
71    optional parameters
72    are used only for the purpose of the testing.
73    ''
74
75    if filename_TLE != None:
76        parameters_IQ = extract_IQ_filename(filename_IQ)
77        mediainfo_IQ = pymediainfo.MediaInfo.parse(os.path.
78            join(directory_IQ, filename_IQ))
79        mediainfo_IQ_tracks = mediainfo_IQ.tracks
80        audio = mediainfo_IQ_tracks[1]
81        audio_dict = audio.to_data()
82
83        local_TZ = pytz.timezone('Europe/Prague')
84        naive_datetime_IQ = datetime.datetime.strptime(
85            parameters_IQ.get('date') + parameters_IQ.get('
86            time'), '%Y%m%d%H%M%S')
87        local_datetime_IQ = local_TZ.localize(
88            naive_datetime_IQ, is_dst=None)
89        utc_datetime_IQ = local_datetime_IQ.astimezone(pytz.
90            utc)
91
92        samplerate = '--samplerate' + str(audio_dict.get('
93            sampling_rate')) + ' '
94        if (int(audio_dict.get('resolution')) == 16) & (
95            audio_dict.get('format_settings') == 'Little/
96            Signed'):
97            intype = '--intypeui16'
98            outtype = '--outtypeui16'
99        else:           #in case of strange Wave format, e.g
100            . float samples
101        print('Format yet not tested: ')
102        print('*resolution:', audio_dict.get('
103            resolution'))
104        print('*format_settings', audio_dict.get('
105            format_settings'))
106    return 1

```

```

96     tlefile      = '--tlefile' + os.path.join(
97         directory_TLE, filename_TLE[:4], filename_TLE) + '
98             '
99     tlename      = '--tlename' + satellite_name
100        +
101    location     = '--location' + location_SDR
102        +
103    frequency   = '--frequency' + satellite_frequency
104        +
105    time         = '--time' + local_datetime_IQ.
106        isoformat()[:-6] +
107    infile       = os.path.join(directory_IQ, filename_IQ)
108        +
109 #    for i in filename_IQ.split('.')[:-1]:
110 #        filename+= i
111 #    if offset_corr:
112 #        outfile    = os.path.join(directory_IQ, w2s.join(
113 #            filename_IQ.split('.')[:-1]) + '.UD.OC.' +
114 #            filename_IQ.split('.')[-1])
115 #    else:
116 #        outfile    = os.path.join(directory_IQ, w2s.join(
117 #            filename_IQ.split('.')[:-1]) + '.UD.' +
118 #            filename_IQ.split('.')[-1])
119
120    sox_cmd_rbche = '--rate' + str(audio_dict.get(
121        'sampling_rate')) + '--bits16--channels2--'
122        encodingsigned-integer'
123    sox_cmd_out = 'sox' + sox_cmd_rbche + '--typeraw-
124        ' + sox_cmd_rbche + '--typewav' +
125        outfile
126    sox_cmd_inp = 'sox' + sox_cmd_rbche + '--typewav'
127        + infile + sox_cmd_rbche + '--typeraw'
128
129    if offset_corr:
130        offset = int(satellite_frequency) - (1000 * int(
131            parameters_IQ.get('frequency')))
132        offset = '--offset%d' % (offset)
133    doppler_cmd = 'dopplertrack'+ samplerate + intype
134        + outtype + tlefile + tlename + location +

```

```

                frequency + time + offset
117     else:
118         doppler_cmd = 'doppler' + track + samplerate + intype
119                     + outtype + tlefile + tlename + location +
120                     frequency + time
121
122         full_command_doppler = sox_cmd_inp + '|'
123                     + doppler_cmd + '|'
124                     + sox_cmd_out
125
126         os.system (full_command_doppler)
127         print(full_command_doppler)
128         print('undoppler:' + outfile + str(type(outfile)))
129         return outfile
130
131     else:
132         return None

```

### A.3 Skript wav2spectrogram.py

Python 3 modul pro tvorbu spectrogramů.

```

1 import matplotlib
2 matplotlib.use('Agg')
3 import numpy as np
4 import scipy.signal as signal
5 import numpy as np
6 import scipy.signal as signal
7 from scipy.io import wavfile
8 import matplotlib.pyplot as plt
9 from matplotlib.ticker import FuncFormatter
10
11 # Formating function used for the purpos of plotting
12 # with MHz
13 def format_mega(x, pos):
14     if x >= 1e6:
15         return '%3.2f' % (x/1e6)
16     #     return x/1e6
17     else:
18         return x

```

```

18 # Little function to join a list of strings into one
19 # string
20 def join(inp_list):
21     out_string = ''
22     for i in inp_list:
23         out_string += i
24     return out_string
25
26 def IQ_to_spectrogram(filename_IQ, mode):
27     rate, data = wavfile.read(filename_IQ, mmap=False)
28     data_cmpl = data.view(np.int16).astype(np.float32).view
29         (np.complex64)    # Changing the data type to complex
30         and
31     data_cmpl = data_cmpl.reshape(data_cmpl.shape[0] *
32         data_cmpl.shape[1])  # Reshaping to fit the specgram
33
34 Fc = int(filename_IQ.split('_')[3][-3])*1000
35
36 cmap = plt.get_cmap('spectral')
37 function_formatter = FuncFormatter(format_mega)
38 # vmin = 10 * np.log10(np.max(np.abs(data_cmpl))) - 40
39
40 color_legend, spectrogram = plt.subplots()
41 Sxx, f, t, cb = spectrogram.specgram(data_cmpl, NFFT
42         =65536, Fs=rate, Fc=Fc, vmin=-80, vmax=-18, sides='
43         onesided', mode=mode)
44 spectrogram.set_ylabel('f [MHz]')
45 spectrogram.yaxis.set_major_formatter(
46         function_formatter)
47 spectrogram.set_xlabel('t [s]')
48 spectrogram.yaxis.limit_range_for_scale(0, rate)
49 color_legend.colorbar(cb, label='P/f [dB/Hz]')
50
51 x1, x2, y1, y2 = spectrogram.axis()
52 spectrogram.axis((x1, x2, 435338000, 435362000)) #
53     leave x range the same, change y (frequency) range
54
55 plt.savefig('..'+join(filename_IQ.split('.')[:-1])+'.
56     png', dpi=600, bbox_inches='tight', pad_inches=0.5)

```

```
48     plt.close()
49
50     return '...' + join(filename_IQ.split('..')[:-1]) + '.png'
```