ČESKÉ VYSOKÉ
UČENÍ TECHNICKÉ
V PRAZE

F3

Fakulta elektrotechnická
Katedra měření

Diplomová práce

# Zobrazení Waterfall spektra na PC

**Patrik Petrýdes**
Letectví a kosmonautika, obor Avionika

Květen 2022
Vedoucí práce: doc. Dr. Ing. Pavel Kovář

# Poděkování / Prohlášení

Chtěl bych poděkovat svému vedoucímu práce, doc. Dr. Ing. Pavlu Kovářovi za vstřícnost a ochotu při konzultacích a vypracování diplomové práce.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškerou použitou literaturu.

V Praze, 20. května 2022

# Abstrakt / Abstract

Cílem práce bylo vytvořit program, který přijímá a zobrazuje amplitudové spektrum ve formě waterfall grafu na počítači. Program byl realizován ve třech verzích. První verze byla vytvořena za pomocí programovacího jazyku Python a jeho knihoven PyQt a PyQtGraph. Druhá verze byla vytvořena za pomocí programovacího jazyku Python a jeho knihoven Tkinter a Matplotlib. A třetí verze byla vytvořena za pomocí programovacího jazyku C++ a jeho knihovny QCustomPlot. Část práce byla pro tyto programy vytvořit různé funkce pro manipulaci s grafem. Funkce byly v práci jednotlivě vysvětleny. Všechny verze programu byly popsány a byla testována jejich výkonnost pro 2D, a pro některé z nich i 3D, zobrazení. Pro porovnání se testovalo na třech počítačích s různým výpočetním výkonem. Z testování vzešel závěr, že zadání bylo splněno jen pro jeden z těchto programů.

**Klíčová slova:** Fourierova řada, Fourierova transformace, FT, diskrétní Fourierova transformace, DFT, rychlá Fourierova transformace, FFT, amplitudové spektrum, spektrogram, waterfall graf, waterfall spektrum, programovací jazyk Python, programovací jazyk C++

The aim of the work was to create a program that receives and displays the amplitude spectrum in the form of a waterfall graph on a computer. The program was implemented in three versions. The first version was created using the Python programming language and its PyQt and PyQtGraph libraries. The second version was created using the Python programming language and its libraries Tkinter and Matplotlib. And the third version was created using the C ++ programming language and its QCustomPlot library. Part of the work was to create various functions, for manipulating the graph, for these programs. The functions were explained individually in the work. All versions of the program have been described and tested for performance in 2D, and for some, 3D, displays. For comparison, the programs were tested on three computers with different computing power. The testing concluded that the assignment was met for only one of these programs.

**Keywords:** Fourier series, Fourier transform, FT, discrete Fourier transform, DFT, fast Fourier transform, FFT, amplitude spectrum, spectrogram, waterfall plot, waterfall spectrum, Python programming language, C++ programming language

**Title translation:** Display of Waterfall spectrum on PC

# Obsah /

# Tabulky / Obrázky

# Kapitola 1
# Úvod

Signál lze reprezentovat v časové a spektrální oblasti. Pro monitorování signálů je vhodné analyzovat spektrum signálu. K zobrazování spektra signálu slouží takzvaný spektrogram, v některých případech též zvaný waterfall graf, což je graf, zobrazující časový vývoj amplitudového spektra. Práce se zabývá vyvinutím programu, pro zobrazení tohoto grafu na počítači, se specifickými požadavky. Programy byly ve výsledku vytvořeny tři. Dvě verze byly napsány v programovacím jazyce Python, jen za pomocí různých knihoven. A třetí verze byla napsaná v programovacím jazyce C++.

Spektrum signálu slouží ke kontrole modulace signálů, monitorování rušení signálů, monitorování využití rádiového spektra, monitorování mobilních a satelitních komunikací a další. To znamená, že sledování spektra signálu je velmi důležité pro různé technické účely.

Data jsou přijímána z SDR přijímače prostřednictvím TCP/IP. SDR je zkratka pro Software Defined Radio, ve kterém se rádiové komunikace zpracovávají pomocí softwaru nebo firmwaru a úlohy zpracování signálu se tedy provádí je programově namísto zpracování hardwarem. SDR jsou dnes cenově velmi dostupné, což je dělá velice populární mezi radioamatéry, protože umožňují rychle vyhledávat radiové signály jednotlivých radiokomunikačních služeb a stanic.

První část práce se zabývá krátkým teoretickým úvodem do tématu. Protože je hlavním cílem práce vytvoření programu pro zobrazení spektra, nebude práce popisovat teorii do detailů. Protože ne každý zná pojmy waterfall graf nebo spektrogram, budou na konci této části vysvětleny.

Druhá část se zabývá samotným návrhem programu. Na začátku se prozkoumávají použité programovací jazyky a jejich knihovny. Dále jsou popsány jednotlivé funkce programu a možnosti jeho využití. Poté jsou konečně popsány a vyhodnoceny všechny verze programu. Vyhodnocení se zabývá hlavně porovnáním programů s ohledem na výkonnost. Na konci práce je ještě krátký průzkum již existujících nástrojů.

# Kapitola 2
# Fourierova řada a transformace

Tato kapitola se zabývá vysvětlením základních pojmů Fourierova řada a Fourierova transformace. Tyto pojmy jsou v práci vysvětleny pro pochopení některých základních konceptů spojených s analýzou spektra, protože přímo souvisí s řešenou problematikou.

## 2.1 Fourierova řada

Při řešení technických problémů, kde se vyskytují periodické děje, jako například při akustických nebo elektrických kmitech, je vhodné vyjádřit danou funkci pomocí nekonečných trigonometrických řad.

Nejjednodušším netriviálním příkladem periodických funkcí jsou základní goniometrické funkce sinus a kosinus. Lze proto očekávat, že periodická funkce se dá aproximovat buď lineární kombinací konečného počtu goniometrických funkcí, nebo přímo nekonečnou funkční řadou, jejíž členy jsou goniometrické funkce. Fourierova řada slouží k rozkladu periodického signálu na součet sinusových funkcí.

První lze definovat trigonometrickou řadu, což je nekonečná funkční řada s periodou $T = \frac{2\pi}{\omega}$:

$$\frac{a_0}{2} + \sum_{k=1}^{\infty}(a_k cos(k\omega x) + b_k sin(k\omega x)) \tag{1}$$

Kde $a_0$, $a_k$ a $b_k$ jsou konstanty, nazývající se koeficienty trigonometrické řady a $k$ je celé číslo. Konverguje-li řada na nějakém základním intervalu periodicity délky $T = \frac{2\pi}{\omega}$, potom je konvergentní na celém svém rozsahu.

Pokud máme periodickou funkci $f$ se základní periodou $T$, integrovatelnou na intervalu periodicity $\langle \alpha, \alpha + T \rangle$ a $\alpha$ je reálné číslo. Potom lze trigonometrickou řadu rozvést jako:

$$\begin{aligned}
&\frac{a_0}{2} + \sum_{k=1}^{\infty}(a_k cos(k\omega x) + b_k sin(k\omega x)) \\
a_0 &= \frac{2}{T}\int_{\alpha}^{\alpha+T} f(x)dx \\
a_k &= \frac{2}{T}\int_{\alpha}^{\alpha+T} f(x)cos(k\omega x)dx \\
b_k &= \frac{2}{T}\int_{\alpha}^{\alpha+T} f(x)sin(k\omega x)dx
\end{aligned} \tag{2}$$

V tomto případě se trigonometrická funkce nazývá Fourierova řada funkce $f$ a koeficienty $a_k$, $b_k$ této řady se nazývají Fourierovy koeficienty funkce $f$. Fourierova řada je konvergentní jen pokud periodická funkce $f$ splňuje takzvané Dirichletovy podmínky.

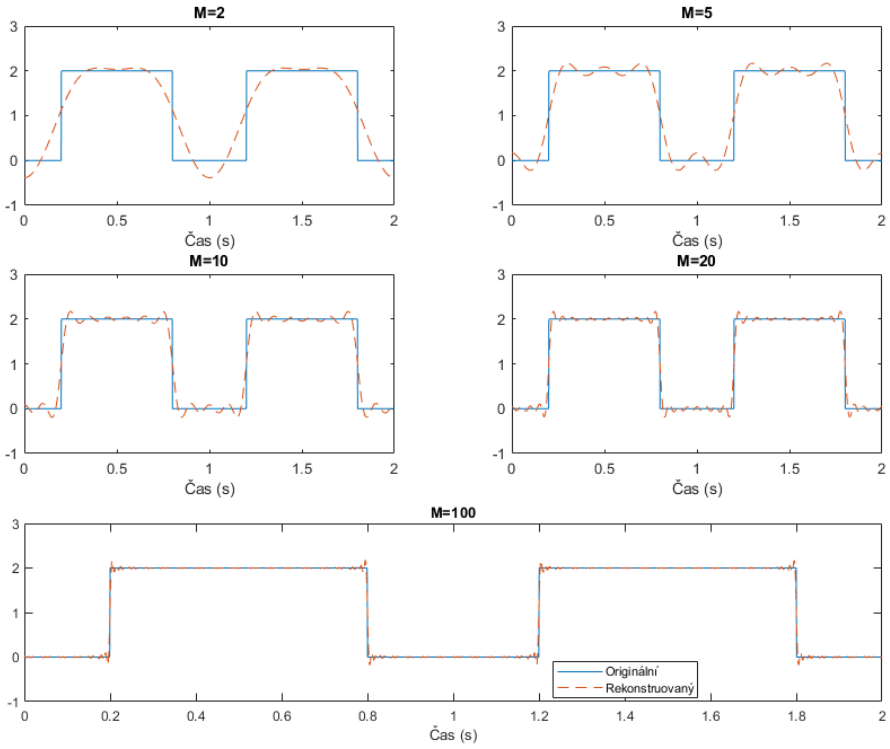
S vhodnými koeficienty lze provést jeden cyklus (nebo periodu) součtů pro aproximaci libovolné funkce v daném intervalu. Proces odvozování vah, které popisují danou funkci,

je formou Fourierovy analýzy. Je však třeba mít na paměti, že Fourierova řada může být použita pouze pro periodické funkce.

Pro pochopení funkce Fourierovy řady lze uvést následující specifický příklad. Máme obdélníkový periodický signál, který chceme rekonstruovat pomocí součtu sinusových funkcí. Jako první získáme váhové koeficienty, které pomůžou danou funkci rekonstruovat. Periodický signál $x_p(t)$ s hlavní periodou $T$ lze tedy rozvést takto:

$$\begin{aligned} x_p(t) &= a_0 + \sum_{i=1}^{M} a_k cos\,(\omega t k) + \sum_{k=1}^{M} b_k sin\,(\omega t k) \\ a_0 &= \frac{1}{T}\int_0^T x_p(t)dt \\ a_k &= \frac{2}{T}\int_0^T x_p(t) cos\,(\omega t k)\,dt \\ b_k &= \frac{2}{T}\int_0^T x_p(t) sin\,(\omega t k)\,dt \end{aligned} \tag{3}$$

V tomto případě máme váhové koeficienty $a_i$ a $b_i$ a jak se bude $M$ zvyšovat, váha faktorů se bude zmenšovat a jejich příspěvek k součtu bude menší. Proto, jak se $M$ zvyšuje, se sumace přibližuje a blíže skutečnému periodickému signálu. Obrázek číslo 2.1 ukazuje důsledek zvyšování $M$ a jak je obdélníkový signál rekonstruován omezeným počtem sumací $M$. Jak je z obrázku vidět, jak $M$ stoupá, rekonstruovaný signál konverguje k původnímu periodickému signálu. [1–2]




**Obrázek 2.1.** Součet Fourierovy řady pro $M = 2, 5, 10, 20$ obdélníkového signálu ve dvou periodách. [1]

## 2.2 Fourierova transformace

Pojem Fourierovy řady lze zobecnit i na neperiodické signály. Taková operace se nazývá Fourierova transformace.

Signál musí splňovat určitá kritéria známá jako Dirichletovy podmínky před tím než lze aplikovat Fourierovu transformaci. Naštěstí většina signálů tyto podmínky splňuje.

Fourierova transformace signálu $x(t)$ je definována jako:

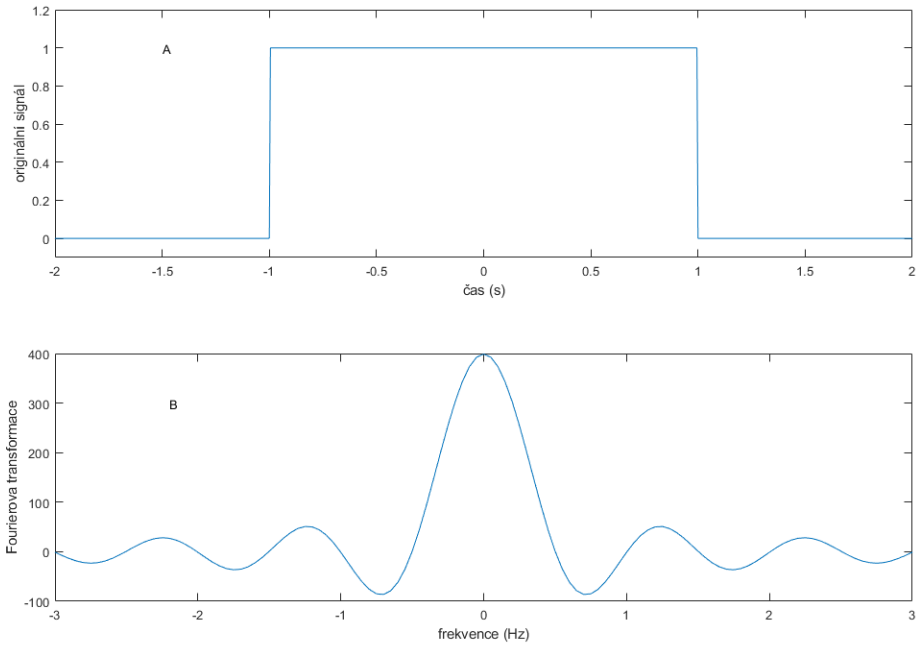
$$X(i\Omega) = \int_{-\infty}^{\infty} x(t)e^{-j\Omega t}dt \tag{4}$$

Je zajímavé, že Fourierova transformace je „invertibilní“, což znamená, že pokud máme kompletní informace o Fourierově transformaci signálu, můžeme znovu získat původní signál v časové oblasti, pomocí inverzní Fourierovy transformace. Inverzní Fourierova transformace je definována jako:

$$x(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} X(j\Omega)e^{-j\Omega t}d\Omega \tag{5}$$

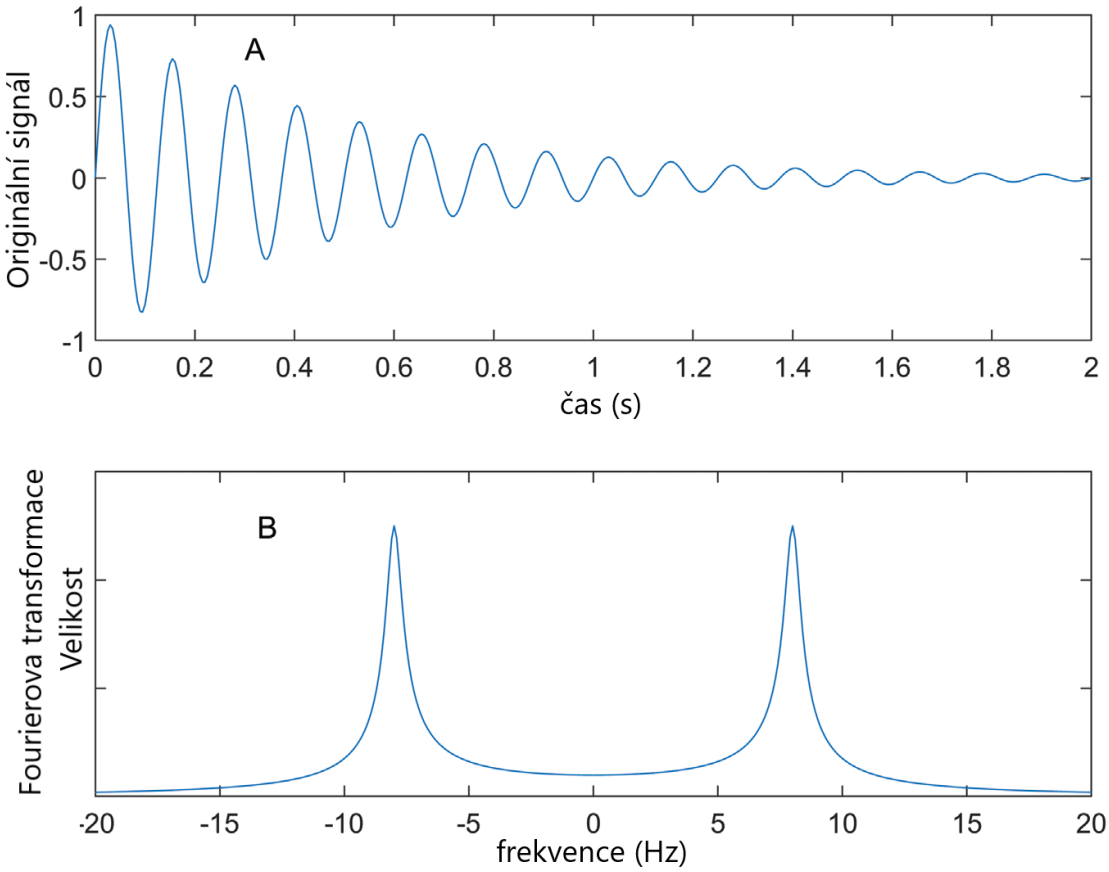
Představme si jeden pulz s neomezenou šířkou. Pokud je na tento signál aplikována Fourierova transformace, vznikne transformovaný signál, jak je vidět na obrázku číslo 2.2. Tento signál má speciální tvar, který je známý jako funkce sinc.

Z obrázku číslo 2.2b je také vidět, že většina energie transformovaného signálu leží v intervalu $(-0.5, 0.5)$ Hz a mimo tento interval je energie signálu nízká. [1, 3]




**Obrázek 2.2.** (a) Časová reprezentace jednoho pulzu v čase a (b) Fourierova transformace tohoto pulzu ve frekvenční oblasti [1]

Další příklad může být obrázek číslo 2.3, který ukazuje zatlumenou komplexní exponenciálu. V tomto případě se jedná o volný indukční rozpad, který lze zaznamenat




**Obrázek 2.3.** (a) Časová reprezentace signálu volného indukčního rozpadu (FID) a (b) Fourierova transformace tohoto signálu. [1]

pomocí nukleární magnetické rezonanční spektroskopie. Je vidět, že na rozdíl od časové reprezentace signálu, která je rozprostřena v určitém čase, je její frekvenční reprezentace značně zhuštěná do úzkého rozsahu frekvencí.[1]

Pokud manipulujeme se signálem v časové oblasti, manipulace se také odráží ve frekvenční oblasti. Například Fourierova transformace součtu dvou signálů v časové oblasti se bude rovnat součtu Fourierových transformací každého ze signálů. Posun signálu v čase se odráží na fázi Fourierovy transformace. A násobení Fourierových transformací signálů odpovídá konvoluci těchto signálů v čase. [1–2]

# Kapitola 3
# Diskrétní Fourierova transformace

Fourierova transformace může být také definována pro signály v diskrétním čase. Její matematická formulace je podobná té, která byla uvedena pro spojité časové signály, ale pro diskrétní časové signály je třeba integraci nahradit běžným součtem. Fourierova transformace pro diskrétní časový signál $x(n)$ by tedy byla definována jako:

$$X(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x(n)e^{-j\omega n} \tag{1}$$

Z rovnice je vidět, že frekvenční reprezentace diskrétního signálu v časové oblasti je spojitá napříč celým měřítkem. Ve vzorkovaném signálu je jen omezený počet vzorků v omezeném časovém okně. Pokud se ale podíváme na již zmíněný vztah je vidět, že čas $(n)$ je definovaný v rozmezí od mínus nekonečna do nekonečna. Jeden způsob, jak přesto spočítat rovnici mimo její vzorkovaný interval je předpokládat, že signál má mimo tento vzorkovaný interval nulovou amplitudu. Přesná rekonstrukce spojitého, frekvenčně omezeného signálu z jeho vzorků je možná tehdy, pokud byla vzorkovací frekvence vyšší než dvojnásobek nejvyšší harmonické složky vzorkovaného signálu. Toto vychází z věty o vzorkovacím teorému. Abychom tedy neztratili žádné informace o Fourierově transformaci, musí být vzorkování minimálně dvakrát vyšší než je maximální frekvence původního signálu.

Fourierovy transformace ve frekvenční oblasti by odpovídala signálu v čas s neomezeným časovým rozsahem a to, že takový signál by pak měl omezený frekvenční rozsah ve frekvenční oblasti. Tedy místo toho, abychom předpokládali, že signál s omezeným počtem vzorků má nulovou amplitudu mimo tento rozsah, můžeme předpokládat, že se mimo tento rozsah signál opakuje. [1]

Pokud si představíme diskrétní signál v čase $x(n)$, který má nenulovou amplitudu pouze uvnitř rozsahu od 0 do $N-1$ a pokud vzorkuje $X(e^{j\omega})$ v intervalech $\omega = \frac{2\pi}{N}k$, kde $k = 0, ..., N-1$, můžeme vzorkovaný $X(e^{j\omega})$ napsat jako:

$$\tilde{X}(k) = X(e^{j\frac{2\pi}{N}k}) = \sum_{n=0}^{N-1} x(n)e^{-j\frac{2\pi}{N}kn} \tag{2}$$

Tento vztah je známý jako diskrétní Fourierova transformace nebo také analytická rovnice, protože umožňuje frekvenční analýzu vzorku signálu. Je jednoduché algebraicky ověřit, že diskrétní Fourierova transformace je invertibilní a že signál lze rekonstruovat na základě jeho diskrétní Fourierovy transformace. [1]

Protože je diskrétní Fourierova transformace invertibilní, můžeme se podívat na vztah, kterému se často říká „syntézní rovnice“:

$$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} \tilde{X}(k)e^{j\frac{2\pi}{N}kn} \tag{3}$$

Tato rovnice umožňuje rekonstrukci signálu v čase na základě jeho diskrétní Fourierovy transformace. [1]

## 3.1 Frekvenční rozlišení

Diskrétní časové signály jsou složeny ze sekvence vzorků signálu. Lze si tedy představit, že jsou stejné jako pro spojitý časový signál, jen odebírány ve stejně vzdálených intervalech. Pro diskrétní signál $x(n)$ lze tedy psát:

$$x(n) = x_c(nT_s) \tag{4}$$

Kde $x_c(t)$ je libovolný spojitý časový signál, u kterého ale máme pouze jeho odpovídající amplitudy v časech, které jsou celočíselnými násobky $T_s$, což je perioda vzorkování. Matematicky řečeno, tento scénář, po určitém zjednodušení, odpovídá vynásobení $x_c(t)$ sérií ostrých impulsů proložených periodou $T_s$. Tento sled impulsů lze definovat jako:

$$x_s(t) = s(t)x_c(t) \tag{5}$$

V této rovnici je $s(t)$ posloupnost Diracových impulsů, což znamená, že se $s(t)$ rovná nule v libovolném časovém bodě kromě časů, které jsou celočíselnými násobky $T_s$ a v těchto časových bodech se jeho energie rovná jedné. Výsledkem násobení $s(t)$ a $x_c(t)$ je sled impulzů, jehož energie při každém z impulsů se rovná amplitudě $x_c(t)$ v odpovídajícím časovém bodě. Diracovy impulsy jsou ovšem vzhledem ke své nulové šířce v v praxi nerealizovatelné. Proto se vzorkování provádí pomocí periodických obdélníkových impulsů nenulové šířky. [1–2]

Vzorkování diskrétní Fourierovy transformace se provádí v krocích $\frac{2\pi}{N}$ ve frekvenční ose, kde $N$ je počet vzorků signálu v čase. Frekvenční interval mezi dvěma po sobě jdoucími vzorky ve frekvenční oblasti je tedy $\frac{2\pi}{N}$ radiánů za sekundu. Pro vztah mezi frekvenční reprezentací spojitého signálu v čase a diskrétního signálu v čase platí $\Omega T_s = \omega$, kde $T_s$ je perioda vzorkování a její recipročí $F_s = \frac{1}{T_s}$ je vzorkovací frekvence. Konkrétně pro $2\pi$ radiánů za sekundu ve frekvenční oblasti signálu s diskrétním časem, odpovídá $2\pi F_s$ ve frekvenční oblasti spojitého časového signálu. Odpovídající frekvenční rozlišení pro vzorkovaný spojitý časový signál je $\frac{F_s}{N}$.

Podíváme-li se na spojitý časový signál, který má časový rozsah $T$, tedy, že signál byl vzorkovaný v časovém okně mezi 0 a $T$ sekundami signálu s vzorkovací frekvencí $F_s$. To znamená, že $N$, neboli počet vzorků, bude $F_sT$. Pokud si toto $N$ dosadíme do frekvenčního rozlišení, zjistíme, že $\frac{F_s}{N} = \frac{1}{T}$, tedy že frekvenční rozlišení se rovná převrácené hodnotě časového rozsahu spojitého časového signálu.

Pokud tedy potřebujeme získat diskrétní Fourierovu transformaci, která dokáže brát v potaz i malé změny ve frekvenční oblasti, spojitý časový signál by měl být vzorkován v delším časovém rozsahu, aby se daly zaznamenat i malé změny ve frekvenční oblasti. Z toho vidíme, že pokud chceme zaznamenat změny například o frekvencí 0.1 Hz, spojitý časový signál by měl být vzorkován v rozsahu alespoň 10 sekund, a pokud je požadováno ještě jemnější rozlišení, je možné při výpočtu diskrétní Fourierovy transformace použít ještě delší posloupnost vzorků.

Zdá se, že čím delší je časový rozsah signálu, tím lepší bude jeho frekvenční rozlišení. Ale to platí jen pokud nebereme v potaz „nestacionaritu“ signálu. Ve skutečnosti si nejsme schopni vybrat velmi dlouhý časový rozsah a navíc zaznamenávat signál, ve dlouhém časovém rozpětí, přináší různá technická úskalí.

Další problém je, že když je signál analyzován v omezeném časovém okně, frekvenční charakteristiky samotného okna ovlivňují frekvenční rozlišení. To znamená, že i když je interval frekvence mezi dvěma po sobě jdoucími vzorky v diskrétní Fourierově transformaci jen malé $\Delta f$, pokud má signál dvě frekvenční složky které jsou o něco dále než

$\Delta f$, je možné, že kvůli frekvenčním charakteristikám okna, diskrétní Fourierova transformace stále nemusí být schopna plně rozlišit tyto dva komponenty od sebe navzájem. [1]

# Kapitola 4
# Rychlá Fourierova transformace

Díky rychlé Fourierovy transformaci je dnes už možné přecházet tam a zpět mezi signálem v časové oblasti a frekvenčním spektrem s dostatečnou rychlostí a jednoduchostí, až vznikla zcela nová řada aplikací pro tento klasický matematický nástroj. Tato kapitola je napsána jako úvod do rychlé Fourierovy transformace a s tím souvisejícího Cooley-Tukeyho algoritmu.

Fourierova transformace se dlouho používala pro charakterizaci lineárních systémů a pro identifikaci frekvenčních složek tvořících spojitý signál. Když je však signál vzorkován nebo má být systém analyzován na digitálním počítači, tak musí být použita diskrétní verze Fourierovy transformace. Ačkoli většina vlastností spojité Fourierovy transformace je zachována, několik rozdílů vyplývá z omezení, že diskrétní Fourierova transformace musí pracovat se vzorkovaným signálem definovaným v konečných intervalech.

Rychlá Fourierova transformace (FFT) je jednoduše jen metoda pro optimalizovaný výpočet, z hlediska počtu operací, diskrétní Fourierovy transformace. Rychlá Fourierova transformace lze použít místo spojité Fourierovy transformace pouze v případě, pokud by mohla být použita diskrétní Fourierova transformace, ale s podstatně sníženým časem výpočtu. [4]

Pro pochopení FFT začneme s upraveným, již zmíněným, vztahem pro pár diskrétních Fourierových transformací pro vzorkovaný signál. Tyto vztahy se dají zapsat jako:

$$X(j) = \frac{1}{N}\sum_{k=0}^{N-1} x(k)e^{-i\frac{2\pi}{N}jk}x(k) \quad = \sum_{j=0}^{N-1} X(j)e^{i\frac{2\pi}{N}jk} \tag{1}$$

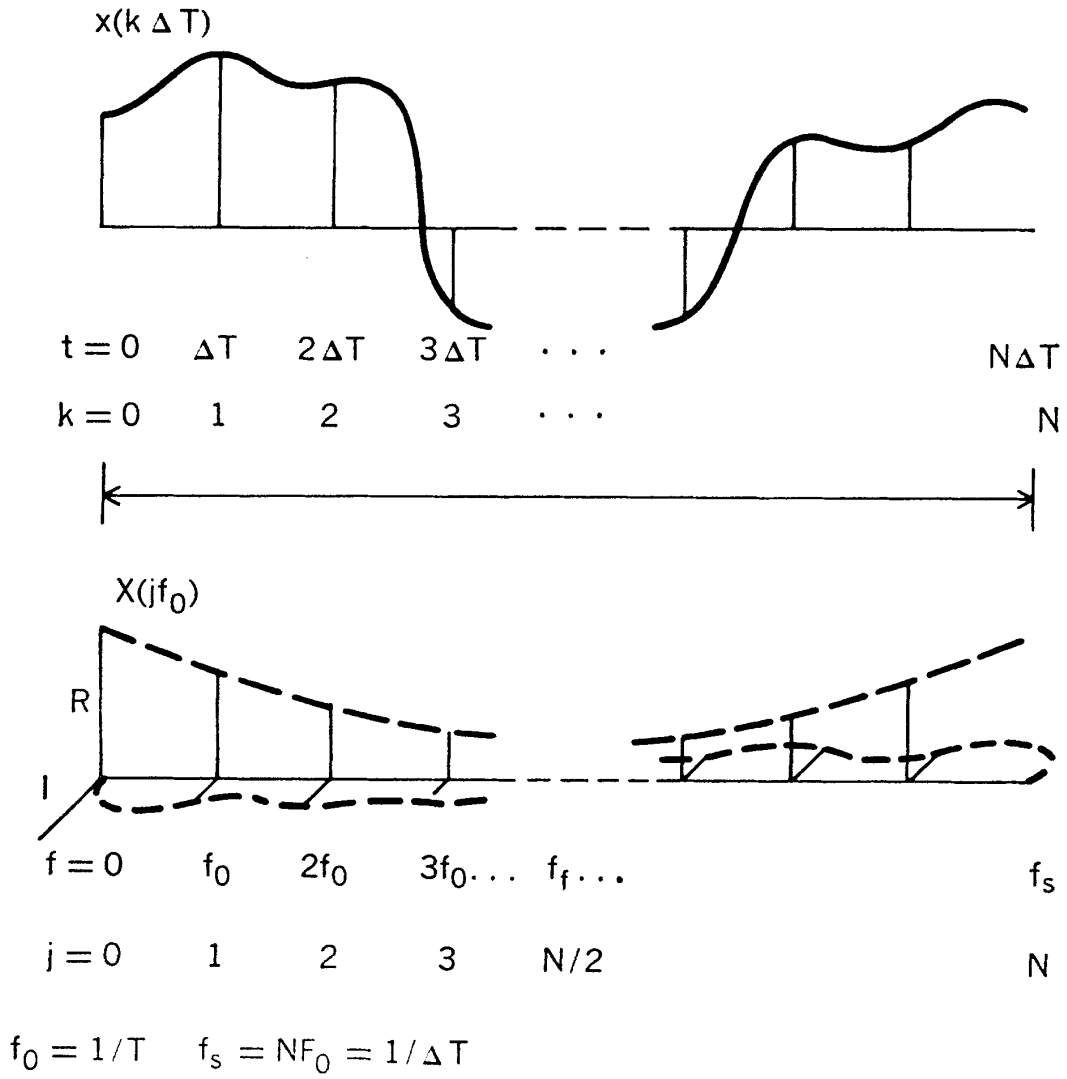
Kde $j = 0, 1, ..., N-1; \quad k = 0, 1, ..., N-1; \quad i = \sqrt{-1}$. Jak $X(j)$, tak $x(k)$ jsou obecně komplexní řady. Velké $X(j)$ představuje funkci ve frekvenční oblasti a malé $x(k)$ je funkce v časové oblasti. [4]

Pokud je výraz $e^{\frac{2\pi i}{N}}$ nahrazen výrazem $W_N$, získá Transformační pár diskrétní Fourierovy transformace tvar:

$$X(j) = \frac{1}{N}\sum_{k=0}^{N-1} x(k)W_N^{-jk}x(k) \quad = \sum_{j=0}^{N-1} X(j)W_N^{jk} \tag{2}$$

Příklad časové řady s reálnou hodnotou a její související diskrétní Fourierova transformace je znázorněna na obrázku číslo 4.1. Předpokládá se, že časová řada $x(k\Delta T)$ je periodická v časové oblasti periody $T$ sekund a množina Fourierových koeficientů $X(jf_0)$ je periodická přes vzorkovací frekvenci $f_i$. U každé funkce je zobrazena pouze jedna úplná perioda. [4]

Základní frekvence $f_0$ a vzorkovací perioda $\Delta T$ se v rovnici (2) explicitně neobjevují, ale každé $j$ by mělo být interpretováno jako harmonické číslo a každé $k$ odkazuje na číslo periody vzorku. To znamená, že skutečná frekvence je součinem $j$ a $f_0$ a skutečný čas je součinem $k$ a $\Delta T$.




**Obrázek 4.1.** Reálný signál a jeho komplexní diskrétní Fourierova transformace zobrazená ve formátu algoritmu rychlé Fourierovy transformace. [4]

Když je $x(k)$ reálná řada, reálná část $X(j)$ je symetrická podle skládací frekvence $f_f$ (kde $f_f = f_s/2$) a imaginární část je antisymetrická. Protože $X(j)$ bylo interpretováno jako periodické. Můžeme tedy říci, že skutečná část $X(j)$ je sudá funkce a že imaginární část $X(j)$ je lichá funkce. To také znamená, že Fourierovy koeficienty mezi $N/2$ a $N-1$ lze považovat za harmonické `negativní frekvence` mezi $-N/2$ a $-1$. Také lze druhou polovinu časové řady interpretovat jako záporný čas, tedy, že nastala před $t = 0$.

Dále se podíváme na odvození Cooley-Tukeyho algoritmu rychlé Fourierovy transformace pro vyhodnocení druhé rovnice (2), pro dané $N = 8$. Cooley-Tukeyho algoritmus je zdaleka nejpoužívanější algoritmus pro rychlou Fourierovu transformaci. Toto odvození je vhodné i pro dopřednou transformaci, protože první rovnice (2) lze přepsat do formy: [4]

$$X(j) = \frac{1}{N}[\sum_{k=0}^{N-1} x(k) * W_N^{jk}]^* \tag{3}$$

Kde hvězdička odkazuje na operaci komplexní konjugace. Alternativně, lze algoritmus rychlé Fourierovy transformace, použitý pro výpočet druhé rovnice (2), změnit předefinováním $W_N$ na $e^{\frac{-2\pi i}{N}}$ a vydělením každého výsledku daným $N$.

Použitím Cooleyho zápisu, zahrnuje algoritmus výpočtu rychlé Fourierovy transformace vyhodnocení výrazu: [4]

$$\hat{X}(j) = \sum_{k=0}^{N-1} A(k) * W^{jk} \tag{4}$$

Kde $j = 0, 1, ..., N-1$ a $W = e^{\frac{2\pi i}{N}}$. $\hat{X}$ a $A$ mohou být interpretovány jako $X^*$ a $x^*/N$, pokud je počítána dopředná transformace, nebo $x$ a $X$, pokud je počítána transformace inverzní.

Když se $N$ rovná 8, je vhodné reprezentovat jak $j$, tak $k$ jako binární čísla. Tedy pro $j = 0, 1, ..., 7$ a $k = 0, 1, ..., 7$, můžeme psát: $j = j_2 4 + j_1 2 + j_0$ a $k = k_2 4 + k_1 2 + k_0$, kde $j_0, j_1, j_2, k_0, k_1, k_2$ mohou nabývat hodnot 0 a 1. Pomocí této reprezentace $j$ a $k$, můžeme psát rovnici (4) jako: [4]

$$\hat{X}(j_2, j_1, j_0) = \sum_{k_0=0}^{1} \sum_{k_1=0}^{1} \sum_{k_2=0}^{1} A(k_2, k_1, k_0) W^{(j_2 4 + j_1 2 + j_0)(k_2 4 + k_1 2 + k_0)} \tag{5}$$

Když vezmeme v potaz, že $W^{m+n} = W^m W^n$, získáme: [4]

$$W^{(j_2 4 + j_1 2 + j_0)(k_2 4 + k_1 2 + k_0} = W^{(j_2 4 + j_1 2 + j_0) k_2 4} W^{(j_2 4 + j_1 2 + j_0) k_1 2} W^{(j_2 4 + j_1 2 + j_0) k_0} \tag{6}$$

Podíváme-li se na tyto pojmy jednotlivě, je vidět, že je lze zapsat ve formě: [4]

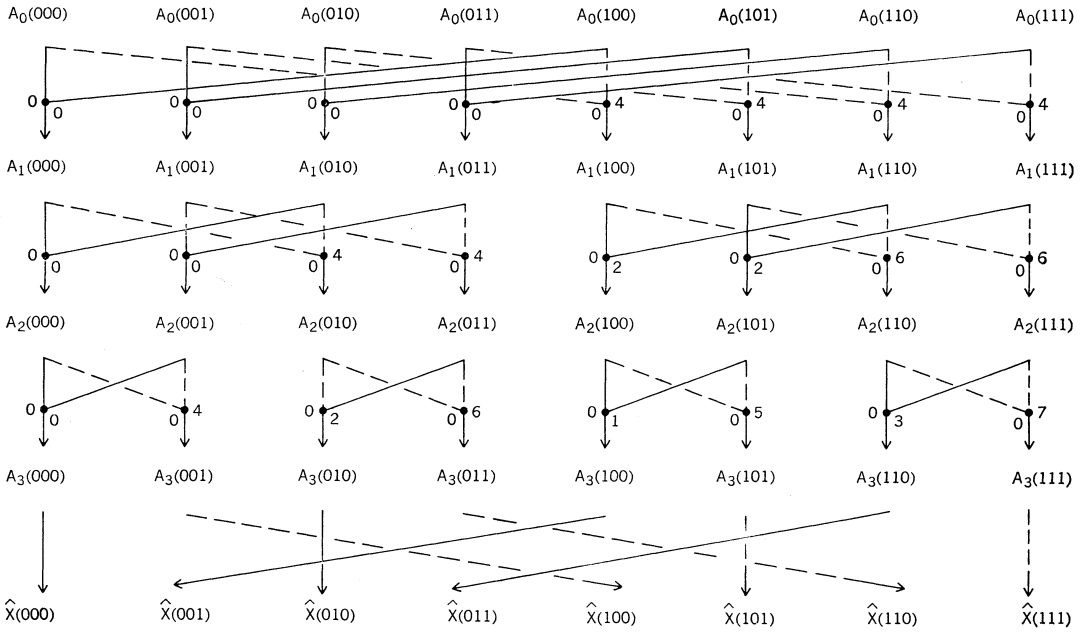
$$\begin{aligned} W^{(j_2 4 + j_1 2 + j_0) k_2 4} &= [W^{8(j_2 2 + j_1) k_2}] W^{j_0 k_2 4} \\ W^{(j_2 4 + j_1 2 + j_0) k_1 2} &= [W^{8 j_2 k_1}] W^{(j_1 2 + j_0) k_1 2} \\ W^{(j_2 4 + j_1 2 + j_0) k_0} &= W^{(j_2 4 + j_1 2 + j_0) k_0} \end{aligned} \tag{7}$$

Všimněme si také, že $W^8 = [e^{\frac{2\pi i}{8}}]^8 = e^{2\pi i} = 1$. Tudíž části rovnic (7) v závorkách lze nahradit jedničkou. To znamená, že rovnici (5) lze zapsat ve formě: [4]

$$\hat{X}(j_2, j_1, j_0) = \sum_{k_0=0}^{1} \sum_{k_1=0}^{1} \sum_{k_2=0}^{1} A(k_2, k_1, k_0) W^{j_0 k_2 4} W^{(j_1 2 j_0) k_1 2} W^{(j_2 4 j_1 2 + j_0) k_0} \tag{8}$$

V této formě je vhodné provádět každý ze součtů samostatně a pracovat s mezivýsledky. Všimněme si, že každá sada se skládá pouze z osmi částí a že je třeba uložit pouze nejnovější sadu. Rovnice tak mohou být přepsány do tvaru: [4]

$$\begin{aligned} A_1(j_0, k_1, k_0) &= \sum_{k_2=0}^{1} A(k_2, k_1, k_0) W^{j_0 k_2 4} \\ A_2(j_0, j_1, k_0) &= \sum_{k_1=0}^{1} A_1(j_0, k_1, k_0) W^{(j_1 2 j_0) k_1 2} \\ A_3(j_0, j_1, j_2) &= \sum_{k_0=0}^{1} A_2(j_0, j_1, k_0) W^{(j_2 4 j_1 2 + j_0) k_0} \\ \hat{X}(j_2, j_1, j_0) &= A_3(j_0, j_1, j_2) \end{aligned} \tag{9}$$




**Obrázek 4.2.** Vývojový diagram Cooley-Tukeyho algoritmu rychlé Fourierovy transformace pro provedení osmibodové transformace. [4]

Části přispívající ke každému součtu jsou znázorněny na obrázku číslo 4.2. Každé malé číslo označuje mocninu $W$ aplikovanou podél sousední dráhy. Poslední operací zobrazenou na obrázku číslo 4.2 je přeuspořádání. To je způsobeno bitovou inverzí v argumentech posledního vztahu v sadě rovnic (9).

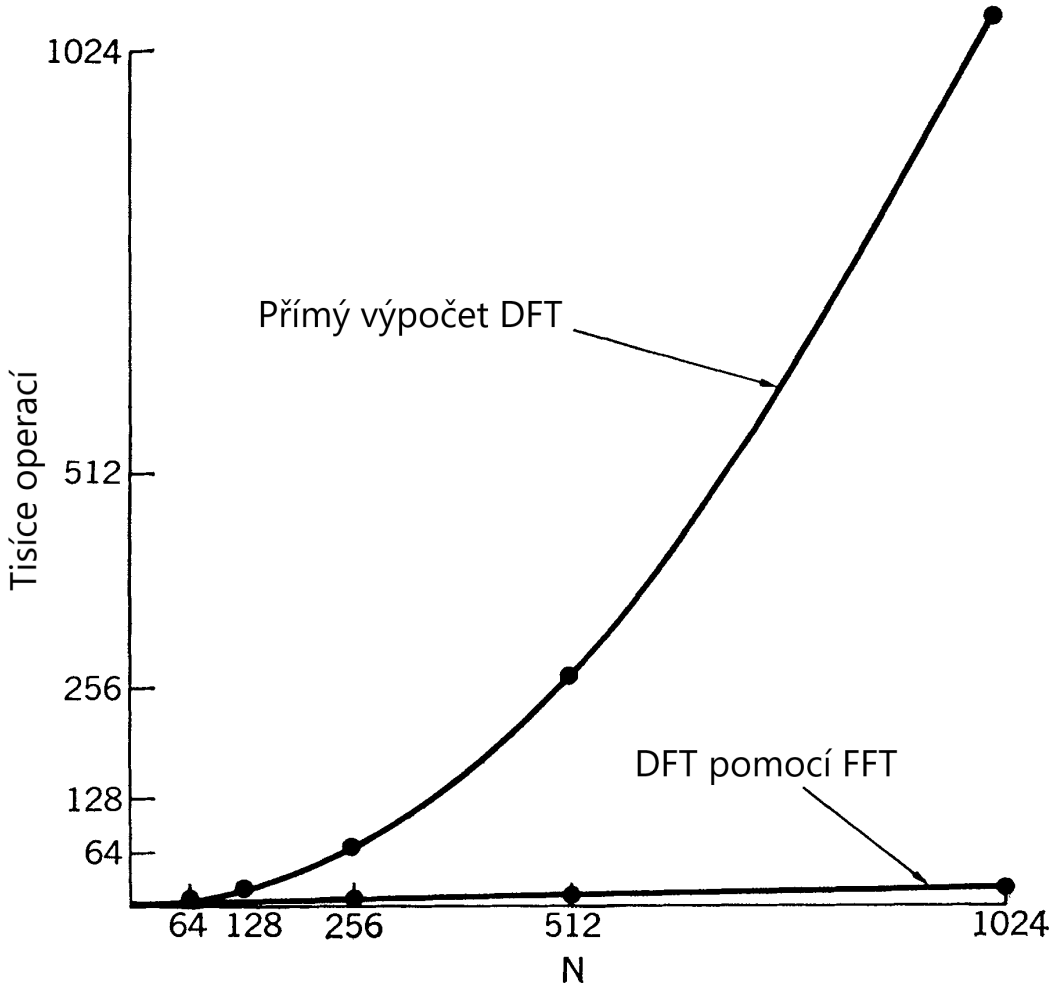
Tato sada rekurzivních rovnic představuje původní Cooleyovu-Tukeyovu formulaci algoritmu rychlé Fourierovy transformace pro $N = 8$. Ačkoli přímé vyhodnocení rovnice (4) pro $N = 8$ by vyžadovalo téměř 64 složitých operací násobení a sčítání, rovnice rychlé Fourierovy transformace zdánlivě potřebuje jen 48 operací. A když si všimneme, že první násobení v každém součtu je ve skutečnosti násobením jedničkou, stane se toto číslo pouze 24. Po postřehu, že $W_0 = -W_4$, $W_1 = -W_5$ etc., lze počet násobení snížit až na 12. Tyto redukce pokračují až k obecnějšímu případu $N = 2^m$, což snižuje výpočet z téměř $N^2$ operací na $(N/2)log_2N$ komplexních násobení, $(N/2)log_2N$ komplexních sčítání a $(N/2)log_2N$ odečítání. Pro $N = 1024$ to představuje snížení výpočtů o více než 200 na 1. Tento rozdíl je graficky znázorněn na obrázku číslo 4.3.

## 4.1 Možnosti využití FFT

Operace obvykle spojené s FFT jsou:

- Výpočet spektrogramu, tedy zobrazení krátkodobého výkonového spektra jako funkce času.
- Konvoluce dvou časových řad pro provádění digitální filtrace.
- Korelace dvou časových řad.

Přestože všechny tyto operace lze provádět bez rychlé Fourierovy transformace, její výpočetní úspory výrazně zvýšily zájem o provádění těchto operací digitálně. Na obrázku číslo 4.3 je vyobrazený rozdíl v počtu potřebných operací mezi rychlou Fourierovou transformací a diskrétní Fourierovou transformací.




**Obrázek 4.3.** Počet operací potřebných pro výpočet diskrétní Fourierovy transformace pomocí algoritmu rychlé Fourierovy transformace ve srovnání s počtem operací potřebných pro přímý výpočet diskrétní Fourierovy transformace. [4]

## 4.2 Spektrogram a waterfall spektrum

Pokud převedeme signál z časové oblasti do frekvenční, nastává otázka, jak tento výsledek zobrazit. K tomu slouží spektrogram, který je v některých případech zvaný také waterfall graf.

Spektrogram je vizuální reprezentace toho, jak se frekvenční spektrum signálu mění v čase. Jsou široce používány v oblasti hudby, lingvistiky, sonaru, radaru, zpracování řeči, seismologie a dalších. Běžným formátem spektrogramu je graf se dvěma osami, na jedné ose se zobrazuje čas a na druhé osa se zobrazuje frekvence. Třetí rozměr udávající amplitudu v konkrétním čase je reprezentován intenzitou nebo barvou každého bodu na grafu.

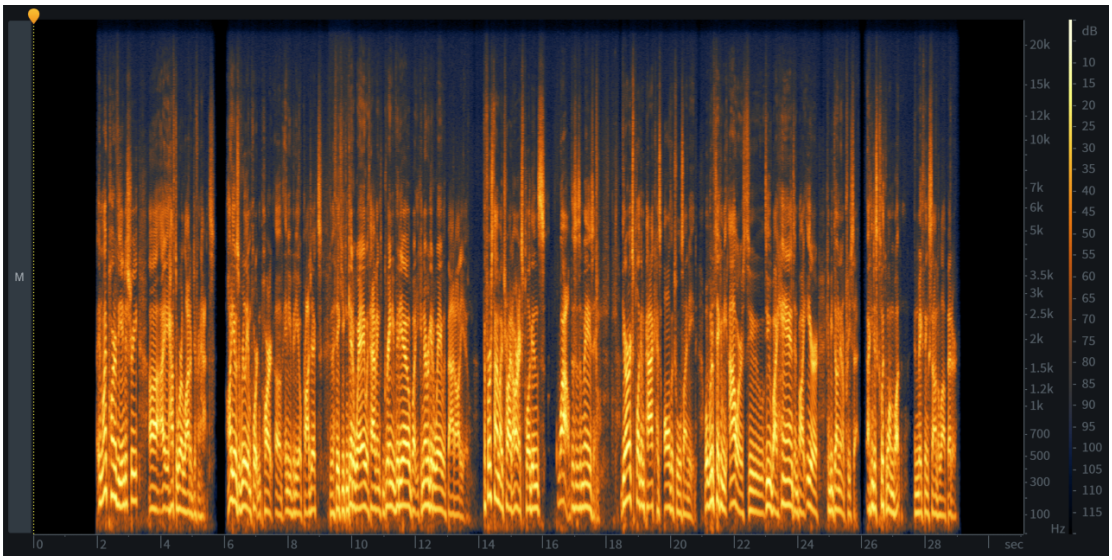
Spektrogramy mohou být vytvořeny ze signálu v časové oblasti více způsoby. Analogicky, aproximováním pomocí řady pásmových filtrů, což byl jediný způsob před nástupem moderního digitálního zpracování signálu, nebo přepočítáním pomocí Fourierovy transformace. Další způsob je vytvoření spektrogramu pomocí rychlé Fourierovy transformace. Digitálně vzorkovaná data v časové oblasti jsou rozdělena na části, které se

obvykle překrývají, a transformovány pomocí Fourierovy transformace, čímž se získá velikost frekvenčního spektra pro každou část. Každá část pak odpovídá jednomu řádku, či sloupci, v obrázku. Tyto části odpovídají velikosti spektra v závislosti na frekvenci pro určitý okamžik v čase.

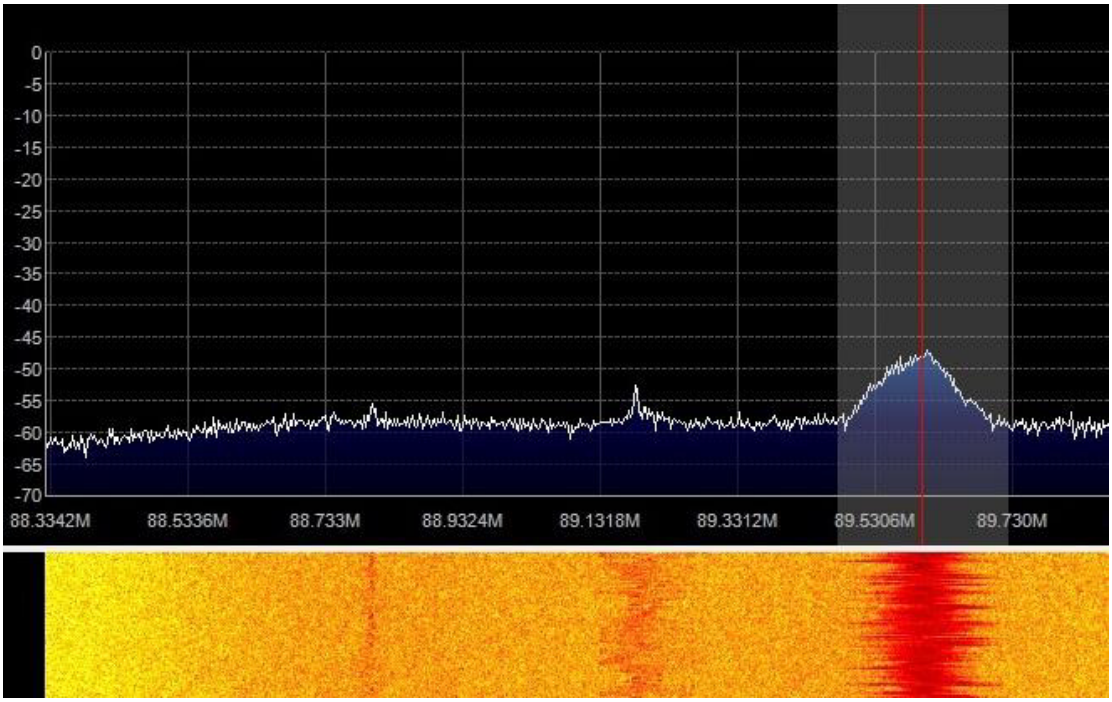
Existuje mnoho variant grafu. Často se prohazují vertikální a horizontální osy, někdy je amplituda nebo velikost reprezentována třetí osou ve 3D grafu, namísto barvy nebo intenzity. Osy frekvence a amplitudy mohou být lineární nebo logaritmické v závislosti na tom, k čemu se graf používá. Zvuk je obvykle reprezentován logaritmickou osou amplitudy v decibelech, zatímco frekvence bývá lineární pro zdůraznění harmonických vztahů nebo logaritmická pro zdůraznění hudebních, tónových vztahů. [5–6]

Tato práce se zabývá programem, který zobrazuje waterfall graf s horizontální osou frekvence, vertikální osou času a třetí osou, nebo barevnou mapou, která odpovídá amplitudě signálu.

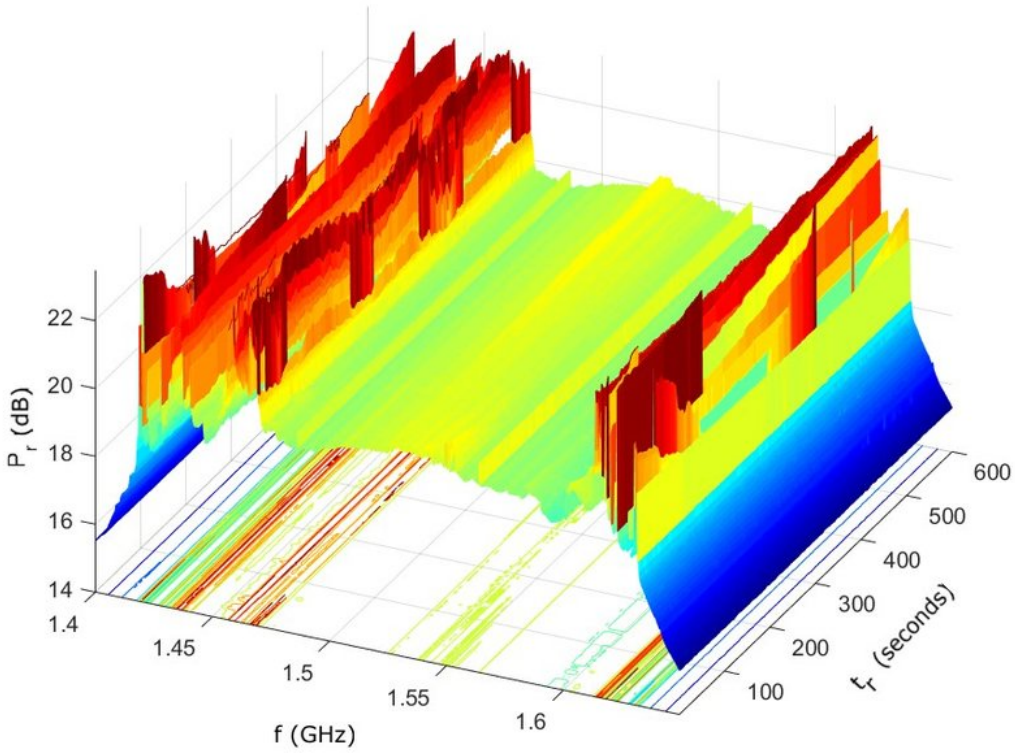
Dále jsou uvedeny některé příklady waterfall grafů:




**Obrázek 4.4.** Spektrogram mluveného slova. Na horizontální ose je čas a na vertikální frekvence. [7]




**Obrázek 4.5.** FFT spektrum signálu FM vysílání a jeho waterfall graf. Na horizontální ose je frekvence a na vertikální amplituda. [8]




**Obrázek 4.6.** Waterfall graf slunečních radiových vzplanutí. [9]

# Kapitola 5
# Programovací jazyky

Počítače jsou jedny z nejuniverzálnějších nástrojů, které máme k dispozici. Jsou schopny provádět ohromující výkony ve výpočtech, umožňují snadnou výměnu informací bez ohledu na jejich fyzické umístění, zjednodušují mnoho každodenních úkolů a umožňují nám automatizovat mnoho procesů, jejichž provádění by bylo jinak únavné nebo nudné. Počítače však nejsou „inteligentní“. Musí jim být jasně řečeno, co přesně mají dělat a právě k tomu slouží programovací jazyky. Počítače zatím samy nedokážou přijít na to, co mají dělat, a proto se spoléhají na námi vytvořené programy, což jsou sady instrukcí, kterým počítač rozumí a kterými se může řídit.

## 5.1 Dělení programovacích jazyků

V závislosti na typu projektu existuje mnoho faktorů, které je třeba vzít v úvahu při výběru jazyka. Zde je seznam některých z těch nejvýznamnějších:

### 5.1.1 Kompilované a interpretované jazyky

Kompilované jazyky jsou překládány do jazyka cílového stroje programem zvaným kompilátor. To může mít za následek velmi rychlý kód, zvláště pokud je kompilátor efektivní při optimalizaci, ale výsledný kód nemusí být dobře portován mezi operačními systémy a proces kompilace může chvíli trvat.

Interpretované jazyky jsou čteny programem zvaným interpret a tímto programem jsou vykonávány. I když jsou stejně přenosné jako jejich interpret a nemají dlouhé kompilační časy, interpretované jazyky jsou obvykle mnohem pomalejší než ekvivalentní kompilované jazyky.

A konečně, jazyky kompilované just-in-time (JIT) jsou jazyky, které se rychle kompilují, ale obvykle s velmi malou optimalizací, což nabízí rovnováhu mezi výkonem a přenositelností.

### 5.1.2 Nízkoúrovňové a vysokoúrovňové jazyky

Vysoká nebo nízká úroveň v tomto případě označuje, do jaké míry povaha jazyka zrcadlí základní systém. Jinými slovy, úroveň programovacího jazyka odkazuje na to, jak podobný jazyk je nativnímu jazyku počítače. Čím vyšší úroveň, tím méně podobná.

Nízkoúrovňový jazyk je obecně velmi podobný strojovému kódu, a proto je vhodnější pro programy jako jsou ovladače zařízení nebo velmi výkonné programy, které skutečně potřebují přístup k hardwaru. Obecně je tento termín vyhrazen pro samotný strojový kód a jazyk symbolických adres, ačkoli mnoho jazyků nabízí prvky nízké úrovně. Protože však nízkoúrovňový jazyk podléhá všem nuancím hardwaru, ke kterému přistupuje, je obecně obtížné přenést program napsaný v nízkoúrovňovém jazyce na jiné platformy. Jazyky nízké úrovně se prakticky nikdy neinterpretují, protože to obecně maří účel.

Vysokoúrovňové jazyky se zaměřují spíše na pojmy, které jsou pro lidskou mysl snadno srozumitelné, jako jsou předměty nebo matematické funkce. Vysokoúrovňové jazyky jsou

obvykle srozumitelnější než jazyky na nízké úrovni a vývoj programu v jazyce na vysoké úrovni obvykle trvá méně času než v jazyce nízké úrovně. Jako kompromis je obecně potřeba obětovat určitou míru kontroly nad tím, co výsledný program skutečně dělá. Není však nemožné kombinovat funkce na vysoké a nízké úrovni v jazyce.

## 5.2 Typový systém

Typový systém odkazuje na pravidla, kterými se musí řídit různé typy proměnných jazyka. Některé jazyky nemají typy, a proto se na ně tato část nevztahuje. Protože však většina jazyků (včetně C++) má typy, jsou tyto informace důležité.

Tyto charakteristiky typování se nemusí nutně vzájemně vylučovat a některé jazyky je míchají.

### 5.2.1 Silně a slabě typové systémy

Silný typový systém omezuje, jak lze různé typy proměnných navzájem převádět bez jakýchkoli konvertujících příkazů. Ideální systém silného typování by zakazoval implicitní „přetypování“ na typy, které nedávají žádný smysl. Slabý typový systém by se snažil najít nějaký způsob, aby převod fungoval

### 5.2.2 Explicitní typování a typové odvozování

Toto se zabývá tím, jak kompilátor/interpret pro jazyk odvozuje typy proměnných. Mnoho jazyků vyžaduje, aby byly typy proměnných explicitně definovány, a proto se spoléhají na explicitní typování. Některé však mohou odvodit typ proměnné na základě kontextu, v těchto případech se používá typové odvození.

### 5.2.3 Statická a dynamická typová kontrola

Pokud je jazyk napsán staticky, pak kompilátor/interpret provede kontrolu typu jednou před spuštěním/zkompilováním programu. Je-li jazyk dynamicky typově kontrolován, pak jsou typy kontrolovány za běhu.

### 5.2.4 Bezpečnost typování

Bezpečnost typování se týká míry do jaké jazyk zakáže operace při typování proměnných, které by mohly vést k nedefinovanému chování nebo chybám. Bezpečný jazyk udělá více pro to, aby k takovým operacím nebo konverzím nedocházelo, zatímco nebezpečný jazyk dá v tomto ohledu na uživatele větší odpovědnost.

## 5.3 Paradigmata

Programovací paradigma je metodologie nebo způsob programování, který programovací jazyk podporuje. Programovací jazyk může pochopitelně podporovat více paradigmat, poté je nazýván multiparadigmatický. Zde je shrnutí několika běžných paradigmat:

- Deklarativní – Deklarativní jazyk se zaměří spíše na specifikaci toho, co má jazyk dosáhnout, než na to, jakými prostředky toho má dosáhnout.
- Funkcionální – Funkcionální programování je podmnožinou deklarativního programování, které se snaží vyjádřit problémy pomocí matematických rovnic a funkcí. Pracuje výhradně s neměnnými objekty.
- Generické – Generické programování se zaměřuje na psaní základních algoritmů z hlediska typů, které budou specifikovány až při skutečném použití algoritmu, což

umožňuje určitou shovívavost k programátorům, kteří se chtějí vyhnout přísným silně typovým pravidlům.
- Procedurální – Program je strukturován/rozdělen do jednotlivých procedur (podprogramů, funkcí), které jsou postupně volány z hlavní procedury. Každá procedura obsahuje sled programových instrukcí, které se mají vykonat.
- Objektově-orientované – Objektově-orientované programování (někdy zkráceně OOP) je druh strukturovaného programování, které vyjadřuje programy v termínech „objektů“, které jsou určeny k modelování objektů v reálném světě. Tyto objekty mohou obsahovat jak proměnné, tak funkce nazývané metody.

## 5.4 Standardizace

To, jestli má jazyk formální standard, může být velmi důležité pro zajištění toho, že programy napsané pro práci s jedním kompilátorem/interpretem budou fungovat s jiným. Některé jazyky jsou standardizovány Americkým národním standardizačním institutem (ANSI), některé jsou standardizovány Mezinárodní organizací pro standardizaci (ISO) a některé mají neformální, ale de facto standard neudržovaný žádnou normalizační organizací. [10–11]

# Kapitola 6
# Zvolené programovací jazyky

V této kapitole budou popsány programovací jazyky, které byly vybrány pro sepsání programu. Při vytváření programu byly použity dva programovací jazyky. Přesněji řečeno se jedná o programovací jazyk Python a programovací jazyk C++. Kapitola si dává za úkol oba tyto jazyky popsat a vypsat jejich nejdůležitější vlastnosti.

## 6.1 Python

Python je volně dostupný, objektově orientovaný programovací jazyk na vysoké úrovni, s dynamickou sémantikou. Jazyk je tedy silně a dynamicky typovaný. Jazyk vytvořil v roce 1990 Guido Von Rossum ve Stichting Mathematisch Centrium v Nizozemsku. Samotný jazyk byl ve skutečnosti vyvinut velkým týmem dobrovolníků a je dovoleno ho upravovat. Ve fázi vývoje, ve které se nacházel, měl třídy s dědičností, zpracováním výjimek, funkcemi a základními datovými typy jako list, dict, str a dalšími. V květnu 2000 se Guido a vývojový tým Pythonu přestěhovali na BeOpen.com a vytvořil tým BeOpen PythonLabs. Ve stejném roce se tým PythonLabs přestěhoval do společnosti Digital Creations, která je nyní známá jako Zope Corporation.

Python 2.0 byl vydán 16. října roku 2000 s mnoha funkcemi včetně garbage collectoru, který pomáhá při problémech souvisejících se zpracováním paměti při programování. Zajímavá věc věc na Pythonu je, že je podporován komunitou jeho vývoj je velmi transparentní.

Brzy poté byl 3. prosince 2008 po dlouhé době testování vydán Python 3.0, což byla první zpětně nekompatibilní verze. Hlavní vlastnosti Pythonu 3.0 byly také zpětně portovány na zpětně kompatibilní Python 2.6 a 2.7. V této práci byl používán Python verze 3.10.

Jazyk má vysokou popularitu, kvůli zvýšené produktivitě, kterou poskytuje cyklus editace-testování-ladění, který je, ve srovnání s jinými programovacími jazyky, velmi rychlý.

Python má jednoduchou, snadno naučitelnou syntaxi, která je vybavena mnoha anglickými slovy pro snadnější čitelnost, což pomáhá zvýšit produktivitu a efektivitu. Python se snaží o to, aby byla většina symbolů blízko k anglickému jazyku, namísto odkazování na nejednoznačné symboly, které mohou být v jazyce vyžadovány, abyste se dostali k určitým funkcím. Byl navržen tak, aby fungoval na lidské úrovni, takže je čitelný a srozumitelný, bez nutnosti se ponořit do obskurního strojového kódu, hexadecimálních znaků nebo dokonce jedniček a nul.

Python se dá najít v srdci některých z nejzajímavějších a nejmodernějších technologií na světě. Je to kód, který spojuje superpočítačové algoritmy dohromady, používá se ve vesmírném průmyslu a ve vědě a inženýrství. Některé umělé inteligence, jako Alexa a Siri, Cortana a Google Assistant využívají Python pro svou výkonnou technologii rozpoznávání hlasu.

Python se dá použít k automatizaci měření a interaktivnímu zpracování dat. Jazyk je schopen pracovat s velkými databázemi a počítat velká čísla bez námahy ve srovnání

s mnoha jinými programovacími jazyky. Může být použit jako interní skriptovací jazyk, takže může být pro určité funkce spouštěn jinými programy. Po naučení se Pythonu, je možné psát komplexní software To vše dělá Python velmi univerzální jazyk.

Celkově vzato se dá Python snadno naučit, ať už jste začátečník v programování nebo zkušený v jiných jazycích. Je to rychlý, uživatelsky přívětivý, ale přesto velmi robustní a komplexní programovací jazyk. [12–15]

### 6.1.1 Výhody Pythonu

Existuje mnoho důvodů, proč je Python pro mnohé oblíbeným jazykem. Některé z nejčastějších důvodů jsou vypsány níže:

- Python je jednoduchý programovací jazyk, který se lze, při srovnání s jinými programovacími jazyky, snadno naučit. Jeho syntaxe je jednoduchá a podobná anglickému jazyku. Středník a složené závorky jsou nepoužívané, místo toho definuje blok kódu odsazení. Pro začínající programátory je to doporučený programovací jazyk.
- Python je schopen provádět složité úkoly jen s několika řádky kódu.
- Python je interpretovaný jazyk, což znamená, že každý řádek programu se provádí samostatně. Výhoda interpretovaného jazyka spočívá v tom, že ladění je jednoduché a přenosné.
- Python může fungovat na různých platformách, včetně Windows, Linux, UNIX a Macintosh. Python se tedy dá považovat za přenosný programovací jazyk a umožňuje programátorům vytvořit software pro několik konkurenčních platforem vytvořením pouze jednoho programu.
- Python je bezplatný programovací jazyk, který může používat každý. Na jeho oficiálních stránkách[1] je volně dostupný. Má rozsáhlou komunitu po celém světě, která pracuje na vytvoření nových knihoven a funkcí do Pythonu. Kdokoli může, bezplatně, získat jeho zdrojový kód.
- Python umožňuje objektově orientované programování, které zavádí pojmy o třídách a objektech. Objektově orientované metody pomáhají programátorům při psaní opakovaně použitelného kódu a vývoj aplikací s menším množstvím kódu.
- Python je rozšiřitelný. To znamená, že lze použít jiné jazyky, jako je C/C++ pro zkompilování kódu. Tím se převede program na bajtový kód, který lze spustit na libovolné platformě.
- Python nabízí rozmanitou sadu knihoven pro různé účely, včetně strojového učení, vývoj webu a skriptování. Tensor flow, Pandas, Numpy, Keras a Pytorch jsou jen některé příklady knihoven strojového učení. Webové vývojové rámce Python zahrnují Django, Flask a Pyramids.
- Pro vývoj grafického programu se dají použít knihovny pro grafická rozhraní jako PyQt, Tkinter a Kivy.
- Python se snadno integruje s jazyky jako C, C++ a JAVA.
- Python používá dynamickou alokaci paměti a není tedy nutné specifikovat datový typ proměnné. Když proměnné přiřadíme hodnotu, paměť proměnné je automaticky přiděleno za běhu. [12–15]

## 6.2 C++

C++ je multiparadigmatický, kompilovaný programovací jazyk, který vznikl jako modifikace programovacího jazyka C. V současné době patří C++ mezi nejrozšířenější

[1] `www.python.org`

programovací jazyky. C++ bylo navrženo s orientací na systémové programování a vestavěný software s omezenými zdroji a velké systémy, přičemž jeho design klade důraz na výkon, efektivitu a flexibilitu použití. C++ je také užitečné i pro jiné účely, přičemž hlavními silnými stránkami jsou softwarová infrastruktura a aplikace s omezenými zdroji, včetně desktopových aplikací, videoher, serverů (například webové vyhledávání nebo databáze) a výkonově kritické aplikace.

Programovací jazyk C++ má historii sahající až do roku 1979, kdy Bjarne Stroustrup pracoval na své Ph.D. tezi. Jedním z jazyků, se kterými měl Stroustrup možnost pracovat, byl jazyk zvaný Simula, což je, jak název napovídá, jazyk primárně určený pro simulace. Jazyk Simula 67 – což byla varianta, se kterou Stroustrup pracoval – je považován za první jazyk podporující paradigma objektově orientovaného programování. Stroustrup zjistil, že toto paradigma bylo velmi užitečné pro vývoj softwaru, nicméně jazyk Simula byl pro praktické použití příliš pomalý.

Krátce nato začal pracovat na „C with Classes“, což, jak název napovídá, mělo být nadmnožinou jazyka C. Jeho cílem bylo přidat objektově orientované programování do jazyka C, což byl a stále je jazyk dobře respektovaný pro svou přenositelnost bez obětování rychlosti nebo nízkoúrovňové funkčnosti. Jeho jazyk kromě všech funkcí jazyka C zahrnoval třídy, základní dědičnost, inline funkce, výchozí argumenty funkcí a silnou typovou kontrolu.

První kompilátor C s třídami se jmenoval Cfront, který byl odvozen od kompilátoru C nazvaného CPre. Byl to program navržený tak, aby překládal kód C s třídami do běžného C. Cfront byl později opuštěn v roce 1993 poté, co bylo obtížné do něj integrovat nové funkce, jmenovitě výjimky C++. Nicméně Cfront měl obrovský dopad na implementace budoucích kompilátorů a na operační systém Unix.

V roce 1983 byl název jazyka změněn z „C with Classes“ na C++. Operátor ++ je, v jazyce C, operátor pro inkrementaci proměnné, což poskytuje určitý pohled na to, jak Stroustrup na jazyk pohlížel. V této době bylo přidáno mnoho nových funkcí.

V roce 1985 byl zveřejněn Stroustrupův odkaz na jazyk s názvem *The C++ Programming Language*. Ve stejném roce byl C++ implementován jako komerční produkt. Jazyk v té době ještě nebyl oficiálně standardizován. Jazyk byl znovu aktualizován v roce 1989, aby zahrnoval chráněné a statické členy a také dědictví z několika tříd.

V roce 1990 byl vydán *The Annotated C++ Reference Manual*. Ve stejném roce byl vydán kompilátor Turbo C++ společnosti Borland jako komerční produkt. Turbo C++ přidalo množství dalších knihoven, které měly značný dopad na vývoj C++. Ačkoli poslední stabilní vydání Turbo C++ bylo v roce 2006, kompilátor je stále používán.

V roce 1998 vydala komise pro standardy C++ první mezinárodní standard pro C++ ISO/IEC 14882:1998, který byl neformálně známý jako C++98. V roce 2003 výbor reagoval na četné problémy, které byly hlášeny s jejich standardem z roku 1998, a odpovídajícím způsobem jej revidoval. Změněný jazyk byl nazván C++03.

V roce 2005 vydala komise pro standardy C++ technickou zprávu (nazvanou TR1), která podrobně popisuje různé funkce, které plánují přidat do nejnovějšího standardu C++. Nový standard byl neformálně nazván C++0x, protože se očekávalo, že bude vydán někdy před koncem prvního desetiletí. Je ironií, že nový standard byl vydán až v polovině roku 2011. Do té doby bylo vydáno několik technických zpráv a někteří kompilátoři začali přidávat experimentální podporu pro nové funkce.

V polovině roku 2011 byl dokončen nový standard C++ (nazvaný C++11). Projekt knihovny Boost měl značný dopad na nový standard a některé nové moduly byly odvozeny přímo z odpovídajících knihoven Boost. Standard C++11 zavedl tak ohromné množství změn a novinek, že se dá s nadsázkou říci, že C++11 je novým programovacím

jazykem. Přesto ale přes všechny tyto změny, zůstává veškerý kód vytvořený dřívějšími revizemi jazyka C++, plně funkční.

V prosinci 2014 pak byla vydána současně platná verze ISO/IEC 14882:2014 (C++14), neobsahující žádné dramatické změny, spíše jen opravy a drobná vylepšení. [16, 10, 17–18]

### 6.2.1 Výhody C++

Dále je možné vypsat, co C++ nabízí jako programovací jazyk:

- C++ je otevřený jazyk standardizovaný ISO. Na nějakou dobu neměl C++ žádný oficiální standard a byl udržován de facto standardem, nicméně od roku 1998 je C++ standardizován výborem ISO[2].
- C++ je kompilovaný jazyk. C++ se kompiluje přímo do nativního kódu stroje, což mu umožňuje být jedním z nejrychlejších jazyků na světě, pokud je optimalizován.
- C++ je silně typovaný, nebezpečný jazyk, což znamená, že kód může být neverifikovatelný, tedy že jeho bezpečnost nelze ověřit. C++ je jazyk, který očekává, že programátor bude vědět, co dělá, ale ve výsledku umožňuje neuvěřitelné množství kontroly.
- C++ podporuje jak explicitní typování, tak i typové odvozování. Od novějšího standardu C++ podporuje C++ více způsobů typování, což umožňuje flexibilitu a způsob, jak se vyhnout zdlouhavosti kódu tam, kde je to žádoucí.
- C++ podporuje statickou i dynamickou typovou kontrolu. C++ umožňuje kontrolovat převody typů buď při kompilaci, nebo za běhu, což opět nabízí další stupeň flexibility. Většina typové kontroly v C++ je však statická.
- C++ nabízí mnoho voleb paradigmatu. C++ nabízí pozoruhodnou podporu pro procedurální, generická a objektově orientovaná programovací paradigmata, přičemž je možná i řada dalších paradigmat.
- C++ je přenosný. Jako jeden z nejčastěji používaných jazyků na světě a jako otevřený jazyk má C++ širokou škálu kompilátorů, které běží na mnoha různých platformách. Kód, který používá výhradně standardní knihovnu C++, poběží na mnoha platformách jen s málem nebo žádnými změnami.
- C++ je dopředně kompatibilní s C. Protože C++ přímo stojí na jazyku C, je kompatibilní s téměř veškerým kódem C. C++ může používat knihovny C jen s několika nebo i žádnými úpravami kódu knihoven.
- C++ má velkou podporu knihoven. [11, 10]

[2] `https://www.open-std.org/jtc1/sc22`

# Kapitola 7
# Použité knihovny

Aby programátoři mohli sdílet svůj kód s ostatními programátory, tak využívají takzvané knihovny. Knihovna je kód, který řeší nějakou ucelenou funkcionalitu a obvykle obsahuje návod (dokumentaci), jak tento kód používat. Klíčové vlastnosti knihoven jsou znovupoužitelnost (můžeme je použít v různých programech) a abstrakce, což znamená, že jsou knihovny použitelné v různých programech, a že není třeba rozumět, jak knihovna funguje, pouze ji stačí využít k vyřešení konkrétního problému. Tato kapitolu bude popisovat hlavní knihovny použité jak v Pythonovských verzích programu, tak v programu napsáném v C++.

## 7.1 Python

### 7.1.1 Tkinter

Knihovna widgetů Tk pochází z programovacího jazyka Tool Command Language (Tcl). Tcl a Tk byly vytvořeny koncem 80. let Johnem Oustermanem, profesorem na Berkeley, jako snazší způsob programování inženýrských nástrojů používaných na univerzitě. Díky své rychlosti a relativní jednoduchosti Tcl/Tk rychle rostl v popularitě mezi akademickými, inženýrskými a unixovými programátory. Podobně jako samotný Python, vznikl Tcl/Tk na platformě Unix a teprve později migroval na macOS a Windows. Praktický záměr Tk a kořeny Unixu dodnes ovlivňují jeho design, a jeho jednoduchost je stále, ve srovnání s jinými sadami nástrojů, jeho hlavní předností.

Tkinter je rozhraní Pythonu pro knihovnu grafického rozhraní Tk a je součástí standardní knihovny Pythonu už od vydání verze 1.1 v roce 1994, což z něj dělá de facto knihovnu grafického rozhraní pro Python. Dokumentace pro Tkinter lze najít ve standadní dokumentaci Pythonu[1].

Programátoři Pythonu, kteří chtějí sestavit grafické rozhraní, mají několik možností sad nástrojů. Tkinter je někdy podceňovaný, protože je již vcelku zastaralý. Na druhou stranu má však mnoho výhod. Mezi tyto výhody patří například:

- Je součástí standardní knihovny. Až na pár výjimek je Tkinter k dispozici kdekoli je dostupný Python. Není potřeba nic instalovat, vytvářet virtuální prostředí, kompilovat binární soubory nebo hledat instalační balíčky na internetu. U jednoduchých projektů, které je potřeba udělat rychle, je to jasná výhoda.
- Tkinter je velice stabilní. I když se vývoj Tkinter nezastavil, je velmi pomalý. API je roky stabilní, hlavní změny jsou jen další funkce a opravy chyb. Kód používající Tkinter pravděpodobně poběží beze změn po mnoho let.
- Tkinter je pouze sada nástrojů grafického rozhraní. Na rozdíl od některých jiných knihoven grafického rozhraní, nemá Tkinter vlastní knihovnu vláken, síťový zásobník nebo API souborového systému. Spoléhá, pro takové věci, na běžné knihovny Pythonu, takže je ideální, pokud je třeba vytvořit grafické rozhraní pro už existující kód Pythonu.

[1] `https://docs.python.org/3/library/tkinter.html`

- Psaní kódu je velmi jednoduché. Tkinter je přímočarý, objektově orientovaný design grafického rozhraní. Pro naučení se programovat s Tkinterem, není třeba se učit stovky tříd widgetů.

Na druhou stranu má Tkinter i mnoho nedostatků. Mezi tyto nevýhody patří například:

- Tkinter má dnes velmi zastaralý vzhled. To se naštěstí v posledních letech zlepšilo, díky aktualizacím v samotném Tk a přidání tématických knihoven widgetů.
- Tkinter postrádá různé komplexní widgety, jako Rich Text formát textu nebo widgety pro vykreslování HTML. Naštěstí je možné některé funkce nahradit kombinací více jednoduchých widgetů.

Tkinter může být špatnou volbou pro herní uživatelské rozhraní nebo úhlednou komerční aplikaci, avšak pro datově řízené aplikace, jednoduché nástroje, konfigurační dialogy a další aplikace obchodní logiky, Tkinter nabízí vše, co je potřeba. [19–20]

### 7.1.2 Numpy

NumPy (zkrácený výraz pro numerický Python) je open source Python knihovna pro vědecké výpočetní úkony. Hlavní účel knihovny NumPy je práce s poli a maticemi. Knihovna obsahuje dlouhý seznam užitečných matematických funkcí, včetně některých funkcí pro lineární algebru, Fourierovy transformace a generování náhodných čísel. Pro výpočet lineární algebry využívá NumPy, buď knihovnu lineární algebry LAPACK, pokud je nainstalována v systému, nebo svojí vlastní implementaci. LAPACK je známá knihovna, původně napsána ve Fortranu, kterou využívá i MATLAB.

NumPy je založen na svém předchůdci Numeric. Numeric byl poprvé vydán v roce 1995 a nyní je už zastaralý a neprochází dalším vývojem. Numeric ani NumPy, z různých důvodů, nejsou ve standardní knihovně Pythonu.

V roce 2001 řada lidí inspirovaných Numericem vytvořila SciPy, open source vědeckou výpočetní knihovnu, která poskytuje funkce podobné funkcím MATLAB, Maple a Matematica. Poté byl vytvořen Numarray jako alternativa k Numeric. Knihovna Numarray je už nyní také zastaralá. Numarray byla v některých oblastech lepší než Numeric, ale fungovala to velmi odlišně. Z toho důvodu SciPy zachoval svojí závislost na filozofii Numeric a objektech Numeric array.

V roce 2005 se Travis Oliphant, první přispěvatel do SciPy, rozhodl s touto situací něco udělat. Pokusil se integrovat některé funkce Numarray do Numeric. Došlo ke kompletnímu přepsání, které vyvrcholilo vydáním knihovny NumPy 1.0 v roce 2006. V té době měl NumPy jak všechny funkce Numeric a Numarray, tak i mnoho dalších.

Původně byl kód NumPy součástí SciPy. Později byl však oddělen a nyní je NumPY samostatná knihovna, která je ale stále využívána knihovnou SciPy pro zpracování polí a matic.

Pokud se využije kód NumPy a samotný Python k dosažení stejného výsledku, ukáže se, že kód NumPy je mnohem čistší než kód Pythonu. Je třeba méně smyček, protože operace pracují přímo na polích a maticích. Knihovna také usnadňuje práci, protože obsahuje mnoho matematických funkcí. Základní algoritmy obstály ve zkoušce času a byly navrženy s vysokou úrovní výkonu.

Pole NumPy jsou uloženy efektivněji než ekvivalentní datové struktury v základním Pythonu. Vytváření i čtení polí je také výrazně rychlejší. Zlepšení výkonu je úměrné velikosti pole. U velkých polí se opravdu vyplatí použít NumPy. Soubory velké až několik terabajtů lze paměťově mapovat na pole, což vede k optimálnímu čtení a zápisu dat.

Nevýhodou polí NumPy je, že jsou specializovanější než prosté Pythonovské seznamy. Mimo kontext numerických výpočtů jsou pole NumPy daleko méně užitečné.

Velká část kódu NumPy je napsána v programovacím jazyku C. Díky čemuž je NumPy rychlejší než čistý kód Pythonu. Existuje také NumPy C API, která umožňuje další rozšíření funkčnosti pomocí jazyka C. [21–22]

### 7.1.3 Matplotlib

Matplotlib je komplexní vykreslovací knihovna pro vytváření statických, animovaných a interaktivních vizualizací v Pythonu. Byla vytvořena pro programovací jazyk Python a jeho numerickou matematickou nadstavbu NumPy. Poskytuje objektově orientované API (Application Programming Interface) pro vkládání grafů do aplikací pomocí uživatelských rozhraní. Matplotlib byl původně napsán jako open source alternativa pro MATLAB. Ale v dnešní době je jeho objektovně orientované API přizpůsobitelnější a výkonnější než jeho starší modul pyplot, který poskytuje rozhraní podobné MATLABu. Matplotlib byl navržen tak, aby byl stejně účelný jako MATLAB, s možností použití pomocí Pythonu a výhodou, že je zdarma a open source.

Matplotlib podporuje interaktivní a neinteraktivní vykreslování a umí ukládat obrázky v několika výstupních formátech , jako png, jpg, ps a další. Může používat více sad nástrojů pro grafické rozhraní a poskytuje širokou škálu typů grafů (spojnicové, sloupcové, kruhové diagramy, histogramy a mnoho dalších). Zároveň je vysoce přizpůsobitelný, flexibilní a snadno použitelný. Matplotlib má možnost být použitý jak v interaktivním, tak i v neinteraktivní skriptech. Lze jej použít ve skriptech bez grafického rozhraní, vložené do grafických aplikací nebo na webové stránky. [23–24]

Další přednosti Matplotlibu jsou:

- Je součástí skutečného programovacího jazyka. Zatímco jazyk MATLAB, přesto, že je Turingovsky kompletní, postrádá mnoho rysů univerzálního programovacího jazyka jako je Python.
- Je velmi přizpůsobitelný a rozšiřitelný. Matplotlib se hodí skoro pro každý účel možného použití protože má mnoho typů grafů, funkcí a možností konfigurace.
- Je integrován s jazykem LaTeX, což je velmi užitečné při psaní vědeckých prací.
- Je multiplatformní a přenosný. Matplotlib může běžet například na Linuxu, Windows nebo Mac OS X, protože Python lze spustit téměř na každé dostupné architektuře.

Na druhou stranu asi největší nevýhodou Matplotlibu je jeho rychlost. Při dělání čehokoli, co vyžaduje rychlé aktualizace grafu nebo interaktivitu v reálném čase, Matplotlib není tou nejlepší volbou.

### 7.1.4 PyQt

PyQt byl poprvé vydán v roce 1998 a od té doby se neustále vyvíjí. Knihovnu spravuje společnost Riverbank Computing Limited. Dosud je PyQt nejstabilnější a nejrozšířenější knihovnou propojující Python s Qt.

PyQt zahrnuje třídy, které pokrývají grafická uživatelská rozhraní, ale také zpracování XML, síťovou komunikaci, regulární výrazy, vlákna, databáze SQL, multimédia, procházení webu a další technologie dostupné v Qt. Na rozdíl od samotného Qt je knohovna PyQt licencovaná pod GNU GPL v3, která, stručně řečeno, vyžaduje, aby programy napsané s použitím PyQt byly šířeny pod stejnou licencí a se zdrojovým kódem. [25]

Samotná knihovna Qt je detailněji popsána v sekci 7.2.1.

### 7.1.5 PyQtGraph

PyQtGraph je čistě Pythonovská knihovna pro grafické zobrazení postavená na PyQt (případně PySide) a NumPy. Je určena pro použití v matematických, vědeckých a

inženýrských aplikacích. Navzdory tomu, že je celá napsána v Pythonu, je knihovna velmi rychlá díky velkému využití NumPy pro výpočty a frameworku Qt GraphicsView pro rychlé zobrazení. grafiky. PyQtGraph je distribuován pod open source licencí MIT.

Jedna z výhod PyQtGraph je, že obsahuje množství dodatečných funkcí. PyQtGraph je mnohem víc než knihovna pro vykreslování, knihovna se snaží pokrýt mnoho aspektů vývoje vědeckých či inženýrských aplikací pomocí pokročilejších funkcí, jako jsou analytické nástroje ImageView a ScatterPlotWidget, segmentování dat na základě ROI, stromy parametrů, vývojové diagramy, víceprocesorové zpracování a další. Další výhodou je přenosnost, protože je knihovna čistě Pythonovská, což znamená, že běží prakticky na každé platformě, která podporuje NumPy a PyQt. [26]

### 7.1.6 PyInstaller

PyInstaller je knihovna pro zabalení Pythonovského kódu se všemi jeho závislostmi a moduly do jednoho balíčku. Díky tomu může uživatel spustit zabalenou aplikaci bez instalace interpretu Python nebo jakýchkoliv knihoven. PyInstaller podporuje verzi Pythonu 3.7 a novější a správně zabaluje mnoho známých knihoven Pythonu, jako je NumPy, Matplotlib, PyQt, wxPython a další.

PyInstaller je testován na Windows, MacOS X a Linuxu. Nejedná se však o univerzální kompilátor, k vytvoření aplikace pro Windows, je třeba spustit PyInstaller na Windows a k vytvoření linuxové aplikace ji spustit na Linuxu a tak dále.

Pyinstaller pracuje tak, že zanalyzuje kód, aby objevil každou knihovnu, kterou skript potřebuje ke spuštění. Poté shromáždí kopie všech těchto souborů, včetně aktivního interpretu Pythonu, a umístí je se skriptem do jedné složky nebo volitelně do jednoho spustitelného souboru.

PyInstaller je úspěšně používán s AIX, Solaris, FreeBSD a OpenBSD, ale vývojový tým neposkytuje žádnou záruku, protože veškerý kód pro tyto platformy pochází z externích příspěvků. [27]

## 7.2 C++

### 7.2.1 Qt

Qt je aplikační vývojový framework pro vývoj aplikací s grafickým uživatelským rozhraním, umožňující sestavení aplikace pro více různých platforem s využitím jediné společné kódové základny.

Qr byl vytvořen v roce 1991 Norskou společnost Trolltech. V roce 2008 jej získala Nokia, která jej používala pro mobilní operační systém Symbian. Symbian později podlehl Androidu, takže Nokia přešla na Microsoft Windows Phone a prodala Qt společnosti s názvem Digia. V současné době je Qt ve vlastnictví The Qt Company, bývalé dceřiné společnosti Digia.

V Qt frameworku je možné, kromě aplikací s grafickým uživatelským rozhraním, vyvíjet i konzolové aplikace či služby využívající jen jeho jádro případně jej využívat pouze jako nástroj pro tvorbu uživatelských rozhraní již existujících aplikací.

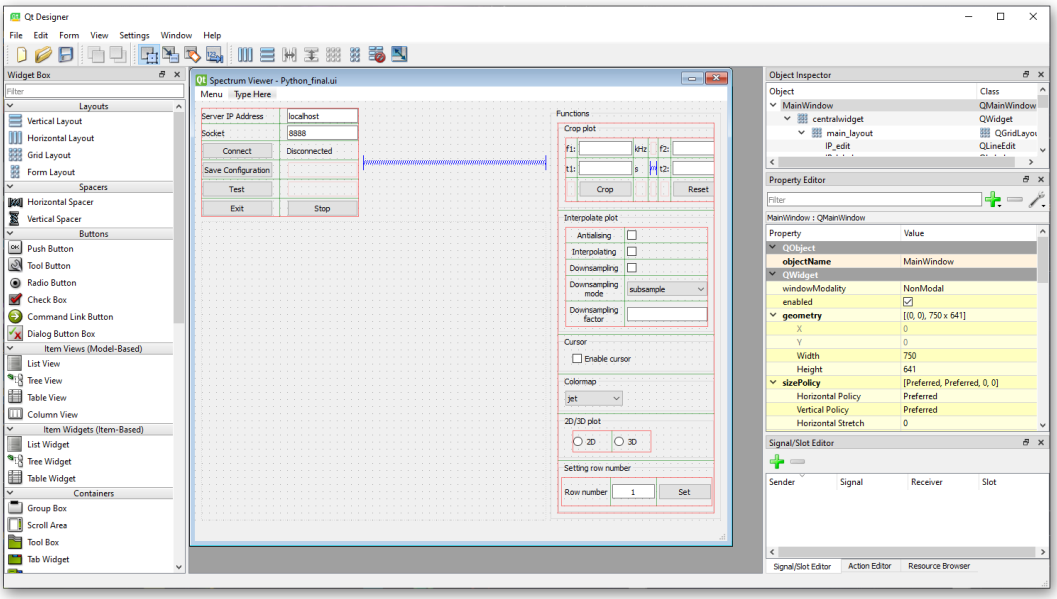
Qt obsahuje širokou kolekci mnoha různých tříd, funkcí a dalších datových struktur, které se nabízí pro použití při tvorbě vlastních aplikací. K tvorbě uživatelského rozhraní obsahuje Qt nepřebernou škálu přizpůsobitelných widgetů. Mimo to má vestavěnou podporu pro databáze, sítě, rastrovou i vektorovou grafiku, zpracování zvuku a videa a tak dále.

Pro vývoj uživatelského rozhraní Qt, a zejména interakci s ním, využívá mechanismus signálu a slotu, který zajišťuje snadnou komunikaci mezi jednotlivými objekty daného uživatelského rozhraní. Objekty mohou komunikovat i mnoha jinými způsoby, mechanismus signálu a slotu má však tu výhodu, že je nezávislý na rozhraní jednotlivých objektů.

Pokaždé, když nějaký objekt změní svůj stav, reaguje na tuto situaci vysláním signálu. Signálová zpráva slouží k notifikaci ostatních objektů, které jsou určeny jako příjemci tohoto signálu. Vyslání signálu aktivuje sloty, což jsou většinou metody daných tříd, které signál obslouží. K určitému signálu může být přiřazeno i více slotů než jeden.

Qt je k dispozici pod různými licencemi. Společnost Qt prodává komerční licenci, ale Qt je také k dispozici jako bezplatný software pod několika verzemi GPL a LGPL.

Qt nejen poskytuje výkonnou sadu nástrojů uživatelského rozhraní nazvanou Qt Designer, která umožňuje navrhovat uživatelské rozhraní bez psaní jediného řádku kódu, ale také umožňuje přizpůsobit si komponenty uživatelského rozhraní prostřednictvím jednoduchého skriptovací jazyka Qt Style Sheets. Jak Qt Desginer vypadá můžeme vidět na obrázku číslo 7.1. [28]




**Obrázek 7.1.** Vzhled softwaru, pro tvoření uživatelského prostředí, Qt Designer.

Velkou komponentou Qt frameworku je vývojové prostředí Qt Creator, v němž jsou integrovány veškeré nástroje potřebné pro kompletní vývoj aplikace, od editace zdrojového kódu, včetně funkce automatického dokončování, přes jeho překlad a ladění, až po možnost vizuálního návrhu grafického uživatelského rozhraní pomocí Qt Designeru.

### 7.2.2 QCustomPlot

Framework Qt obsahuje nástroje pro zobrazení jednoduchých grafů, ale jejich schopnosti jsou ve srovnání s jinými vyspělejšími knihovnami značně omezené. Jedna z vhodných knihoven je QCustomPlot. QCustomPlot je jednoduchá knihovna Qt podporující 2D grafy s možností exportování do různých formátů, jako PDF nebo PNG, JPEG a BMP.

Knihovna QCustomPlot nemá žádné další závislosti a je dobře zdokumentovaná. Díky vysokému výkonu při vykreslování je knihovna vhodná pro vykreslování v reálném čase,

ale zároveň dokáže knihovna vytvářet vysoce kvalitní grafy. Navíc všechny třídy této knihovny jsou zabaleny do dvou souborů, hlavičkového souboru *qcustomplot.h* a implementačního souboru *qcustomplot.cpp*. Proto je použití QCustomPlot velmi jednoduché. Knihovna QCustomPlot je licencovaná pod licencí GPL. [29]

# Kapitola 8
# Popis programu pro zobrazení Waterfall spektra

Cílem této práce bylo vytvořit program, který dokáže zobrazovat waterfall spektrum na počítači. V této kapitole bude popsáno, jaké byly požadavky a jestli byly splněny a bude popsán celý program.

Program byl primárně vytvořený v programovacím jazyku Python, pomocí knihoven PyQt a Pyqtgraph, ale byla vytvořena i zkušební verze v programovacím jazyku C++. Tato druhá verze byla vytvořena pomocí knihoven Qt a QCustomPlot. Dokáže přijímat data a zobrazovat je, stejně jako verze v Pythonu, ale chybí jí některé funkce, jako kurzor nebo možnost ukládání a otevírání souborů. Vhled všech verzí programů lze vidě na obrázcích číslo 9.1, 9.2 a 9.3.

## 8.1 Požadavky

Požadavek byl vytvořit program pro zobrazovaní přijímaných dat z SDR přijímače a zobrazovat je v grafu. Přes TCP/IP přicházejí data z SDR přijímače. Tyto data jsou již částečně zpracované a úkolem bylo je v reálném čase zobrazit a přidat různé funkce pro snadnější prohlížení spektra a manipulaci s ním. Všechny jednotlivé funkce budou popsány dále v kapitole.

Přesné požadavky na program byly:

- Program musí data přijímat.
- Program musí umět spektrum ukládat a uložené spektrum prohlížet.
- Program musí být schopný pracovat na běžném PC v reálném čase.
- Program musí být schopný zobrazit spektrum o maximální délce minimálně 16384.
- Program musí zvládat zobrazovat data s rychlostí 100 řádků za sekundu.
- Program musí umět provádět decimaci a interpolaci spektra.
- Uživatelské rozhraní programu musí podporovat nastavení barevného profilu, amplitudového měřítka, zobrazit výřez a další funkce

## 8.2 Popis programu

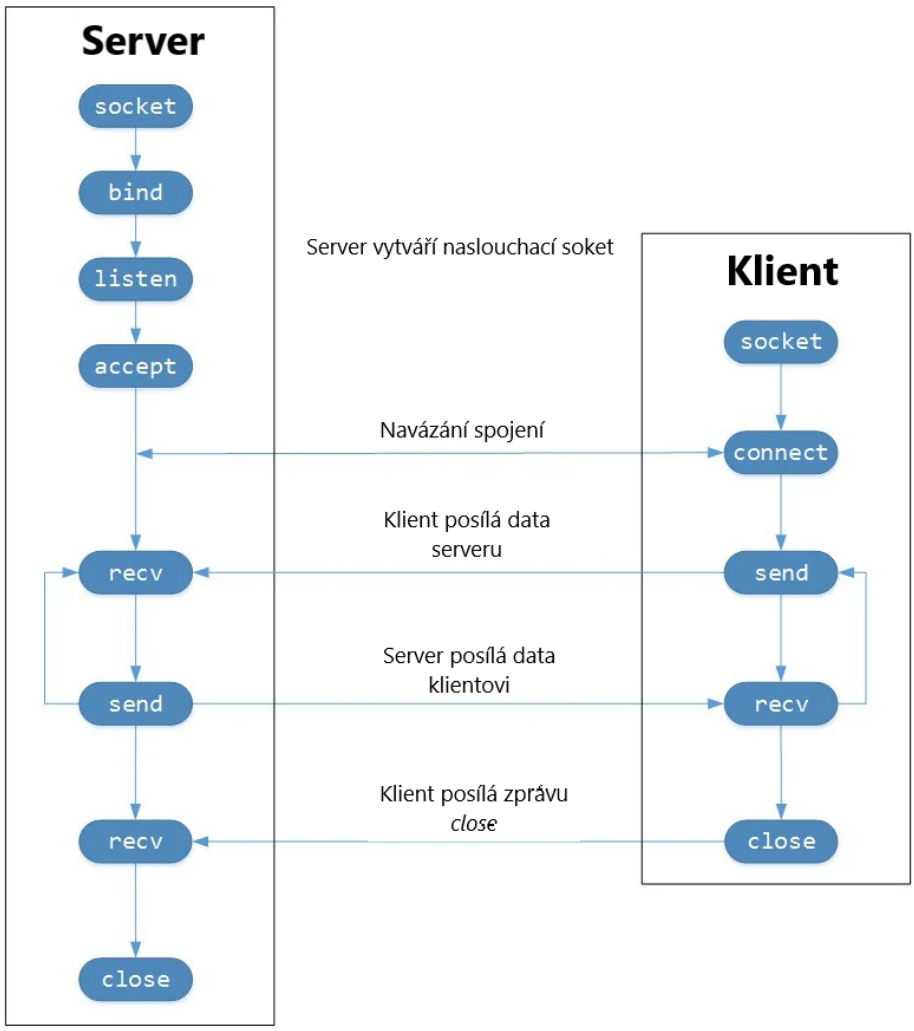
Program byl primárně vytvořený v programovacím jazyku Python, pomocí knihoven PyQt a Pyqtgraph, ale byla vytvořeny i další zkušební verze jak v jazyce Python, tak v programovacím jazyku C++. Druhá verze programu v Pythonu byla vytvořena pomocí knihoven Tkinter a Matplotlib a verze programu v C++ byla vytvořena pomocí Qt a QCustomPlot. Zatímco obě verze v Pythonu mají všechny požadované funkce, verze v C++ sice dokáže přijímat a zobrazovat data, stejně jako verze v Pythonu, ale chybí jí některé funkce, jako je kurzor nebo možnost ukládání a otevírání souborů. Všechny verze programu a jejich vyhodnocení budou popsány dále v kapitole.

## 8.3 Funkce

Částí práce bylo vytvoření různých funkcí v ptogramu. V této sekci budou vysvětleny všechny důležité funkce, které byly do programu implementovány. Zmíněné funkce budou vysvětleny tak, jak fungují v hlavní verzi programu, tedy ve verzi vytvořené v Pythonu, za pomocí PyQt a Pyqtgraph. Některé funkce tady můžou v jiných verzích vypadat trochu jinak a jak již bylo řečeno, ve verzi programu v C++ některé funkce chybí.

### 8.3.1 Příjem signálu

V první řadě musí mít program způsob, jak signál přijmout. Příjem a převod dat do zobrazitelné podoby byl specifikován vedoucím práce. Jak již bylo řečeno signál přichází přes TCP/IP. Po vyplnění IP adresy a soketu serveru, odkud přicházejí data, stačí stisknout tlačítko `Connect`. Tímto se spustí funkce, která se pomocí modulu `Socket` s tímto serverem spojí. Pomocí příkazu `socket` byl vytvořen objekt soketu a určen typ soketu jako `SOCK_STREAM`. Díky čemuž se určí výchozí protokol jako TCP. Na obrázku číslo 8.1 lze vidět digram popisující funkce modulu `Socket`. [30–31]




**Obrázek 8.1.** Vývojový diagram TCP soketu. [30]

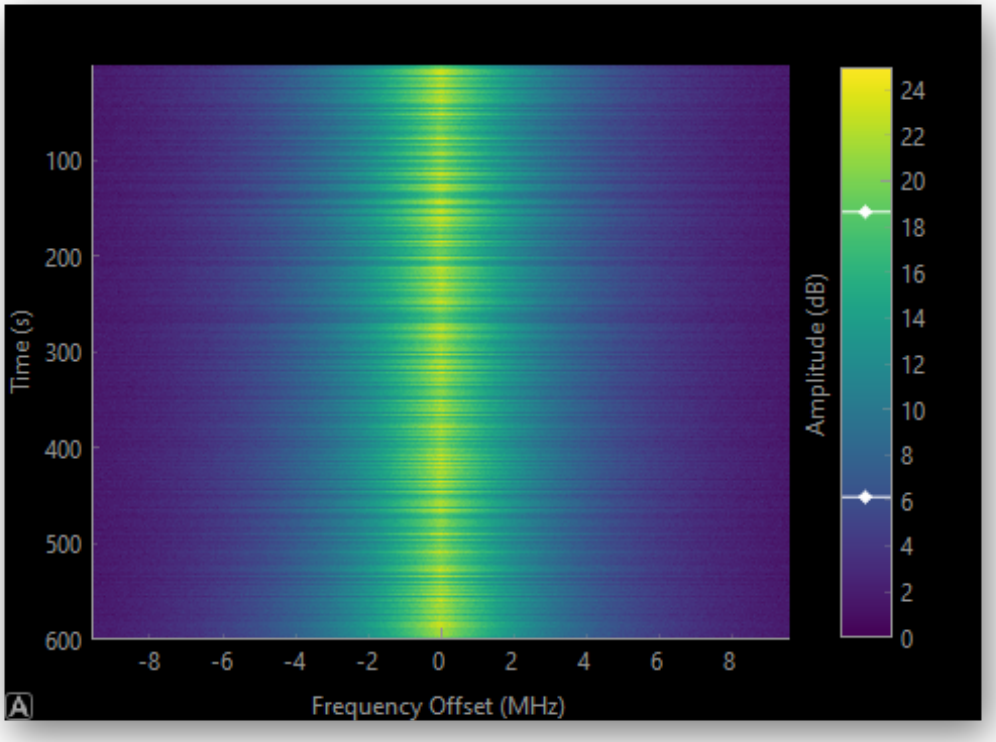
Na začátku API zavolá server, aby nastavil „naslouchací“ `listen` soket. Tento soket naslouchá pro příchozí spojení od klienta. Když se klient připojí, server zavolá `accept` k přijetí připojení. Klient zavolá `connect` k navázání spojení se serverem. Data se vyměňují mezi klientem a serverem pomocí volání `send` a `recv`. [30–31]

Poté, co se spojí program se serverem, zavolá se funkce, která bude v určeném časovém intervalu přijímat data. Jako první se najde začátek dat a zkontroluje se, že přišly data správné délky. Až poté se můžou začít zpracovávat a vykreslovat. Před vykreslením se provede FFT shift, což je matematická operace, která posune nulové frekvenční složky signálu do středu spektra.

### 8.3.2 Zobrazování grafu

První se vytvoří okno grafického prostředí, do kterého se bude graf vykreslovat. Data přicházejí po řádcích, které se postupně skládají do matice. Příchozí data by měly mít délku rovnající se mocnině dvou, protože by data měly, už před příchodem, projít rychlou Fourierovou transformací. Tyto data se poté vykreslí jako obrázek, s tím, že každý bod má nějakou hodnotu, kterou mi dává amplituda v tom bodě. Tuto velikost zobrazuji jako barvu v dané barevné mapě.

Jak takový graf vypadá můžeme vidět na obrázku číslo 8.2.




**Obrázek 8.2.** Fiktivní waterfall graf s náhodným hodnotami.

### 8.3.3 Nastavení parametrů

Důležitá věc, která se musí vyřešit před přijímáním a vykreslováním dat, je znát vlastnosti přicházejícího signálu. Program dává možnost otevřít okno `Options`, ve kterém se dají nastavit všechny důležité parametry. Tyto parametry jsou:

- Délka spektra - délka značí to, kolik bodů má každý přijatý řádek. Tato hodnota musí být mocnina dvou.
- Hloubka spektra - hloubka značí to, kolik řádků spektra zobrazuji.
- Offset frekvence - říká, jakým hodnotám frekvence přicházející data odpovídají. Tedy, pokud se offset frekvence rovná 9600 kHz, předpokládá se, že přijatá data jsou v intervalu od −9600 kHz do 9600 kHz.
- Vzorkovací perioda - určuje, jak často se data přijímají.
- Minimální a maximální amplituda - určuje, minimální a maximální hodnoty amplitudy přijímaného signálu. Tato informace je užitečná, kvůli správnému vykreslení barevné mapy.

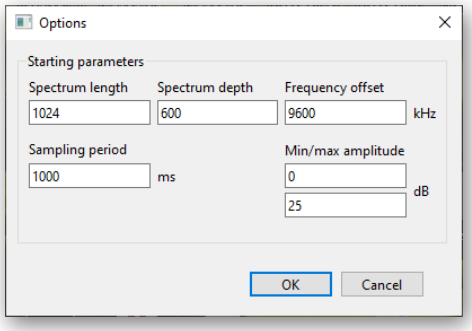
Všechny tyto parametry je třeba nastavit, ještě před tím, než začnu přijímat data, jinak nebude graf ukazovat správné hodnoty.

### 8.3.4 Výřez grafu

Další funkcí je výřez grafu. Ořezávání jen omezí zobrazenou část grafu v daných limitech. Tyto limity mohou být jak v ose frekvence, tak v ose času. Jak se tyto nastavují v programu lze vidět na obrázku číslo 8.3.

Program je interaktivní, což znamená, že má program možnost graf přibližovat nebo posouvat pomocí počítačové myši.

Po kliknutí tlačítka `Reset` se celý graf vrátí do původní podoby.




**Obrázek 8.3.** Funkce ořezávání grafu.

### 8.3.5 Interpolace spektra

Jelikož se data vykreslují do grafu jako obrázek, lze vcelku jednoduše graf interpolovat.

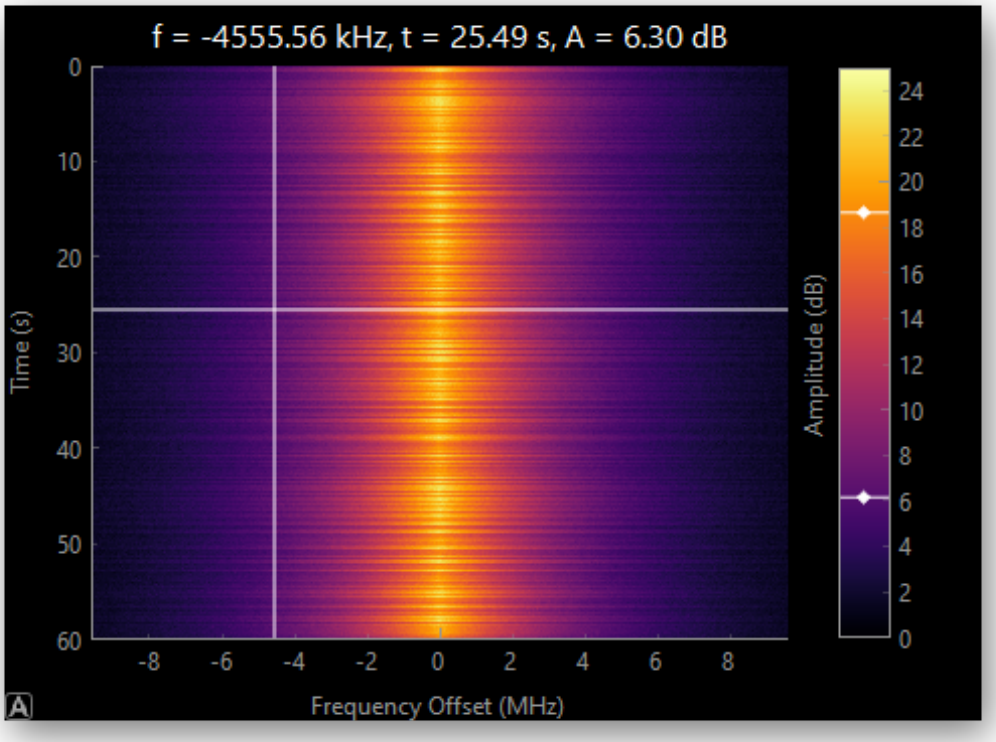
Obrázky jsou reprezentovány samostatnými pixely, buď vykreslené na obrazovce, nebo v souboru. Když data, která tvoří obrázek, mají jiné rozlišení než jejich zobrazení na obrazovce, může docházet k aliasingovým efektům. Při podvzorkování dat je aliasing redukován nejprve vyhlazováním a poté podvzorkováním vyhlazených dat.

V programu je možnost spustit jak antialising, tak interpolaci. Interpolace celý obrázek vyhladí, což způsobí, že při přiblížení nebudou vidět jednotlivé pixely dat, ale data se budou zdát spojitá.

### 8.3.6 Kurzor

Kurzor, po spuštění, jednoduše vytvoří horizontální a vertikální čáry, které se střetávají v místě, kam ukazuje kurzor počítačové myši. Informace o datech z tohoto místa grafu se pak ukáží nad grafem. Mezi tyto informace patří frekvence, čas a amplituda.

Na obrázku číslo 8.4 můžeme vidět jak takový kurzor vypadá.




**Obrázek 8.4.** Ukázka kurzoru.

### 8.3.7 Barevná mapa

Funkce barevné mapy jen specifikuje použité barvy v barevné mapě z daných předpřipravených barevných map. Smyslem výběru barevné mapy je nastavit dobrou a čitelnou reprezentaci amplitudy v grafu. To, kterou barevnou mapu si uživatel vybere závisí jen na osobní preferenci. Bravené mapy jsou k nalezení ve virtuálním prostředí programu, přesněji ve složce */pyqtgraph/colors/maps*, ve formátu *CSV*. To znamená, že je možné přidat další další barevné mapy, pokud jsou ve stejném formátu. Takové mapy se dají sehnat například z projektu *Colorcet*[1]. Program je sám při spuštění najde zařadí do seznamu možných barevných map.

Na obrázku číslo 8.5 jsou uvedeny některé barevné mapy, které má program k dispozici.

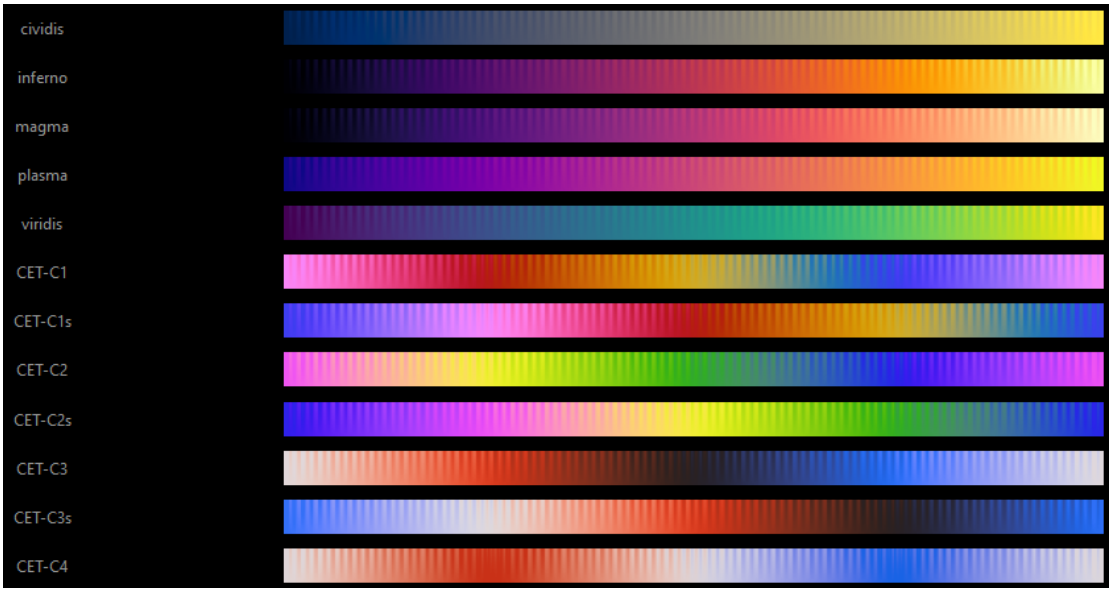
### 8.3.8 Dimenze grafu

V programu je doplněná i funkce pro nastavení, v jakých dimenzích se graf bude vykreslovat. V případě dvou dimenzí je graf vykreslován se dvěma osami, tedy s osami frekvence a času, a amplituda je daná barevnou mapou. V případě tří dimenzí je amplituda vykreslená ve třetí ose.

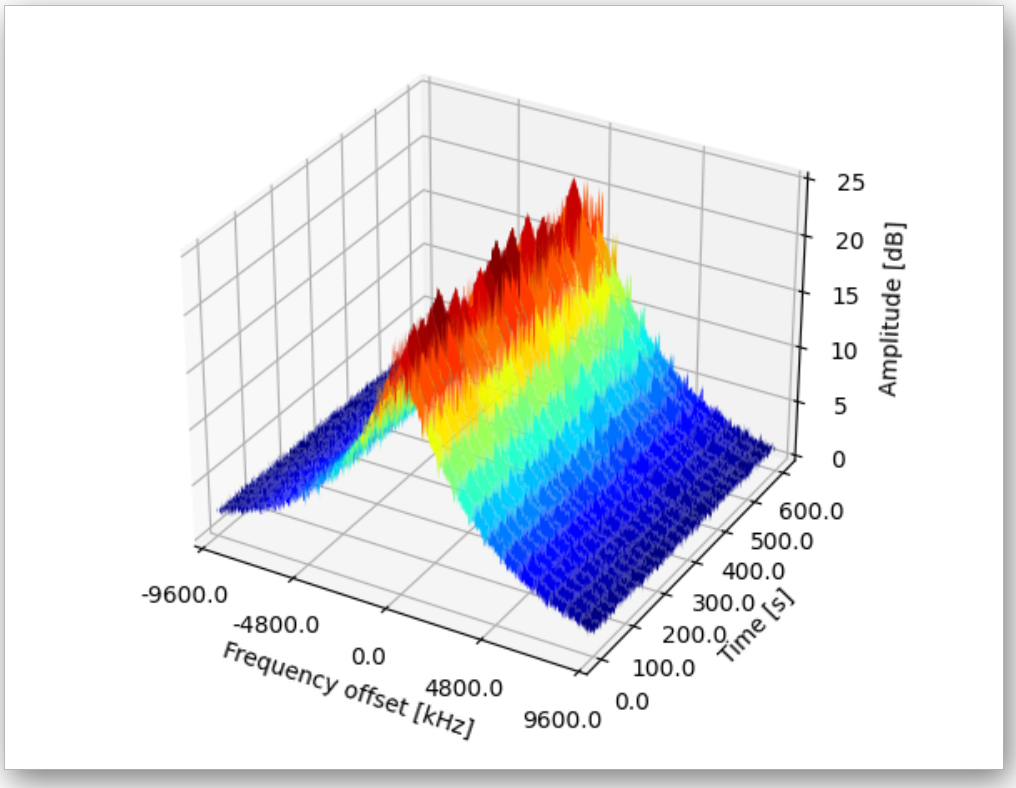
Vykreslování grafu se třemi dimenzemi je časově náročnější, proto je doporučeno použít větší vzorkovací periodu nebo zvýšit hodnotu `Number of drawn rows`, která je popsána v sekci 8.3.9.

Na obrázku číslo 8.6 můžeme vidět, jak vypadá graf se třemi dimenzemi.

[1] `https://colorcet.com/index.html`




**Obrázek 8.5.** Ukázka barevných map.




**Obrázek 8.6.** Ukázka grafu se třemi dimenzemi.

### 8.3.9 Počet vykreslených řádků

Další funkce je v programu popsána jako `Number of drawn rows`. Tato hodnota udává to, jak často se graf vykresluje. Zatímco vzorkovací perioda říká, jak často se data přijímají, `Number of drawn rows` říká, jak po kolika přijatých řádcích se mají data vykres-

lit. To znamená, že pokud je vzorkovací perioda 1 sekunda a `Number of drawn rows` má hodnotu 10, vykreslí se data každých 10 přijatých řádků, tedy každých 10 sekund.

Tato funkce existuje z jediného důvodu, a to z takového, že přijímání dat a skládání je do matice je daleko rychlejší než tyto data vykreslovat. Zvýšením `Number of drawn rows`, se tedy bude méně zatěžovat výkon počítače a může dát možnost snížení hodnoty vzorkovací periody.

### 8.3.10 Ukládání

Přes funkci ukládání se dá graf uložit jako obrázek nebo rovnou jako soubor dat. Po kliknutí na `Menu` v horní liště programu se otevře výběr, ve kterém je možnost `Save as ...`.

Program dokáže, buď ukládat graf jako obrázek v několika formátech, výslovně: Jpg, png a svg, nebo ukládat graf přímo jako soubor hodnot ve formátu csv. Při výběru csv se data uloží i s hlavičkou, ve které jsou uložené parametry aktuálně zobrazených dat.

### 8.3.11 Otevírání dat

Další funkce programu je otevírání dat ve formátu csv. Data by měla být uložená s hlavičkou. Pokud tedy data otevřu načtou se i parametry grafu, potřebné k vytvoření správného grafu.

### 8.3.12 Ukládání konfigurace

Program má také možnost uložení nastavených parametrů do souboru `INI`, což je konfigurační textový soubor. Pokaždé, když se program spustí se všechny data ze souboru automaticky načtou. Mimo parametrů se do souboru ukládají i informace pro `Socket`. Tedy IP adresa a adresa soketu.

### 8.3.13 Testovací verze

Poslední důležitá funkce je testovací verze programu. Program má v sobě zabudovaný kód, který dokáže vytvářet fiktivní data a zapisovat je do grafu, bez nutnosti přijímat reálná data z SDR přijímače. Po stisknutí tlačítka `Testing version` ve hlavní liště programu, se vytvoří tlačítko `Test`. Toto tlačítko funguje obdobně jako tlačítko `Connect`, program jen začne vykreslovat fiktivní data, daná nastavenými parametry, místo dat reálných.

Tato funkce existuje z důvodu možnosti testovat program, bez přístupu k reálným datům přes TCP/IP

## 8.4 Možnosti využití programu

Program dokáže zobrazovat již zpracované amplitudové spektrum v reálném čase a je tedy schopný přijímat data z různých zdrojů. Program je určený k zobrazení spektra různých radiových, komunikačních signálů nebo i slunečních radiových vzplanutí, ale protože data přicházejí přes TCP/IP je možné přijímat i jiná data ve stejném formátu. Tato sekce vypisuje některé možnosti, ke kterým by šel program využít.

### 8.4.1 Sledování slunečních erupcí

Sluneční erupce a vzplanutí představují nejsilnější procesy uvolňování energie v celé Sluneční Soustavě. Poruchy spojené s těmito jevy, jako jsou magnetická mračna, paprsky urychlených částice, zesílené záření (v rentgenových, UV a rádiových pásmech) šířící se ze Slunce přes meziplanetární prostor, jsou nyní známé jako efekty vesmírného

počasí. V případě dosažení Země a jejího okolí, představují vážné riziko pro technologie. Různé typy vesmírného počasí mohou ovlivnit různé technologie na Zemi. Sluneční vzplanutí mohou produkovat silné rentgenové záření, které degraduje nebo blokuje vysokofrekvenční rádiové vlny používané pro rádiovou komunikaci. Solární energetické částice (energetické protony) mohou proniknout elektronikou satelitu a způsobit elektrické selhání. Tyto energetické částice také blokují rádiovou komunikaci ve vysokých zeměpisných šířkách během slunečních bouří. Aby bylo možné podat deterministické předpovědi vesmírného počasí, musíme ho nejprve pochopit. Výzkum fyziky slunečních erupcí představuje klíč k budoucím předpovědím vesmírného počasí. [32, 9]

Sluneční radiové emise (zejména vysokoenergetické solární rádiové záblesky) mají velký vliv na navigační, komunikační, mobilní a IoT (internet věcí) systémy. Unikátní rádiové vzplanutí z prosince 2006 s špičkovou úrovní jednoho milionu SFU (jednotka slunečního toku) způsobila významnou redukci v poměru signál/šum (SNR) většiny přijímačů GPS, některé přijímače úplně ztratily informace o satelitech. Nicméně degradace výkonu může nastat také u nižších úrovní rádiových vzplanutí, například solární rádiová vzplanutí o intenzitě 100 000 SFU způsobují v běžném GPS přijímači s typickou teplotou hluku 150 K a ziskem antény 3 dB, pokles poměru signál/šum o 15 dB, což může přijímači zabránit přijímat nové signály. [9]

### 8.4.2 Detekce rušení GNSS

V posledních letech zaznamenaly globální navigační satelitní systémy (GNSS) rychlý nárůst aplikací v různých odvětvích. Hlavní hrozba pro široké přijetí GNSS se týká zranitelnosti GNSS vůči rušení signálu. Nežádoucí signály v pásmech GNSS mohou vážně zhoršit službu a ovlivnit výkon. Účinky se pohybují od ztráty přesnosti až po úplné přerušení služby. To může vést ke katastrofálním důsledkům v operacích kritických z hlediska bezpečnosti. Je proto nanejvýš důležité, aby mohly být vyvinuty a rozmístěny spolehlivé a robustní techniky pro detekci rušení k ochraně infrastruktur a služeb GNSS před neúmyslným i záměrným rušením.

Signály GNSS jsou díky extrémně nízkému výkonu velmi náchylné na šum. Jakékoli zvýšení úrovně šumu na anténě přijímače nepříznivě ovlivní výkon přijímačů GNSS. Pokud je úroveň rušení tak vysoká, že elektronické součásti přijímače jsou saturované, signály mohou být neobnovitelné. Pokud je na přední straně přítomen dodatečný šum, přijímač se může setkat s následujícími situacemi:

- Nízký šum ovlivní přesnost měření.
- Střední šum způsobí problémy se sledováním a ztíží získávání satelitních signálů.
- Vysoký šum zcela zničí schopnost přijímače získat/sledovat požadované signály. [33]

# Kapitola 9
# Popis jednotlivých verzí programu

V této kapitole jsou vysvětleny všechny části programů testování a vyhodnocení. V kapitole nebudou popsány všechny funkce, jen kód pro příjem a zobrazení dat. Informace o elektronických přílohách, tedy programech přiložených k práci, budou vypsány v příloze B. V příloze jsou také instrukce ke spuštění programů. Vyhodnocení programu bude hlavně z ohledu výkonnosti programů, protože v tom se programy nejvíce lišily.

Programy byly testované na třech počítačích s různým výpočetním výkonem. První počítač měl tyto specifikace:

- Procesor - AMD Ryzen 5 1600 šesti-jádrový procesor s frekvencí 3.2 GHz
- Grafická karta - NVIDIA GeForce GTX 1650 SUPER
- RAM - 16 GB DDR4
- Disk - NVMe Samsung 980 SSD
- OS - Windows 10 Education

Druhý počítač měl tyto specifikace:

- Procesor - Intel Core i3 5005U dvou-jádrový procesor s frekvencí 2 GHz
- Grafická karta - Lenovo Intel(R) HD Graphics 5500
- RAM - 4 GB DDR3
- Disk - KINGSTON SA400S37480G SSD
- OS - Windows 10 Home

A třetí počítač měl tyto specifikace:

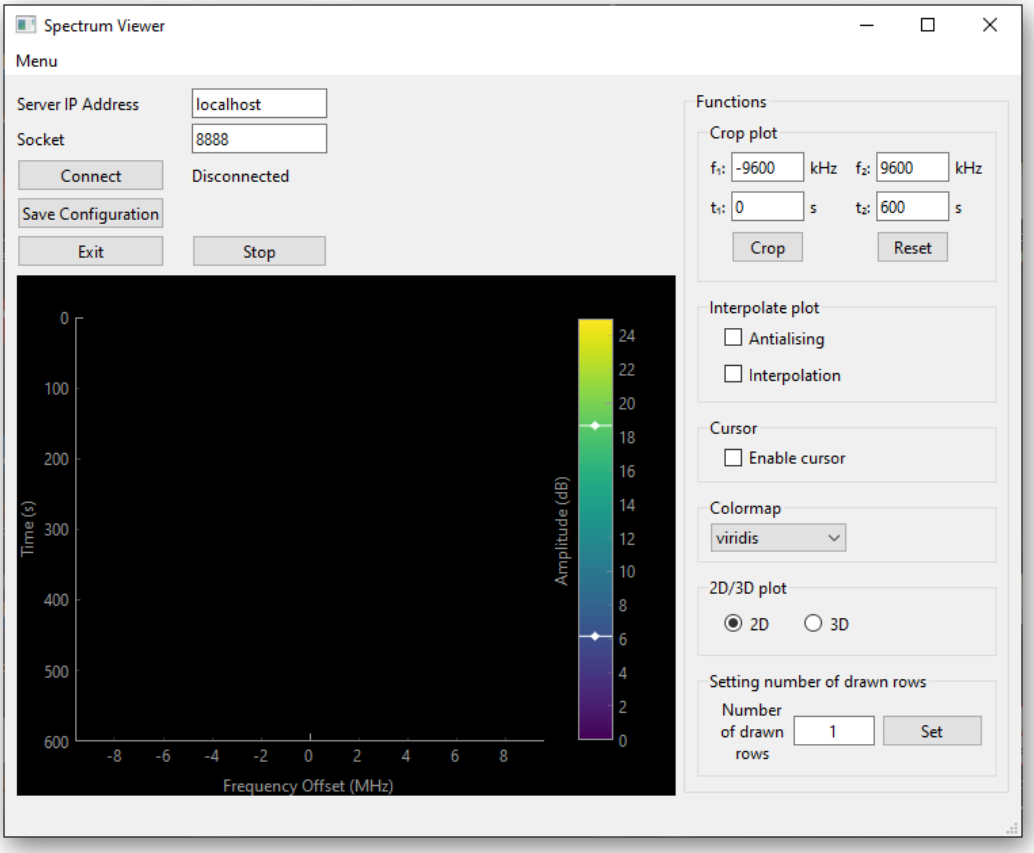
- Procesor - AMD FX-8320 osmi-jádrový procesor s frekvencí 3.5 GHz
- Grafická karta - NVIDIA GeForce GTX 1060 3 GB
- RAM - 8 GB DDR3
- Disk - Seagate ST1000DM003 HDD
- OS - Windows 10 Pro

## 9.1 Pythonový program s knihovnami PyQt a Pyqtgraph

Pro vývoj programu bylo využita verze 3.10 Pythonu. S knihovnami PyQt verze 6.2.3, NumPy verze 1.22.3, PyQtGraph verze 0.12.4, PyOpenGL verze 3.1.6 a PyInstaller verze 5.0.1.

Uživatelské prostředí programu bylo vytvořeno pomocí aplikace Qt Designer. S její pomocí se vytvoří soubor typu *User Interface*, který lze snadno převést na kód Pythonu. Tímto způsobem byl vytvořen základní kód grafického rozhraní programu. Vzhled programu lze vidět na obrázku číslo 9.1.

Poté bylo nutné vytvořit způsob, jak přijímat data přes TCP/IP. První byl vytvořen objekt soketu `self.spec_client` a poté propojen se serverem pomocí příkazů:




**Obrázek 9.1.** Ukázka programu v programovacím jazyce Python pomocí knihoven PyQt a PyQtGraph.

```
self.spec_client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
self.spec_client.connect((ROBOT_IP, ROBOT_PORT))
```

Kde `ROBOT_IP` je IP adresa serveru a `ROBOT_PORT` je soket serveru. Díky tomuto je navázáno spojení se serverem je možné přijímat data. Protože chceme přijímat data v určitém intervalu, musí existovat způsob, jak volat server v určitém čase. Toto je provedeno pomocí třídy *QTimer*, která je součástí PyQt. Poté, co se vytvoří spojení se serverem, spustí se object *QTimeru*, který bude opakovaně volat danou funkci v intervalu daném vzorkovací periodou. Přijatá data jsou zapisována do matice `self.waterfall`, řádek této matice se tedy rovná délce spektra. Zavolaná funkce, jako první se najde, kde řádek začíná a celý se načte pomocí kódu:

```
while True:
    msg = self.spec_client.recv(1)
    if msg[0] == 127:
        spec = self.spec_client.recv(self.waterfall_fbins)
        if len(spec) == self.waterfall_fbins:
            break
        spec = spec + \
        self.spec_client.recv(self.waterfall_fbins - len(spec))
        if len(spec) == self.waterfall_fbins:
            break
```

Data se přijímají příkazem `recv`. Data mají vždy délku délky spektra + 1. V tomto případě přijímám data po jednom paketu. Když má paket, v decimální soustavě, hodnotu 127 znamená to, že tam řádek začíná. Poté zkontroluji, že mi přišla délka řádku rovnající se délce spektra. Poté jsou data převedeny do logaritmické měřítka a proveden FFT shift, což převede nulové frekvenční složky dat do středu grafu.

```
e = s // 8
v = s % 8
ss = np.fft.fftshift(0.2 * np.log10(v * (2 ** e) + 0.1))
```

Funkce FFT shift je součástí knihovny NumPy. V Pythonu znamenají operátory // dělení se zbytkem, % zbytek dělení a ** umocňování.

Protože přijatá data se zapisují jako nový řádek matice, v případě programu je to řádek poslední, je třeba všechny elementy matice posunout o jeden řádek výš. V Pythonu je taková operace triviální a je popsána zde:

```
self.waterfall[1:self.waterfall_depth - 0, :] =
self.waterfall[0:self.waterfall_depth - 1, :]
self.waterfall[0, :] = ss[:]
```

Když jsou data přijatá a vložená do matice `self.waterfall` lze je vykreslit do grafu. Graf je vytvořený pomocí widgetu knihovny PyQtGraph, *PlotWidget*. Data jsou zobrazena ve formě obrázku s barevnou mapou. Objekt obrázku je definovaný jako `self.imv` a matice `self.waterfall` je do něj vložená pomocí příkazů:

```
self.imv = pg.ImageItem()
self.imv.setImage(self.waterfall)
```

Tímto se jednoduše vytvoří základní graf se dvěma dimenzemi. Ještě je třeba přesněji definovat prostředí grafu, upravit osy a přidat objekt barevné mapy `self.colorbar`.

```
self.PlotView.setLabel('left', "Time", units='s')
self.PlotView.setLabel('bottom', "Frequency Offset", units='Hz')
self.colorbar = pg.ColorBarItem(colorMap='viridis',
                                values=(self.amp_min, self.amp_max),
                                interactive=True, label='Amplitude (dB)')
self.colorbar.setImageItem(self.imv, insert_in=self.PlotView.plotItem)
```

Kde `self.PlotView` je objekt widgetu *PlotWidget* a, `self.amp_min` a `self.amp_max` jsou specifikovaný hodnoty maximální a minimální hodnoty amplitudy. Barevná mapa je v tomto případě „viridis“, což je jedna z vestavěných map z knihovny PyQtGraph. Protože má každý řádek počet prvků rovnající se délce spektra je třeba je převést tak, aby odpovídali danému frekvenčnímu rozsahu, tedy hodnotám od `-self.freq_offset` do `self.freq_offset`. Toho lze dosáhnout vynásobením osy frekvence hodnotou frekvenčního offsetu a posunutím dat o polovinu délky spektra, čehož lze dosáhnout třídou knihovny PyQt *QTransform*. Stejně tak je třeba vynásobit osu času vzorkovací periodou `self.sampling_period`, aby osa odpovídala reálnému času. Argument `interactive` dovoluje interaktivitu barevné mapy, lze tady nastavovat maximální a minimální amplitudu grafu, přímo při vykreslování.

```
self.tr = QtGui.QTransform()
self.tr.translate(-self.waterfall_fbins/2, 0)
self.imv.setTransform(self.tr)
self.PlotView.getAxis('bottom').setScale(self.freq_scale*1e3)
self.PlotView.getAxis('left').setScale(self.sampling_period/1000)
```

Hodnota `self.freq_offset` je v jednotkách kilohertzů a `self.sampling_period` v jednotkách milisekund, proto byly třeba ještě další přepočty do základních jednotek.

Program dokáže zobrazovat data i ve třech dimenzích. Bohužel PyQtGraph nemá vysokou podporu zpracování grafů ve třech dimenzích, proto je vytvoření grafu složitější. Protože je potřeba mít možnost přepínaní mezi těmito dvěma módy zobrazení, k tomu v programu slouží třída *QStackedWidget* knihovny Pyqt. Tato třída složí k tomu mít více widgetů, na sobě, na stejném místě a přepínat, který z nich je zrovna viditelný.

Widget pro graf se třemi dimenzemi má v knihovně PyQtGraph třídu *GLViewWidget*. Graf je vytvořená třídou *GLSurfacePlotItem* jako povrchový graf:

```
y = np.linspace(0, self.waterfall_fbins-1, self.waterfall_fbins)
x = np.linspace(0, self.waterfall_depth-1, self.waterfall_depth)
self.waterfall_threeD = gl.GLSurfacePlotItem(x=x, y=y, z=self.waterfall,
                        computeNormals=False, smooth=False)
```

Proměnné $x$ a $y$ je nutné definovat, protože povrchový graf musí předem znát hodnoty osy frekvence a času a až poté k nim přiřadit hodnoty amplitudy. Příkaz `linspace` je třída součástí knihovny NumPy, která dokáže vytvořit matici, s reálnými čísly, danou začátkem a koncem intervalu, a počet prvků. Argumenty `computeNormals` a `smooth` jsou nastaveny jako „nepravda“, protože to zvyšuje rychlost vykreslování. Argument `computeNormals` nastavuje, jestli se mají počítat normálové vektory a argument `smooth` nastavuje, jestli se mají normálové vektory vypočítávat pro každý vrchol a interpolovávat v rámci každé plochy. Data se do grafu vykreslují pomocí příkazu:

```
self.waterfall_threeD.setData(z=self.waterfall)
```

Třída *GLViewWidget* nedokáže sama od sebe vytvořit osy a všechny popisky, proto bylo třeba je vytvořit pomocí tříd *GLTextItem* a *GLGridItem*, protože je tento kód dlouhý, nebude v práci vypsaný.

### 9.1.1 Vyhodnocení

V této sekci je testována rychlost příjmu dat a jejich vykreslování. V zadání bylo specifikováno, že program musí zvládat přijímat data stokrát za sekundu. V prvním sloupci tabulky číslo 9.1 je vidět, jak dlouho průměrně trvá data přijmout, převést do logaritmického měřítka, provést FFT shift a vložit data do matice `self.waterfall`. Ve druhém sloupci je vidět, jak dlouho průměrně trvá data vykreslit. A ve třetím sloupci je doba jakou trvá vše dohromady. Časy byly měřeny pomocí modulu *time*, který je součástí standardní knihovny Pythonu. Bylo přijato deset řádků a časy byly zprůměrovány. Časy byly měřeny na dvou počítačích pro možnost porovnání.

Jak je z tabulek 9.1, 9.2, 9.3, 9.4 vidět většina běžných stolních počítačů by měla mít dostatečný výpočetní výkon pro zpracování a vykreslování dat zadanou rychlostí. Bohužel některé počítače, jako v tomto případě počítač číslo 3, nemusí mít dostatečný výpočetní výkon. Přesto je vykreslování dostatečně rychlé, aby šel program považovat za dostatečně výkonný pro splnění zadání. A díky tomu, že všechny ostatní funkce fungují jak mají, lze říci, že program splnil všechny požadavky, které byly dány zadáním.

Lze si také všimnout, že při vykreslování ve třech dimenzích, trvá vykreslování přibližně dvakrát déle než u vykreslování ve dvou dimenzích.

| | Doba příjmu dat [ms] | Doba vykreslování [ms] | Celkem [ms] |
|---|---|---|---|
| | 2.05 | 0.68 | 2.73 |
| | 3.22 | 1.32 | 4.54 |
| | 2.02 | 0.54 | 2.56 |
| | 2 | 0.55 | 2.55 |
| | 3.2 | 1.34 | 4.54 |
| | 3.57 | 1.2 | 4.77 |
| | 3.25 | 1.2 | 4.45 |
| | 3.17 | 1.34 | 4.51 |
| | 3.64 | 1.21 | 4.85 |
| | 1.86 | 0.56 | 2.42 |
| Průměr | 2.8 | 0.99 | 3.79 |

**Tabulka 9.1.** Rychlost zpracování dat v Pythonu pomocí knihoven PyQt a PyQtGraph na počítači číslo 1.

| | Doba příjmu dat [ms] | Doba vykreslování [ms] | Celkem [ms] |
|---|---|---|---|
| | 2.72 | 4.2 | 6.92 |
| | 2.76 | 4.16 | 6.92 |
| | 2.91 | 6.27 | 9.18 |
| | 2.76 | 6.23 | 8.99 |
| | 2.85 | 6.39 | 9.24 |
| | 2.89 | 4.84 | 7.73 |
| | 2.87 | 6.15 | 9.02 |
| | 2.81 | 6.08 | 8.89 |
| | 3.18 | 5.9 | 9.08 |
| | 2.76 | 6.51 | 9.27 |
| Průměr | 2.85 | 5.67 | 8.52 |

**Tabulka 9.2.** Rychlost zpracování dat v Pythonu pomocí knihoven PyQt a PyQtGraph na počítači číslo 1 při vykreslování ve třech dimenzích.

## 9.2 Pythonový program s knihovnami Tkinter a Matplotlib

Další verze programu byla vytvořena za pomocí Pythonu verze 3.10 a knihoven Matplotlib verze 3.5.2, NumPy verze 1.22.3 a PyInstaller verze 5.0.1. To jak program vypadá lze vidět na obrázku číslo 9.2.

Protože je příjem dat v této verzi programu, stejně jako ve verzi v sekci 9.1, vytvořený jen s pomocí modulu *Socket*, bez pomoci dalších knihoven, je jeho kód velmi obdobný a nebude zde znovu popisován.

Kód se začne lišit až vytváření samotného uživatelského prostředí. Zobrazení grafického prostředí Tkinter není statické ale cyklicky se obnovuje tak dlouho, dokud není dalším příkazem ukončeno. První se tedy grafické prostředí vytvoří jako objekt třídy *Tk* knihovny Tkinter a ukončí se příkazem `mainloop()`, který značí konec nekonečné smyčky událostí a tedy ukončuje celý blok kódu.

```
root = Tk()
```

| | Doba příjmu dat [ms] | Doba vykreslování [ms] | Celkem [ms] |
|---|---|---|---|
| | 6.35 | 1.94 | 8.29 |
| | 5.05 | 1.77 | 6.82 |
| | 4.59 | 3.05 | 7.64 |
| | 4.61 | 1.74 | 6.35 |
| | 5.57 | 2.95 | 8.52 |
| | 5.04 | 1.77 | 6.81 |
| | 4.71 | 1.49 | 6.2 |
| | 4.68 | 1.91 | 6.59 |
| | 5.67 | 1.92 | 7.59 |
| | 4.94 | 1.93 | 6.87 |
| Průměr | 5.12 | 2.05 | 7.17 |

**Tabulka 9.3.** Rychlost zpracování dat v Pythonu pomocí knihoven PyQt a PyQtGraph na počítači číslo 2.

| | Doba příjmu dat [ms] | Doba vykreslování [ms] | Celkem [ms] |
|---|---|---|---|
| | 10.79 | 3.65 | 14.44 |
| | 10.73 | 2.27 | 13 |
| | 11.72 | 2.41 | 14.13 |
| | 10.65 | 4.36 | 15.01 |
| | 10.87 | 2.89 | 13.76 |
| | 11.09 | 3.1 | 14.19 |
| | 11.45 | 2.71 | 14.16 |
| | 11.06 | 3.73 | 14.79 |
| | 11.06 | 3.84 | 14.9 |
| | 11 | 3.04 | 14.04 |
| Průměr | 11.04 | 3.2 | 14.24 |

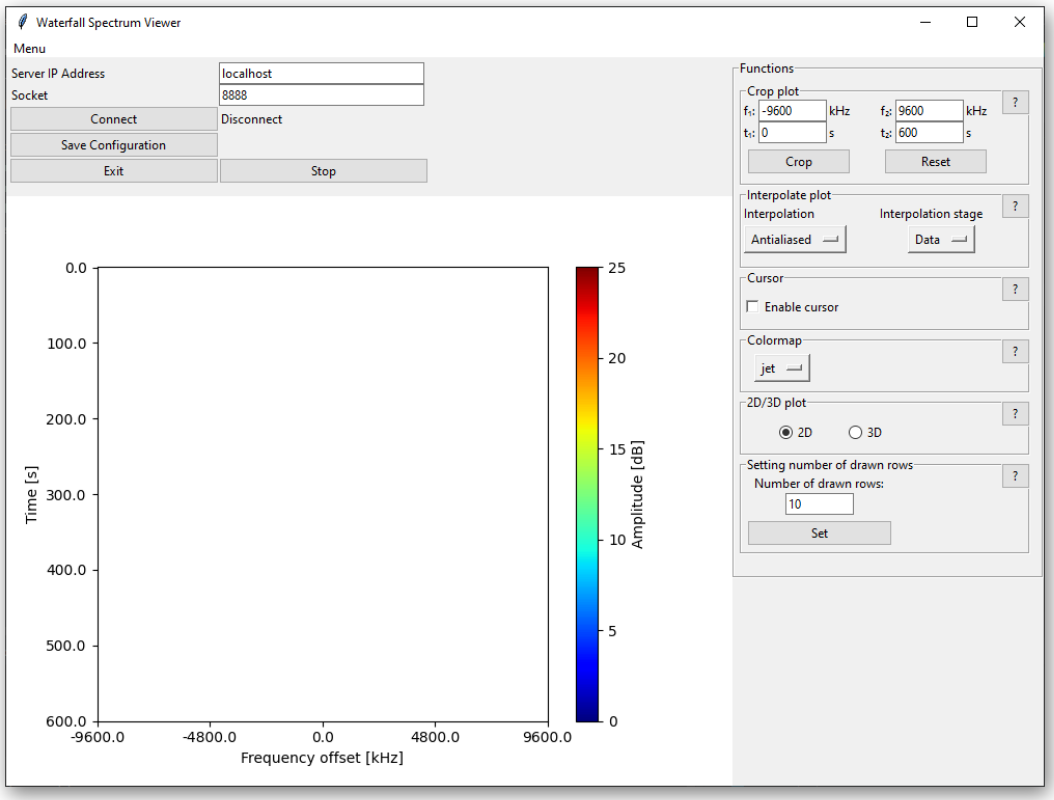
**Tabulka 9.4.** Rychlost zpracování dat v Pythonu pomocí knihoven PyQt a PyQtGraph na počítači číslo 3.

```
root.mainloop()
```

Celý zbytek kódu je vložený mezi tyto dva příkazy. Stojí za zmínku, jak Tkinter umísťuje své objekty a widgety. Pozice jednotlivých objektů uvnitř nadřízených kontejnerů se určují pomocí správců geometrie, jež jsou tři:

- Pack – používá metodu `pack()`. Systém Pack organizuje objekty do bloků před jejich umístěním do nadřazeného okna. Místo přesného absolutního určení pozice objektu lze pozici určit relativně vzhledem k jinému objektu.
- Place – používá metodu `place()`. Systém Place umožňuje explicitně zadat pozici a velikost objektu a to buď absolutně nebo relativně k jinému objektu.
- Grid – používá metodu `grid()`. Systém Grid je flexibilnější než Pack a méně pracný než Place. Systém umisťuje objekty do buněk dvojrozměrného rastru, který se skládá z řádků a sloupců. Pozice objektů je daná číslem řádku a sloupce. Sousední buňky lze vhodně spojovat.

Systémy Pack a Grid by se nikdy neměli kombinovat v jednom okně.




**Obrázek 9.2.** Ukázka programu v programovacím jazyce Python pomocí knihoven Tkinter a Matplotlib.

Dále je třeba vytvořit grafické prostředí, do kterého se bude vkládat graf, to lze vytvořit pomocí třídy *Figure* knihovny Matplotlib. Graf je poté vložen příkazem `add_subplot`:

```
fspec = Figure()
sbandplt = fspec.add_subplot(111)
specplot = sbandplt.imshow(waterfall, cmap='jet')
```

Kde `waterfall` je matice s přijatými daty, obdobně jako matice `self.waterfall` v sekci 9.1. Data budou zobrazená jako obrázek, proto příkaz `imshow`, který nastaví zobrazení grafu jako obrázek s barevnou mapou. Barevná mapa je v tomto případě nastavená jako „jet“, což je jedna z vestavěných barevných map knihovny Maplotlib.

Data lze vykreslovat i ve třech dimenzích. K tomu se musí grafické prostředí vytvořit o trochu jiným způsobem:

```
sbandplt_3D = fspec.add_subplot(111, projection='3d')
x1 = 0
x2 = waterfall_fbins-1
t1 = 0
t2 = waterfall_depth-1
x = np.linspace(0, x2, num=waterfall_fbins)
y = np.linspace(0, t2, num=waterfall_depth)
X, Y = np.meshgrid(x, y)
sbandplt_3D.plot_surface(X, Y, waterfall[t1:t2 + 1, x1:x2 + 1])
```

Argument `projection='3d'` dává Matplotlibu zprávu, že graf bude mít tři dimenze. Graf je znovu vykreslený pomocí povrchového grafu, takže je třeba definovat proměnné `X` a `Y`, které udávají hodnoty frekvence a času.

Nakonec je třeba nastavit backend a data do připraveného grafu vložit. Toho lze dosáhnout následujícím kódem:

```
matplotlib.use("TkAgg")
canvas = FigureCanvasTkAgg(fspec, master=root)
canvas.draw()
canvas.get_tk_widget().grid(column=0, row=1, sticky='nsew')
```

Příkaz `matplotlib.use('TkAgg')` nastaví backend Matplotlibu, jako „TkAgg“, což je standardní backend pro Tkinter. Objekt `canvas` s tímto stává plochou, do které se bude vykreslovat graf a příkazem `grid()` se umístí na specifikovanou pozici. Příkaz `draw()` znamená, že se má graf aktualizovat a znovu vykreslit. Vykreslují se nejen data v grafu, ale i celé prostředí grafu, tedy i osy, popisky a všechny objekty uvnitř objektu `fspec`.

Dále se musí nastavit prostředí grafu, jako osy a popisky, a specifikovat barevnou mapu. Barevná mapa lze vytvořit jednoduchým kódem:

```
cbar = fspec.colorbar(specplot, orientation='vertical', ax=sbandplt)
z_ticks = np.linspace(z_min, z_max, num=6)
cbar.ax.set_yticks(z_ticks)
cbar.ax.set_ylabel('Amplitude [dB]')
```

Data se tedy dají vykreslovat pomocí příkazů:

```
specplot.set_data(waterfall[0:waterfall_depth-1, 0:waterfall_fbins-1])
fspec.canvas.draw()
```

Čímž se do grafu vloží nová data a následně se vykreslí. Dále je třeba nová data přijímat v daném časovém intervalu daném vzorkovací periodou. Knihovna Tkinter nemá spolehlivou a přesnou metodu, jak periodicky volat funkci, proto je tato metoda vytvořená ručně následující funkcí:

```
def every(event, delay, task):
    next_time = time.time() + delay
    while not event.is_set():
        time.sleep(max(0, next_time - time.time()))
        try:
            task()
        except Exception:
            traceback.print_exc()
        next_time += delay
```

Tato funkce je schopná volat funkci danou argumentem `task` v intervalu `delay`. Funkce lze zavolat příkazem:

```
refresh_stop_event = Event()
refresh_thread = threading.Thread(target=lambda: every(refresh_stop_event,
sampling_period, refresh))
refresh_thread.daemon = True
refresh_thread.start()
threading.Thread(target=lambda: every(sampling_period, test)).start()
```

V tomto případě se funkce, která periodicky vykresluje data, jmenuje `refresh()` a volá se v intervalu `sampling_period`, což je hodnota vzorkovací frekvence. Objekt `threading` je objekt knihovny Threading, která je součástí standardní knihovny Python. Funkce tedy běží na novém vlákně a je tedy nezávislá na hlavním vlákně, na kterém běží uživatelské rozhraní. Vlákno je nastavené jako `Daemon`, což znamená že pracuje v pozadí a ukončí se, když se ukončí hlavní vlákno, na kterém pracuje zbytek programu. Pomocí objektu `refresh_stop_event`, což je objekt události vlákna, lze vlákno kdykoliv ukončit.

### 9.2.1 Vyhodnocení

Stejně jako v sekci 9.1.1 bylo měřeno čas za jaký se data zpracují a vykreslí. Výsledky tohoto měření jsou vidět v tabulce 9.5.

| | Doba příjmu dat [ms] | Doba vykreslování [ms] | Celkem [ms] |
|---|---|---|---|
| | 3.45 | 65.76 | 69.21 |
| | 3.65 | 66.88 | 70.53 |
| | 2.2 | 67.28 | 69.48 |
| | 3.49 | 66.42 | 69.91 |
| | 3.55 | 66.7 | 70.25 |
| | 3.64 | 67.14 | 70.78 |
| | 4.39 | 67.78 | 72.17 |
| | 3.64 | 68.59 | 72.23 |
| | 3.48 | 67.52 | 71 |
| | 3.76 | 65.98 | 69.74 |
| Průměr | 3.52 | 67 | 70.5 |

**Tabulka 9.5.** Rychlost zpracování dat v Pythonu pomocí knihoven Tkinter a Matplotlib na počítači číslo 1.

| | Doba příjmu dat [ms] | Doba vykreslování [ms] | Celkem [ms] |
|---|---|---|---|
| | 2.85 | 108.9 | 111.75 |
| | 2.88 | 115.38 | 118.26 |
| | 2.89 | 124.25 | 127.14 |
| | 2.8 | 117.5 | 120.3 |
| | 2.84 | 123.17 | 126.01 |
| | 2.93 | 108.93 | 111.86 |
| | 2.88 | 108 | 110.88 |
| | 2.86 | 108.32 | 111.18 |
| | 2.87 | 102 | 104.87 |
| | 2.94 | 120.15 | 123.09 |
| Průměr | 2.87 | 113.66 | 116.53 |

**Tabulka 9.6.** Rychlost zpracování dat v Pythonu pomocí knihoven Tkinter a Matplotlib na počítači číslo 1 při vykreslování ve třech dimenzích.

V tabulkách 9.5, 9.6, 9.7 a 9.8 je vidět, že žádný z počítačů nedokáže spektrum vykreslovat se zadanou rychlostí. Z toho je zřejmé, že i přestože všechny zadané funkce

| | Doba příjmu dat [ms] | Doba vykreslování [ms] | Celkem [ms] |
|---|---|---|---|
| | 6.45 | 198.73 | 205.18 |
| | 5.46 | 190.23 | 195.69 |
| | 6.13 | 184.36 | 190.49 |
| | 8.05 | 189.71 | 197.76 |
| | 6.5 | 212.9 | 219.4 |
| | 6.55 | 197.5 | 204.05 |
| | 8.11 | 211.77 | 219.88 |
| | 6.75 | 176.66 | 183.41 |
| | 6 | 198.6 | 204.6 |
| | 7.78 | 203.46 | 211.24 |
| Průměr | 6.78 | 296.39 | 303.17 |

**Tabulka 9.7.** Rychlost zpracování dat v Pythonu pomocí knihoven Tkinter a Matplotlib na počítači číslo 2.

| | Doba příjmu dat [ms] | Doba vykreslování [ms] | Celkem [ms] |
|---|---|---|---|
| | 10.23 | 132.01 | 142.24 |
| | 10.44 | 130.96 | 141.4 |
| | 12.5 | 128.73 | 141.23 |
| | 13.97 | 128.16 | 142.13 |
| | 11.2 | 136.76 | 147.96 |
| | 11.14 | 147.26 | 158.4 |
| | 10.24 | 153.6 | 163.84 |
| | 10.46 | 134.07 | 144.53 |
| | 11 | 146.33 | 157.33 |
| | 10.08 | 141.04 | 151.12 |
| Průměr | 11.13 | 137.9 | 149.03 |

**Tabulka 9.8.** Rychlost zpracování dat v Pythonu pomocí knihoven Tkinter a Matplotlib na počítači číslo 3.

v programu pracují správně, kvůli nízké rychlosti vykreslování, tato verze programu nesplňuje zadání. Vykreslování ve třech dimenzích je opět přibližně dvakrát pomalejší než při vykreslování grafu se dvěma dimenzemi.

## 9.3 Program v C++

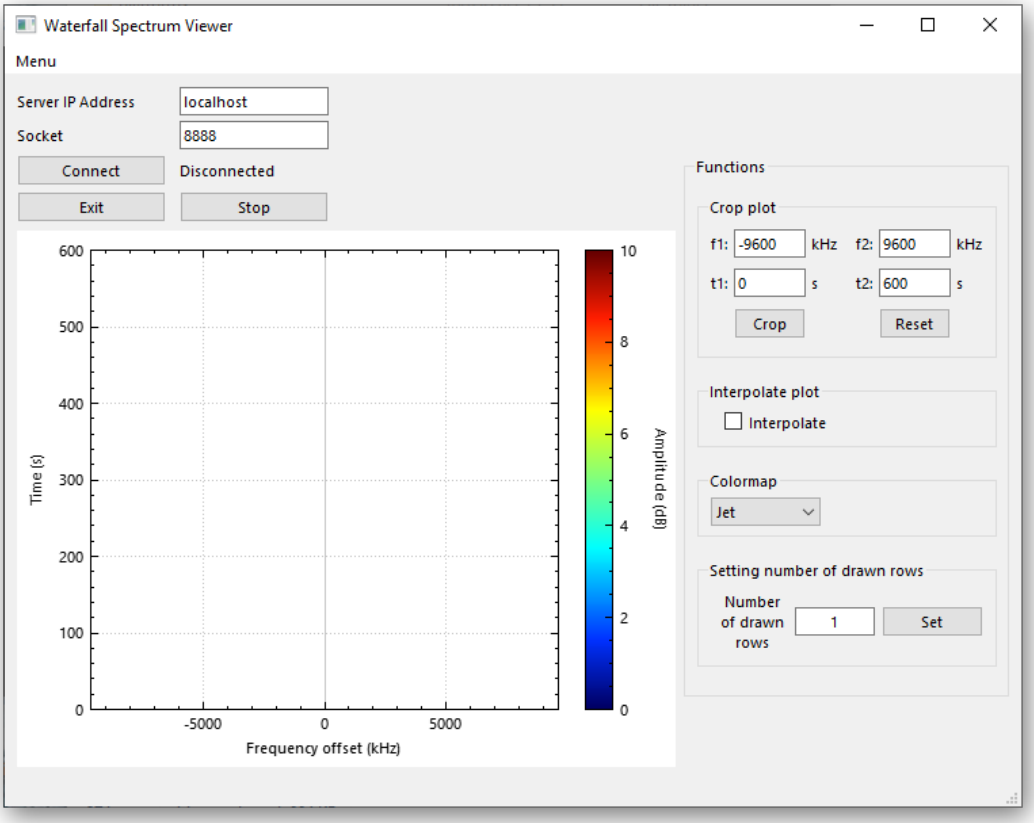
Pro vývoj aplikace bylo zvoleno prostředí Qt Creator 7.0.0, integrální součást Qt frameworku, v kombinaci s programovacím jazykem C++. Jako kompilátor byl použit MinGW 8.1.0, jako debugger GDB 8.1.

Mimo vlastních knihoven Qt frameworku byla použitá jen jediná knihovna, a to knihovna QCustomPlot verze 2.1.0. To, jak program vypadá lze vidět na obrázku číslo 9.3

Program v C++ byl vytvořen spíše jako zkušební, pro vyzkoušení rychlosti vykreslování a zpracování dat, oproti Pythonu.

Kód programu je rozdělený do několika souborů, z toho jsou tři hlavičkové, čtyři implementační a jeden projektový. Přesněji jsou to soubory:

- Waterfall_QCustomPlot.pro
- mainwindow.h
- options_window.h
- qcustomplot.h
- mainwindow.cpp
- options_window.cpp
- qcustomplot.cpp
- main.cpp
- mainwindow.ui
- options_window.ui




**Obrázek 9.3.** Ukázka programu napsaného v programovacím jazykun C++ s použitím knihoven Qt a QCustomPlot.

Soubory *qcustomplot.cpp* a *qcustomplot.h* jsou soubory knihovny QCustomPlot. Soubor *main.cpp* je soubor, který se volá jako první při spuštění programu a otevře grafické rozhraní. V souborech *options_window.cpp*, *options_window.cpp* a *options_window.ui* jsou kódy pro grafické rozhraní okna `Options`, ve kterém se nastavují parametry přijímaných dat. Konečně v souborech *mainwindow.cpp*, *mainwindow.cpp* a *mainwindow.ui* je vypsaný celý zbytek kódu. V popisu tedy budou vypsány části kódu ze souboru *mainwindow.cpp*.

Grafické rozhraní je vytvořeno pomocí aplikace Qt Designer. Ta vytvoří soubor *User Interface*, který lze snadno integrovat s kódem programu. Všechny informace o grafickém prostředí jsou tedy uloženy v těchto souborech.

Příjem dat pracuje na stejném principu jako u Pythonovských verzí programu, proto, i přesto že je kód napsaný v jiném programovacím jazyce, je kód velmi podobný. První byl vytvořen objekt soketu `spec_client`, který byl použit k tomu připojit se k serveru, následujícím kódem:

```
struct sockaddr_in server;
spec_client = socket(AF_INET, SOCK_STREAM, 0);
server.sin_addr.s_addr = inet_addr(IP_adr);
server.sin_family = AF_INET;
server.sin_port = htons(socket_adr);
::connect(spec_client, (struct sockaddr*)&server, sizeof(server))
```

Kde `sockaddr_in` je struktura pro práci s internetovými adresami. Struktura je datový typ, která je tvořena výčtem proměnných, které jsou seskupeny pod jedním jménem. Do této struktury jsou následně uložené hodnoty IP adresy, adresy soketu a domény soketu, které jsou poté použity ve funkci `::connect()` k propojení se serverem. Kvalifikátor `::` umožňuje přístup ke globálnímu jmennému prostoru, kde je funkce `::connect()` definována.

Grafické prostředí je vytvořené pomocí třídy *PlotWidget* knihovny QCustomPlot. Po nastavení os a popisů se graf se vytvoří pomocí následujícího kódu:

```
colorMap->data()->setSize(waterfall_fbins, waterfall_depth);
colorMap->data()->setRange(QCPRange(-freq_offset, freq_offset),
        QCPRange(0, (waterfall_depth*sampling_period/1000)));
```

Kde `colorMap` je objekt stejnojmenné třídy *QCPColorMap* knihovny QCustomPlot, který je součástí třídy *PlotWidget*. Proměnná `waterfall_fbins` má hodnotu délky spektra, `waterfall_depth` má hodnotu počtu řádků grafu, `freq_offset` se rovná hodnotě frekvenčního rozsahu a `sampling_period` je vzrokovací perioda. Příkaz `setSize` nastaví počet bodů grafu v obou osách a příkaz `setRange` nastaví jakým hodnotám tyto body odpovídají. To znamená, že se v grafu data vykreslí rovnou se správnými hodnotami, i přestože má graf ve frekvenční ose jen počet bodů rovnající se délce spektra, ale má zobrazovat data v celém frekvenčním rozlišením.

Další část kódu přiřadí grafu barevnou mapu:

```
PlotWidget->plotLayout()->addElement(0, 1, colorScale);
colorMap->setColorScale(colorScale);
```

Barevná mapa je vytvořena třídou *QCPColorScale* knihovny QCustomPlot. Jejím objektem je v tomto případě objekt `colorScale`. A příkazem `setColorScale` je objekt `colorScale` přiřazen objektu `colorMap` jako barevná mapa.

Po propojení se serverem se začne volat funkce, která vykresluje data. Funkce je volaná pomocí třídy *QTimer*, která je součástí knihovny Qt. Funkce se volá periodicky v čase daném vzorkovací periodou. První se posunou všechny řádky matice kódem:

```
for (int x=0; x<waterfall_fbins; ++x)
{   for (int y=waterfall_depth-2; y>=0; --y)
       waterfall[y+1][x] = waterfall[y][x];}
```

Data se přijímají pomocí následujícího kódu:

```
while(true)
    {recv(spec_client, received_data, 1, 0);
     memcpy(&msg, received_data, sizeof(int));
```

```
    if(msg == 127)
        {for (int i=0; i<waterfall_fbins; ++i)
            {recv(spec_client, received_data, 1, 0);
             memcpy(&msg, received_data, sizeof(int));
             double e = msg / 8;
             double v = msg % 8;
             waterfall[0][i] = 0.2 * log10(v * pow(2,e) + 0.1);}
             break;}
```

Kód je obdobný kódu v Pythonovských verzí programu, jen nemá kontrolu přijetí celého řádku. Začnou se přijímat data po paketech, až přijde ten, který má hodnotu 127. Data se převádějí z binárního systému na systém decimální pomocí příkazu `memcpy`, což je funkce, která kopíruje bajty mezi vyrovnávacími paměťmi. Poté se data převedou do logaritmického měřítka a vloží do matice `waterfall`.

Poté se provede FFT shift a data se vloží do grafu:

```
for (int x=0; x<waterfall_fbins-1; ++x)
    std::swap(waterfall[0][x], waterfall[0][x+waterfall_fbins/2]);
for (int x=0; x<waterfall_fbins; ++x)
  for (int y=0; y<waterfall_depth; ++y)
    colorMap->data()->setCell(x, y, waterfall[y][x]);
PlotWidget->replot();
```

Data se vkládají do grafu pomocí příkazu `setCell`. Nakonec se data vykreslí pomocí příkazu `replot()`

### 9.3.1 Vyhodnocení

Znovu byla změřená doba, za kterou se data přijmou a zpracují, a poté vykreslí. Naměřené hodnoty jsou viditelné v tabulce 9.9. Časy byly naměřené pomocí třídy *QElapsedTimer* knihovny Qt, která dokáže změřit uplynulý čas mezi dvěma událostmi. Doba příjmu dat je doba, za kterou se posunou řádky matice `waterfall`, pomocí TCP/IP se přijmou data a provede se operace FFT shift. Doba vkládání dat je čas, za který se data vloží do grafu a doba vykreslovaní je doba, za kterou se graf znovu překreslí.

| | Doba příjmu dat [ms] | Doba vkládání dat [ms] | Doba vykreslování [ms] | Celkem [ms] |
|---|---|---|---|---|
| | 2.83 | 2.46 | 2.44 | 7.73 |
| | 1.82 | 2.44 | 2.46 | 6.72 |
| | 2.72 | 3.7 | 3.95 | 10.37 |
| | 2.83 | 2.32 | 3.29 | 8.44 |
| | 2.91 | 4.74 | 3.07 | 10.72 |
| | 3.02 | 2.29 | 2.44 | 7.75 |
| | 2.96 | 5.15 | 3 | 11.11 |
| | 2.92 | 3.22 | 2.46 | 11.07 |
| | 2.53 | 2.29 | 2.44 | 7.26 |
| | 2.96 | 5.37 | 2.35 | 10.68 |
| Průměr | 2.75 | 3.4 | 2.79 | 8.94 |

**Tabulka 9.9.** Rychlost zpracování dat v C++ pomocí knihoven Qt a QCustomPlot na počítači číslo 1.

| | Doba příjmu dat [ms] | Doba vkládání dat [ms] | Doba vykreslování [ms] | Celkem [ms] |
|---|---|---|---|---|
| | 5.82 | 21.46 | 9.54 | 36.82 |
| | 5.85 | 22.14 | 7.91 | 35.9 |
| | 6.08 | 21.66 | 6.67 | 34.41 |
| | 5.23 | 21.6 | 6.69 | 33.52 |
| | 5.42 | 20.39 | 7.31 | 33.12 |
| | 5.71 | 22.27 | 4.2 | 32.18 |
| | 6.06 | 21.72 | 6.6 | 34.38 |
| | 5.85 | 21.7 | 6.95 | 34.5 |
| | 6.39 | 21.89 | 3.43 | 31.71 |
| | 5.53 | 20.89 | 6.76 | 33.18 |
| Průměr | 5.8 | 21.57 | 6.6 | 33.97 |

**Tabulka 9.10.** Rychlost zpracování dat v C++ pomocí knihoven Qt a QCustomPlot na počítači číslo 2.

| | Doba příjmu dat [ms] | Doba vkládání dat [ms] | Doba vykreslování [ms] | Celkem [ms] |
|---|---|---|---|---|
| | 7.8 | 15.97 | 7.64 | 23.77 |
| | 8.43 | 16.21 | 6.48 | 31.12 |
| | 16.9 | 17.31 | 4.25 | 38.46 |
| | 17.39 | 16.74 | 5.91 | 40.04 |
| | 10.64 | 16.4 | 5.25 | 32.29 |
| | 18.09 | 13.89 | 4.58 | 36.56 |
| | 10.63 | 17.34 | 5.48 | 33.45 |
| | 17.86 | 22.58 | 5.87 | 46.31 |
| | 17.6 | 14.49 | 4.52 | 36.61 |
| | 6.78 | 11.35 | 5.29 | 23.42 |
| Průměr | 13.21 | 16.23 | 5.53 | 34.97 |

**Tabulka 9.11.** Rychlost zpracování dat v C++ pomocí knihoven Qt a QCustomPlot na počítači číslo 3.

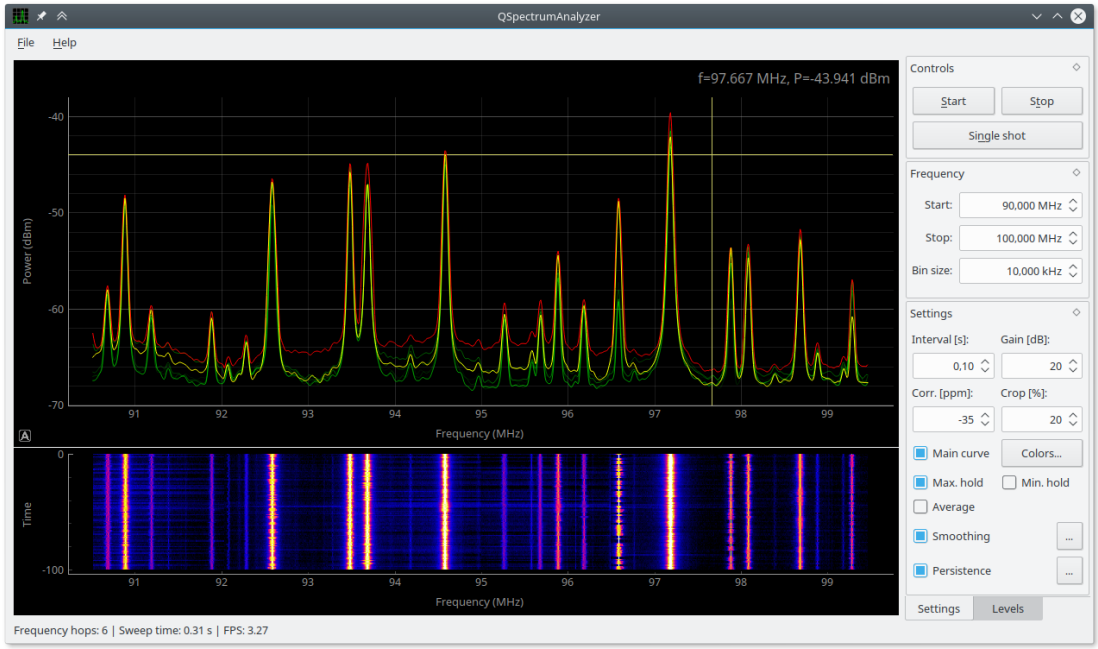
Z tabulek 9.9, 9.10 a 9.11 je jasné, že program nemůže spolehlivě vykreslovat 100 řádků za sekundu. Je také vidět, že doba příjmu dat je vyšší než u Pythonových verzí programu, to je způsobené hlavně posuvem řádků matice, což je v programu pomalejší než u Pythonových verzí. Znamená to, že pokud se bude vykreslovat větší počet dat, tedy bude větší délka spektra nebo bude zobrazen větší počet řádků, bude příjem dat a vykreslování výrazně pomalejší. Při délce spektra 16384 a 6000 vyobrazenými řádky byl naměřený čas, na počítači číslo 1, za který se tabulka posunula, průměrně 20 ms. Protože v programu chybí některé zadané funkce a program nedokáže vždy zpracovat data zadanou rychlostí, tato verze programu nesplňuje zadání.

# Kapitola 10
# Dostupné softwarové nástroje

Pro sledování spektra je k dispozici spousta programů, avšak drtivá většina z nich jsou programy pro sledování zvukových signálů. Pokud chceme najít program pro sledování spektra elektromagnetických signálů, je nabídka o mnoho skromnější. V kapitole je popsán jeden program, který slouží k příjmu dat z SDR a jejich zobrazování a několik dalších programovacích prostředí, které by bylo možné k podobnému účelu využít.

## 10.1 QSpectrumAnalyzer

QSpectrumAnalyzer je open-source grafické rozhraní založené na Linuxu pro spektrální analýzu. QSpectrumAnalyzer zobrazuje data z různých platforem SDR jako rtl_power nebo rtl_power_fftw a lze jej použít s RTL-SDR ke sledování aktivity signálu v širokém pásmu frekvenčního spektra. Vzhled programu lze vidět na obrázku číslo 10.1. [34]




**Obrázek 10.1.** Ukázka programu QSpectrumAnalyzer. [34]

Program QSpectrumAnalyzer je sepsaný v programovacím jazyce Python a stojí na knihovnách PyQt a PyQtgraph. QSpectrumanalyzer funguje jako klient pro různé backendy SDR a dokáže pracovat s těmito backendy:

- Soapy_power[1] je výchozí a doporučený univerzální backend SDR v QSpectrumAnalyzer. Je založen na SoapySDR a podporuje téměř všechny platformy SDR (RTL-SDR, HackRF, Airspy, SDRplay, LimeSDR, bladeRF, USRP a některá další zařízení

[1] `https://github.com/xmikos/soapy_power`

SDR). Je vysoce konfigurovatelný a podporuje krátkou dobu akvizice dat pro kontinuální měření téměř v reálném čase.

- Hackrf_sweep[2] backend umožňuje monitorování širokopásmového spektra rychlým přeladěním rádia bez nutnosti individuálních požadavků na ladění od hostitelského počítače. Tento backend podporuje pouze platformu HackRF.
- Rtl_power[3] je originální backend pro zařízení RTL-SDR.
- Rtl_power_fftw[4] je alternativní backend pro zařízení RTL-SDR a má různé výhody oproti rtl_power. Například lepší výkon FFT (díky použití knihovny fftw) a možnost využít krátkou dobu akvizice pro kontinuální měření v téměř reálném čase (minimální interval v původním rtl_power je 1 sekunda).
- Rx_power[5] je také založen na SoapySDR, a proto podporuje téměř všechny platformy SDR. Ale je mnohem pomalejší než soapy_power a nepodporuje kontinuální měření téměř v reálném čase (minimální interval je 1 sekunda, stejně jako rtl_power). Backend momentálně v QSpectrumAnalyzer není podporován. [34]

## 10.2 MATLAB

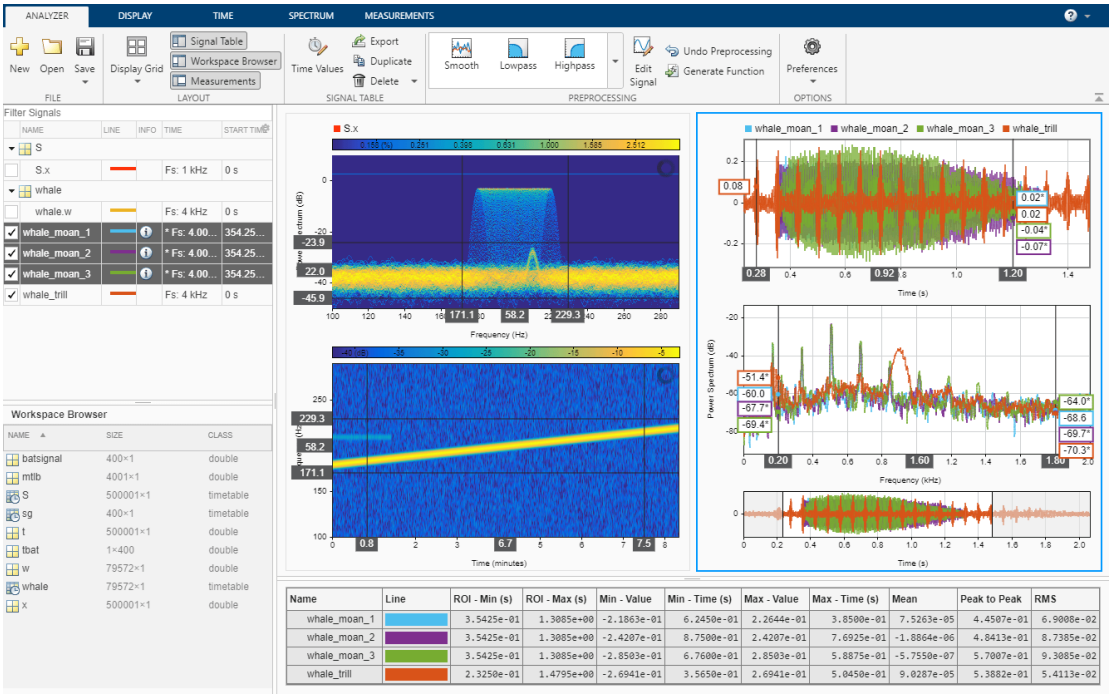
MATLAB je programovací prostředí pro vysoce výkonné numerické výpočty a vizualizace. Poskytuje interaktivní prostředí se stovkami vestavěných funkcí pro technické výpočty, grafiku a animace. Název MATLAB je zkratka pro MATrix LABORatory. Vestavěné funkce MATLABu poskytují vynikající nástroje pro výpočty lineární algebry, analýzu dat, zpracování signálů, optimalizaci, numerické řešení běžných diferenciální rovnicí a mnoho dalších typů vědeckých výpočtů. Obsahuje také funkce pro grafiku a animaci.

Jazyk MATLABu se velmi snadno učí a používá. Existuje také mnoho volitelných „toolboxů“, nebo-li aplikačních knihoven, dostupných od vývojářů MATLABu. Tyto aplikační knihovny jsou kolekce funkcí napsaných pro speciální aplikace, jako jsou symbolické výpočty, zpracování obrazu nebo statistika. Počet těchto aplikačních knihoven se postupem času rozrůstá. Součástí MATLABu je také Simulink, což je prostředí pro modelování, simulaci a analýzu dynamických systémů pomocí blokových diagramů. [35]

MATLAB obsahuje několik aplikačních knihoven, které jsou užitečné k zobrazování spektra. První aplikační knihovnou by byla určitě knihovna „Signal Processing Toolbox“. Tato knihovna poskytuje funkce a aplikace pro správu, analýzu, předzpracování a extrahování vlastností ze vzorkovaných signálů. Knihovna obsahuje nástroje pro návrh a analýzu filtrů, převzorkování, vyhlazování, odstraňování trendu a odhad výkonového spektra. Knihovna obsahuje aplikaci Signal Analyzer, která lze použít pro vizualizaci a zpracování více signálů současně v časové, frekvenční a časově frekvenční doméně. Zajímavá je například funkce spectrogram, která dokáže vytvářet spektrogram pomocí krátkodobé Fourierovy transformace. [36]

Další aplikační knihovnou by mohla být knihovna „Parallel Computing Toolbox“. Knihovna umožňuje řešit výpočetně a datově náročné problémy pomocí vícejádrových procesorů, GPU a počítačových clusterů. Funkce MATLAB Parallel Server umožňuje spouštět aplikace na clusterech nebo cloudech beze změny kódu aplikace. Knihovna lze tedy použít s funkcí MATLAB Parallel Server k provádění maticových výpočtů, které jsou příliš velké na to, aby se vešly do paměti jednoho přístroje. [38]

[2] https://github.com/greatscottgadgets/hackrf
[3] https://github.com/keenerd/rtl-sdr
[4] https://github.com/AD-Vega/rtl-power-fftw
[5] https://github.com/rxseger/rx_tools




| Name | Line | ROI - Min (s) | ROI - Max (s) | Min - Value | Min - Time (s) | Max - Value | Max - Time (s) | Mean | Peak to Peak | RMS |
|---|---|---|---|---|---|---|---|---|---|---|
| whale_moan_1 | | 3.5425e-01 | 1.3085e+00 | -2.1863e-01 | 6.2450e-01 | 2.2644e-01 | 3.8500e-01 | 7.5263e-05 | 4.4507e-01 | 6.9008e-02 |
| whale_moan_2 | | 3.5425e-01 | 1.3085e+00 | -2.4207e-01 | 8.7500e-01 | 2.4207e-01 | 7.6925e-01 | -1.8864e-06 | 4.8413e-01 | 8.7385e-02 |
| whale_moan_3 | | 3.5425e-01 | 1.3085e+00 | -2.8503e-01 | 6.7600e-01 | 2.8503e-01 | 5.8875e-01 | -5.7550e-07 | 5.7007e-01 | 9.3085e-02 |
| whale_trill | | 2.3250e-01 | 1.4795e+00 | -2.6941e-01 | 3.5650e-01 | 2.6941e-01 | 5.0450e-01 | 9.0287e-05 | 5.3882e-01 | 5.4113e-02 |

**Obrázek 10.2.** Ukázka aplikace Signal Analyzer. [37]

MATLAB také zvládá příjímat data z TCP/IP v reálném čase a vytvářet grafické rozhraní. Grafické rozhraní lze vytvořit buď pomocí kódu nebo pomocí aplikace App Designer, což je interaktivní prostředí, ve kterém lze rozvrhnout vizuálních komponenty a naprogramovat chování aplikace. Ukázka aplikace App Designer lze vidět na obrázku číslo 10.3.

Další zajímavou součástí MATLABu je portál File Exchange. File Exchange umožňuje vyhledávat a sdílet aplikace, třídy, funkce, modely v Simulinku, skripty a videa. To znamená, že je možné si stáhnout již vytvořený a funkční kód aplikace.

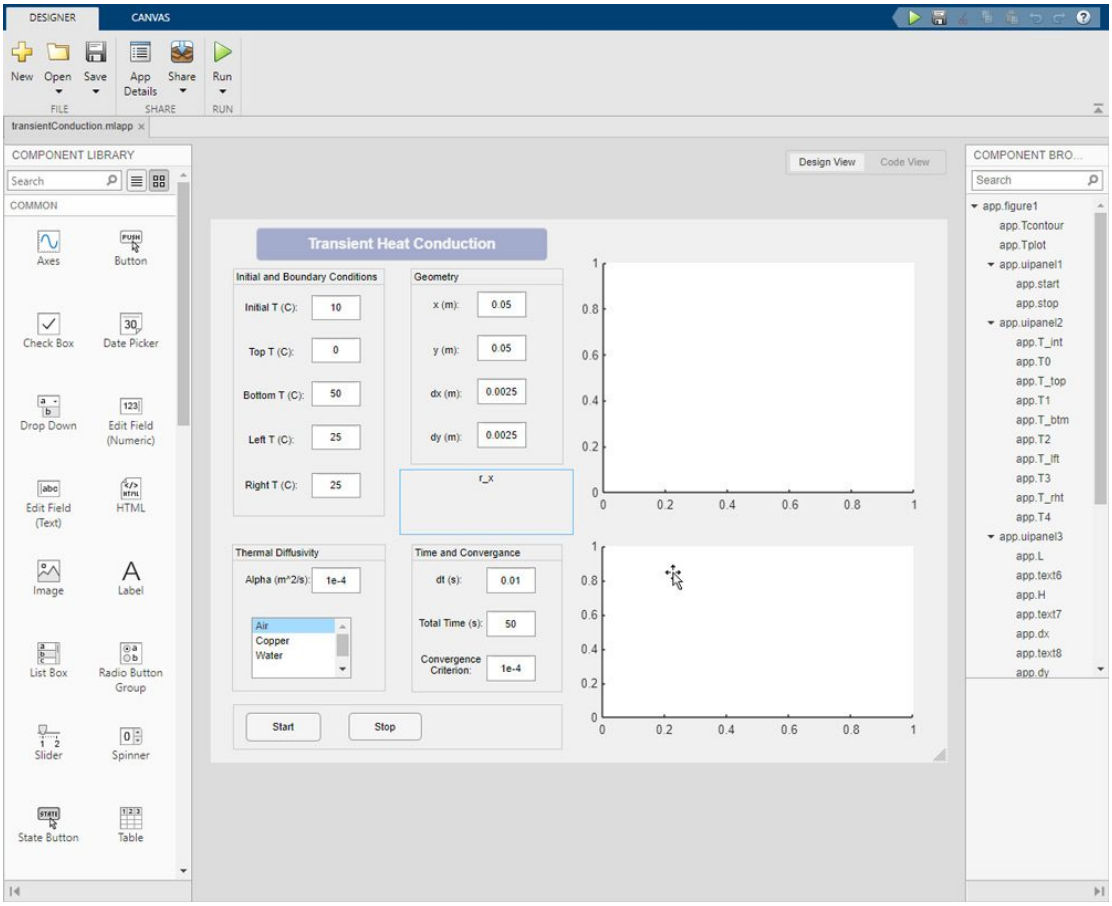
## 10.3 Wolfram Mathematica

Wolfram Mathematica je softwarový systém s vestavěnými knihovnami pro několik oblastí technických výpočtů, které umožňují strojové učení, statistické výpočty, symbolické výpočty, manipulaci s maticemi, vykreslovací grafické funkce, implementaci různých algoritmů, vytváření uživatelských rozhraní a propojení s programy napsanými v jiných programovacích jazycích.

Wolfram Mathematica má výkonné možnosti zpracování signálu, včetně návrhu digitálních a analogových filtrů, filtrování a analýzy signálu pomocí algebraických a numerických metod, které lze aplikovat na zvuk, obraz nebo jiná data. Software také poskytuje široké pokrytí jak numerické, tak symbolické Fourierovy analýzy, podporuje všechny standardní formy Fourierových transformací na datech, funkcích a sekvencích v libovolném počtu dimenzí.

Jazyk Wolfram obsahuje i vestavěné síťové programování, které umožňuje přístup k funkcím soketů TCP na všech platformách, stejně jako řadu funkcí pro připojení k síti.

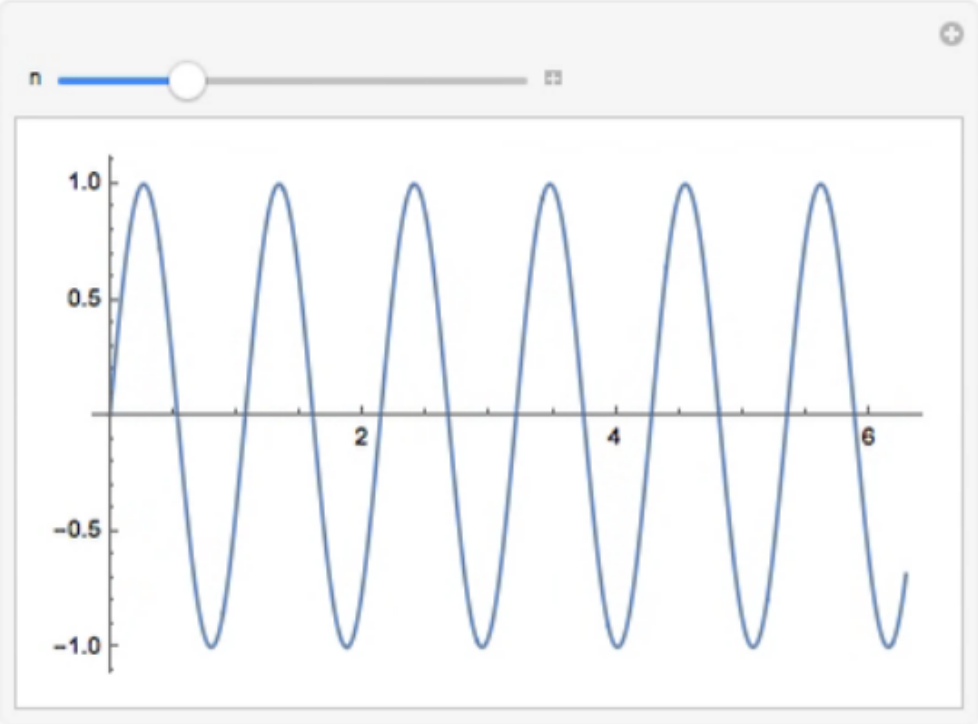
Nakonec software umožňuje vytvářet vlastní uživatelská rozhraní pomocí svých vestavěných funkcí. Velmi rychle se dají vytvářet jednoduché interaktivní aplikace, které lze bez problémů škálovat na velká aplikační rozhraní. Ukázka interaktivního prostředí je vidět na obrázku číslo 10.4. [40]




**Obrázek 10.3.** Ukázka aplikace App Designer. [39]

In[3]:= **Manipulate[Plot[Sin[n x], {x, 0, 2 Pi}], {n, 1, 20}]**

Out[3]=




**Obrázek 10.4.** Ukázka interaktivního prostředí v jazyce Mathematica. [41]

# Kapitola 11
# Závěr

Hlavním úkolem práce bylo vytvořit počítačový program, který dokáže přijímat data z TCP/IP a z přijatých dat vytvořit amplitudové spektrum, neboli waterfall graf. Ve výsledku bylo rozhodnuto vytvořit více programů v různých programovacích jazycích, aby byla možnost porovnat jejich výkonnost.

Na začátku práce byly vysvětleny matematické operace Fourierovy řady, Fourierovy transformace a rychlé Fourierovy transformace. Zejména rychlá Fourierova transformace je důležitá pro zpracovávání spektra signálu, proto byla tato transformace rozebrána více do detailu, než zbytek teorie v práci.

Až v další části práce byly vysvětleny detaily o vypracovaných programech. Pro vytvoření programu byly vybrány programovací jazyky Python a C++. Jazyky byly vybrány, protože se s nimi dobře pracuje a mají mnoho možností knihoven jak pro vizualizaci dat, tak pro vytváření grafických uživatelských rozhraní. Programy by měly pracovat primárně na operačním systému Windows.

Součástí vytváření programu bylo vytvořit různé funkce, které při prohlížení amplitudového spektra v programu pomáhají. Šlo hlavně o funkce ořezávání grafu, interpolaci spektra, kurzor, nastavení barevné mapy a ukládání a otevíraní dat.

Důležitá část práce byla popsat vytvořené programy a zhodnotil jejich výkonnost. Ukázalo se, že dva ze tří programů nemají dostatečnou rychlost zpracování dat, aby je bylo možné reálně spolehlivě využít.

Nakonec byly ještě vypsány některé již dostupné softwarové nástroje pro zobrazovaní spektra signálu. Žádný z nich nemá stejnou funkcionalitu jako vypracovaný program, ale přesto mohou být užitečné při sledování amplitudového spektra.

Program napsaný v programovacím jazyce Python s použitím knihoven PyQt a PyQtGraph všechny úkoly dané zadáním splnil. Tedy dokázal data přijímat a zobrazovat v dostatečné rychlosti a všechny zadané funkce v programu fungovaly správně. Tudíž věřím, že cíle se mi podařilo dosáhnout.

Při vytváření programu jsem se výrazně procvičil v programovacím jazyce Python a C++ a v principech objektového programování. Při tvorbě programu jsem si musel zopakovat velkou část svých znalostí o programování a naučit se úplně nové věci.

# Příloha A
## Zadání práce

ČVUT
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

# ZADÁNÍ DIPLOMOVÉ PRÁCE

### I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Petrýdes** Jméno: **Patrik** Osobní číslo: **461791**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra měření**

Studijní program: **Letectví a kosmonautika**

Studijní obor: **Avionika**

### II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

**Zobrazení Waterfall spektra na PC**

Název diplomové práce anglicky:

**Display of Waterfall spectrum on PC**

Pokyny pro vypracování:

Cílem projektu je vyvinout software pro zobrazení amplitudového spektra na PC ve formě Waterfall pro účely monitorování družicových komunikačních signálů, detekci rušení GNSS a další aplikace.
Vstup spektra bude z SDR přijímače prostřednictvím sériové linky. Software zároveň musí umět spektrum ukládat a uložené spektrum prohlížet. Software musí pracovat v reálném čase na běžném PC. Musí být schopný zobrazit spektrum o maximální délce min. 16384. 100 řádků za sekundu. Software musí umět provádět decimaci a interpolaci spektra. Uživatelské rozhraní musí podporovat nastavení barevného profilu, amplitudového měřítka, zobrazit výřez apod.

Seznam doporučené literatury:

Proakis, J. G.; Monolakis, D. G.: Digital signal processing, 4th. Prentice-Hall, 1996.

Jméno a pracoviště vedoucí(ho) diplomové práce:

**doc. Dr. Ing. Pavel Kovář, katedra radioelektroniky FEL**

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **17.09.2021** Termín odevzdání diplomové práce: **04.01.2022**

Platnost zadání diplomové práce:
**do konce zimního semestru 2022/2023**

| doc. Dr. Ing. Pavel Kovář<br>podpis vedoucí(ho) práce | podpis vedoucí(ho) ústavu/katedry | prof. Mgr. Petr Páta, Ph.D.<br>podpis děkana(ky) |
|---|---|---|

### III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

| . | |
|---|---|
| Datum převzetí zadání | Podpis studenta |

# Příloha B
# Elektronická příloha

Tato příloha popisuje elektronickou přílohu, tedy programy přiložené k práci, a poskytuje instrukce pro spouštění těchto programů.

Soubory pro program v jazyce Python, s knihovnami PyQt a PyQtGraph jsou ve složce jménem *Waterfall_Spectrum_PyQtGraph.zip* a soubory pro program v jazyce Python, s knihovnami Tkinter a Matplotlib jsou ve složce jménem *Waterfall_Spectrum_Matplotlib.zip*. Ve složkách jsou stejnojmenné skriptové soubory, tedy soubory s příponou *.py*, které obsahují zdrojový kód, textové soubory *requirements.txt*, které vypisují všechny potřebné knihovny a složka se spouštěcími soubory. Programy jsou zabalené pomocí knihovny PyInstaller, což znamená, že by program měl jít spustit na jakémkoliv systému Windows, bez jakýchkoliv požadavků. Všechny složky byly komprimované pomocí softwaru *7-Zip*.

Jsou tedy dvě možnosti, jak soubor spustit, buď pomocí připravených spouštěcích souborů, ve složkách *Waterfall_Spectrum_Matplotlib_Packaged* a *Waterfall_Spectrum_PyQtGraph_Packaged*, nebo jen pomocí zdrojového kódu. Všechny potřebné knihovny lze stáhnout pomocí knihovny pip, která je obvykle součástí standardní knihovny Pythonu, přes následující příkaz:

```
pip install -r requirements.txt
```

Kde `requirements.txt` je systémová cesta (Path) k souboru *requirements.txt*. K tomuto je ale pochopitelně třeba mít nainstalovaný Python.

Všechny soubory pro program v C++ jsou v komprimované složce *Waterfall_Spectrum_Viewer_QCustomPlot.zip*. Zdrojový kód programu v C++ je ve složce *Waterfall_Spectrum_Viewer_QCustomPlot*. Jednotlivé soubory již byly popsány v kapitole 9.3. Program je zkompilovaný, což znamená, že by měl program jít spustit, v operačním systému Windows, bez dalších požadavků. Spouštěcí soubor je ve složce *Release*. Ve složce jsou i všechny potřebné části knihoven ve formě dynamicky linkovaných knihoven. Protože jsou k dispozici všechny zdrojové kódy, lze samozřejmě program i zkompilovat a spustit pomocí vývojových prostředí nebo přímo z příkazového řádku.

# Literatura


[1] Afshin Samani. *An introduction to signal processing for non-engineers*. CRC Press/Taylor & Francis Group, 2020. ISBN 978-0-367-20755-7.

[2] Alan V Oppenheim a Alan S Willsky. *Signals & systems*. 2 vyd.. Prentice Hall, 1997. ISBN 0-13-814757-4.

[3] Jonah Gamba. *Radar signal processing for autonomous driving*. Springer Nature, 2020. ISBN 978-981-13-9193-4.

[4] G. D. Bergland. A guided tour of the fast Fourier transform. *IEEE Spectrum*. 1969, 6 (7), 41-52. DOI 10.1109/MSPEC.1969.5213896.

[5] Julius O. Smith III. *Mathematics of the Discrete Fourier Transform (DFT), with Audio Applications*. 2 vyd.. W3K Publishing, 2007. ISBN 978-0-9745607-4-8.

[6] Abdul Rahim Abdullah, Ahmad Zuri Sha'ameri a Norhashimah Mohd Saad. Power quality analysis using spectrogram and gabor transformation. *2007 Asia-Pacific Conference on Applied Electromagnetics*. 2007, 1-5. DOI 10.1109/APACE.2007.4603964.

[7] Nick Messitte. *Understanding Spectrograms*. 2020. `https://www.izotope.com/en/learn/understanding-spectrograms.html`.

[8] Habibur Rahman a Md Mamunoor Islam. A Practical Approach to Spectrum Analyzing Unit using RTL-SDR. *Rajshahi University Journal of Science and Engineering*. 2016, 44 151. DOI 10.3329/rujse.v44i0.30400.

[9] Pavel Puricer, Pavel Kovář a Miroslav Barta. Modernized Solar Radio Spectrograph in the L Band Based on Software Defined Radio. *Electronics*. 2019, 8 861. DOI 10.3390/electronics8080861.

[10] Bjarne Stroustrup. *The C++ programming language*. 4 vyd.. Boston, MA: Addison-Wesley Educational, 2013. ISBN 0-321-56384-0.

[11] The C++ Resource Network. *The Features of C++ as a Language*. `https://www.cplusplus.com/info/description`.

[12] Reed Cartwright. *Python Crash Course : The Ultimate Novice's Program to Discovering Python Programs in Under 12 Hours*. 2022.

[13] *The Complete Python Coding & Programming Manual*. 13 vyd.. Black Dog Media Limited, 2022.

[14] Luciano Ramalho. *Fluent python: Clear, concise, and Effective Programming*. O'Reilly Media, 2022. ISBN 978-1-492-05635-5.

[15] Matt Harrison. *Illustrated guide to Python 3 : a complete walkthrough of beginning Python with unique illustrations showing how Python really works*. CreateSpace Independent Publishing Platform, 2017.

[16] The C++ Resource Network. *History of C++*. `https://www.cplusplus.com/info/history/`.

[17] Bjarne Stroustrup. *The design and evolution of C++*. Boston, MA: Addison Wesley, 1994. ISBN 0-201-54330-3.

[18] Robert Lafore. *Object-oriented programming in C++*. 4 vyd.. Indianapolis, IN: Sams Publishing, 2001. ISBN 0-672-32308-7.

[19] Alan D Moore. *Python GUI Programming with Tkinter*. Birmingham, England: Packt Publishing, 2018. ISBN 978-1-78883-588-6.

[20] John E Grayson. *Python & Tkinter Programming*. London, England: Manning, 1999. ISBN 1-884777-81-3.

[21] Ivan Idris. *NumPy: Beginner's guide*. 3 vyd.. Birmingham, England: Packt Publishing, 2015. ISBN 978-1-78528-196-9.

[22] Travis Oliphant. *Guide to NumPy*. 2006.

[23] Mario Dobler a Tim Grossmann. *Data Visualization with Python*. Birmingham, England: Packt Publishing, 2019. ISBN 978-1-78995-646-7.

[24] Sandro Tosi. *Matplotlib for Python Developers*. Birmingham, England: Packt Publishing, 2010. ISBN 978-1-847197-90-0.

[25] Martin Fitzpatrick. *Create GUI Applications with Python & Qt6*. 2021.

[26] Luke Campagnola. *PyQtGraph*. 2021. `https://www.pyqtgraph.org/`.

[27] David Cortesi. *PyInstaller Manual*. `https://pyinstaller.org/`.

[28] Lee Zhi Eng. *Qt5 C++ GUI Programming Cookbook*. Birmingham, England: Packt Publishing, 2016.

[29] Emanuel Eichhammer. *QCustomPlot*. `https://www.qcustomplot.com/index.php/introduction`.

[30] Nathan Jennings. *Socket Programming in Python (Guide)*. `https://realpython.com/python-sockets/`.

[31] Alex Martelli. *Python in a Nutshell*. Sebastopol, CA: O'Reilly Media, 2003. ISBN 0-596-00188-6.

[32] Space Weather Prediction Center. *Space weather impacts*. `https://www.swpc.noaa.gov/impacts`.

[33] Yeqiu Ying, Timothy Whitworth a Kevin Sheridan. GNSS interference detection with software defined radio. *2012 IEEE First AESS European Conference on Satellite Telecommunications (ESTEL)*. 2012, 1-6. DOI 10.1109/ESTEL.2012.6400121.

[34] Michal Krenek. *QSpectrumAnalyzer*. `https://github.com/xmikos/qspectrumanalyzer`.

[35] Rudra Pratap. *Getting Started with MATLAB*. New York, NY: Oxford University Press, 2009. ISBN 978-0-19-973124-4 .

[36] MathWorks Help Center. *Signal Processing Toolbox*. `https://www.mathworks.com/help/signal/`.

[37] MathWorks Help Center. *Signal Analyzer*. `https://www.mathworks.com/help/signal/ref/signalanalyzer-app.html`.

[38] MathWorks Help Center. *Parallel Computing Toolbox*. `https://www.mathworks.com/help/parallel-computing/`.

[39] MathWorks. *MATLAB GUI*. `https://www.mathworks.com/discovery/matlab-gui.html`.

[40] Stephen Wolfram. *The Mathematica Book*. 5 vyd.. Wolfram Media, 2003.

[41] Wolfram Documentation Center. *Introduction to ManipulateI*. `https://reference.wolfram.com/language/tutorial/IntroductionToManipulate.html`.

[42] Vratislav Davídek a Pavel Sovka. *Číslicové zpracovaní signalů a implementace*. ČVUT, 1996.

[43] E Brigham. *The fast Fourier transform and its applications*. Prentice Hall, 1988. ISBN 0-13-307505-2.

[44] Charles R. Harris, K. Jarrod Millman, Stéfan J. vander Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke a Travis E. Oliphant. Array programming with NumPy. *Nature*. 2020, 585 (7825), 357–362. DOI 10.1038/s41586-020-2649-2.

[45] J. D. Hunter. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*. 2007 , 9 (3), 90–95. DOI 10.1109/MCSE.2007.55.

[46] John G. Proakis a Dimitris G. Manolakis. *Digital signal processing : principles, algorithms, and applications*. Prentice Hall, 1995. ISBN 0-13-394338-9.