VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

WEBOVÉ APLIKACE PRO PODPORU VÝUKY ZÁKLADŮ ZPRACOVÁNÍ SIGNÁLŮ

WEB APPLICATIONS SUPPORTING EDUCATION OF SIGNAL PROCESSING FUNDAMENTALS

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — Bc. Dominik Kuře

AUTHOR

VEDOUCÍ PRÁCE — prof. Mgr. Pavel Rajmic, Ph.D.

SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní program **Audio inženýrství**
specializace Zvuková produkce a nahrávání
Ústav telekomunikací

Student: Bc. Dominik Kuře ***ID:*** 203736
Ročník: 2 ***Akademický rok:*** 2022/23

NÁZEV TÉMATU:

Webové aplikace pro podporu výuky základů zpracování signálů

POKYNY PRO VYPRACOVÁNÍ:

V jazyce JavaScript vytvořte čtyři webové aplikace, které poslouží jako interaktivní podpora výuky v kurzech zaměřených na zpracování signálů. Aplikace budou tématicky zaměřeny na výpočet, význam a názornou ukázku střední a efektivní hodnoty signálu, posunutí, změnu měřítka a zesílení signálu, vliv posunutí a změny měřítka na Fourierovu řadu signálu a převzorkování signálu pomocí různých typů interpolací (např. nejbližší soused, lineární, kubické, sinc). Při tvorbě aplikací se zaměřte především na názornou podobu a funkčnost pro potřebu výuky.

DOPORUČENÁ LITERATURA:

[1] Smékal, Z.: Analýza signálu a soustav. Vysoké učení technické v Brně, 2012. ISBN 978-80-214-4453-9.
[2] Smith, S. W.: The scientist and engineer's guide to digital signal processing. San Diego: California Technical Pub., 1997. ISBN 09-660-1763-3.

Termín zadání: 6.2.2023 ***Termín odevzdání:*** 19.5.2023

Vedoucí práce: prof. Mgr. Pavel Rajmic, Ph.D.

doc. Ing. Jiří Schimmel, Ph.D.
předseda rady studijního programu

ABSTRAKT

Tématem práce je tvorba čtyř webových aplikací, které mají sloužit jako výukový materiál pro studenty, kteří se seznamují se základy zpracování signálů. Jednotlivými oblastmi, na které se aplikace zaměřují, jsou střední a efektivní hodnota signálů, základní operace se signály (zesílení, posunutí, změna měřítka), vliv těchto operací na Fourierovu řadu signálů a také převzorkování signálu pomocí různých druhů interpolací (metoda nejbližšího souseda, lineární interpolace, kubická interpolace a interpolace pomocí funkce sinus cardinalis). Tyto aplikace jsou realizovány pomocí jazyka TypeScript, jenž je rozšířením jazyka JavaScript o statické datové typy. Dále je využita knihovna React, jež slouží pro tvorbu front–endových aplikací a Chart.js, umožňující pohodlnou a velmi detailní práci s grafy. V teoretické části aplikace jsou nejprve v první polovině popsány jednotlivé oblasti zpracování signálů, které je potřeba znát pro vypracování aplikací, a poté jsou představeny informační technologie, jež jsou využity pro implementaci. Kromě již zmíněných technologií se text okrajově zabývá i úvodem do jazyků HTML a CSS a také krátce pojednává o syntaxi JSX. Praktická část popisuje samotnou realizaci těchto aplikací a slouží též jako dokumentace ke zdrojovému kódu. V praktické části je popsáno, jak realizovat jednotlivé druhy signálů (sinus, trojúhelníkový, pilový, obdélníkový s různými střídami, šumový) a jak pro každý z těchto signálů vypočítat Fourierovu řadu, jak lze v kódu provést jednotlivé signálové operace, jakým způsobem lze realizovat různé druhy interpolací a jaké jsou některé konkrétní možnosti výpočtu kubické interpolace (metoda konečných diferencí, kardinální spline, Catmull–Rom spline, přirozená kubická interpolace), jak tyto aplikace vypadají a jaké jsou jejich struktury.

KLÍČOVÁ SLOVA

efektivní hodnota, Fourierova řada, Chart.js, interpolace, JavaScript, React, sinc, střední hodnota, TypeScript, webové aplikace, zpracování signálů.

ABSTRACT

The main topic of the thesis is the creation of four web applications which are used as learning material for students who are studying the basics of signal processing. The areas on which the thesis focuses on are root mean square and expected value of signals, basic signal operations (amplification, geometric translation, scale change), the effect these operations have on the Fourier series of the given signal and also resampling of a signal using different methods of interpolation (nearest neighbour method, linear interpolation, cubic interpolation and interpolation using the sinc function). These applications are implemented using the TypeScript programming language which is an extension of the JavaScript language which enhances it with static types. Other libraries that are used are the React library which is used for front-end web applications and a library which allows easy to implement but still very detailed manipulation of charts called Chart.js. The first half of the theoretical part of the thesis focuses on those areas of signal processing which are necessary to understand so the applications can be created. The second half focuses on information technologies used for the implementation of said applications. Besides the already mentioned technologies, the text also briefly mentions the basics of HTML and CSS languages as well as the JSX syntax. The practical part describes how the applications were implemented and also serves as documentation for the source code. This part shows the reader how to create differently shaped signals in code (sine, triangle, sawtooth, square with different duty cycles, noise) and how to obtain the Fourier series of each of these signals, how to implement different signal operations, how to interpolate between multiple points using different interpolation formulas and what are some of the methods which can be used to apply cubic interpolation (finite difference method, cardinal spline, Catmull–Rom spline, natural cubic interpolation), what the applications look like and what is their structure.

KEYWORDS

root mean square, Fourier series, Chart.js, interpolation, JavaScript, React, sinc, expected value, TypeScript, web applications, signal processing.

KUŘE, Dominik. *Webové aplikace pro podporu výuky základů zpracování signálů*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2023, 74 s. Diplomová práce. Vedoucí práce: prof. Mgr. Pavel Rajmic, Ph.D.

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Bc. Dominik Kuře
VUT ID autora:	203736
Typ práce:	Diplomová práce
Akademický rok:	2022/23
Téma závěrečné práce:	Webové aplikace pro podporu výuky základů zpracování signálů

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

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

Brno ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu prof. Mgr. Pavlu Rajmicovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci. Dále bych rád poděkoval své rodině za podporu a svým kolegům v Kiwi.com za shovívavost ke studiu při práci.

Obsah

Seznam obrázků

Seznam výpisů

Úvod

Diplomová práce se zabývá několika oblastmi v rámci zpracování signálů, z nichž jsou poté některé koncepty převedeny do názorné a interaktivní podoby webové aplikace, které budou dostupné na adrese `https://www.utko.fekt.vut.cz/~rajmic/applets`. Tyto aplikace budou následně sloužit pro výukové účely.

Nachází se zde teorie potřebná k realizaci a pochopení těchto čtyř zadaných aplikací a v rámci praktické části je poté popsán postup implementace a technologie, jež byly využity pro realizaci aplikací. Témata jednotlivých aplikací jsou následující:

- Střední a efektivní hodnoty signálu
- Základní operace se signálem – změna měřítka, zesílení a posunutí
- Vliv základních operací se signálem na jeho Fourierovu řadu
- Převzorkování signálu pomocí různých druhů interpolací

První tři aplikace umožňují uživateli navolit některý ze základních typů signálů – harmonický, pilový, obdélníkový s různými střídami, trojúhelníkový a některé aplikace obsahují i šumový signál. Poté je uživateli nabídnuto několik posuvníků, kterými lze měnit různé parametry signálů a pozorovat jejich vliv na oblast zájmu dané aplikace (střední a efektivní hodnotu, Fourierovu řadu, ...).

Aplikace zaměřená na interpolaci nabídne uživateli několik bodů, jejichž hodnotu lze pomocí myši měnit a pomocí tlačítek je možné vypínat jednotlivé druhy interpolací – nejbližší soused, lineární, několik druhů kubické interpolace a také interpolaci pomocí funkce sinc.

1 Teoretická část studentské práce

Teoretickou část lze vzhledem k povaze zadání rozdělit do dvou oblastí – nejprve budou uvedeny vybrané kapitoly základů signálů a soustav, kterým je nutné porozumět před samotnou implementací jednotlivých aplikací. Druhá část objasní vybrané technologie použité k realizaci aplikace.

1.1 Vybrané kapitoly ze signálů a soustav

1.1.1 Střední a efektivní hodnota signálu

V praxi je často pro popis periodických signálů využito střední a efektivní hodnoty. První zadaná aplikace se zabývá výpočtem a vykreslením těchto hodnot do grafu pro různé průběhy signálu.

Střední hodnota

Střední hodnota je dána vztahem:

$$I_s = \frac{1}{T}\int_0^T i(t)\mathrm{d}t. \tag{1.1}$$

Střední hodnota je také označována jako stejnosměrná složka signálu. Jde o průměrnou hodnotu funkce s časovým argumentem za dobu trvání jedné periody. Fyzikálně ji lze popsat např. s využitím proudu, kdy střední hodnota je taková hodnota stejnosměrného proudu, která přenese stejně velké množství elektrického náboje, jako proud původní. V geometrickém vyjádření je střední hodnota I_s rovna délce strany obdélníka, který má stejnou plochu, jako je plocha pod původní křivkou $i(t)$. [1]

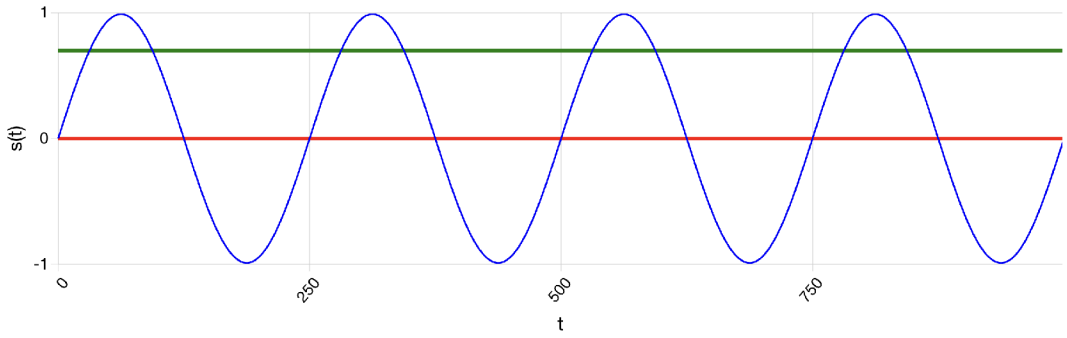
Efektivní hodnota

Efektivní hodnota je dána vztahem:

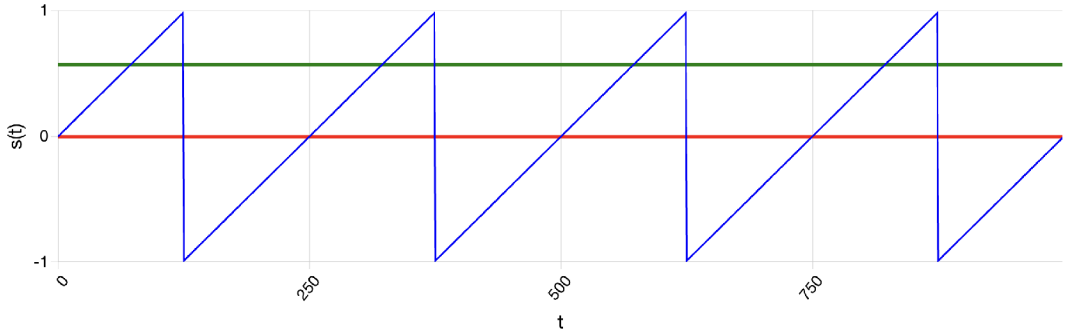
$$I_{ef} = \sqrt{\frac{1}{T}\int_0^T i^2(t)\mathrm{d}t}. \tag{1.2}$$

Efektivní hodnota proměnného periodického signálu je taková hodnota signálu stejnosměrného, který vykoná za stejný čas stejnou práci jako signál střídavý. V praxi se setkáváme s efektivní hodnotou v elektrotechnice při měření proudu či napětí. Zařízení, která slouží k měření efektivní hodnoty bývají označena zkratkou RMS (z anglického „Root mean square“, čili podle kvadratického průměru). Výhodou užívání této hodnoty je, že přímo reprezentuje energetické účinky daného signálu. [1]

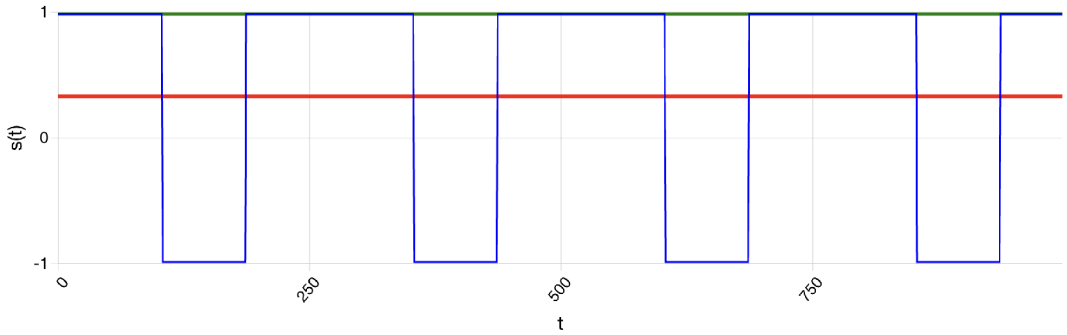
Na následujících obrázcích se nachází několik příkladů výpočtu efektivní a střední hodnoty pro různé signály, které jsou podrobněji rozebrány v praktické části diplomové práce:



Obr. 1.1: Zobrazení střední (červená) a efektivní (zelená) hodnoty signálu funkce $\sin\left(\frac{x\pi}{125}\right)$.



Obr. 1.2: Zobrazení střední (červená) a efektivní (zelená) hodnoty pilového signálu.



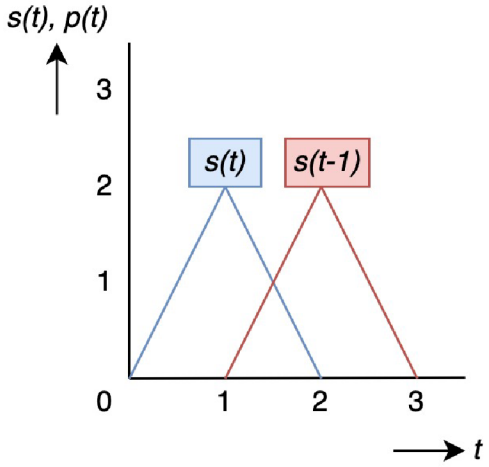
Obr. 1.3: Zobrazení střední (červená) a efektivní (zelená) hodnoty obdélníkového signálu se střídou 2:1.

1.1.2 Základní operace se signálem

V této sekci bude představeno několik základních operací se signálem, které je potřeba znát pro účely vytvoření druhé aplikace v projektu. Signály je možné dělit podle několika kritérií, jedním z nich je rozdělení signálů na spojité a diskrétní. Pro popis následujících operací budou uvažovány signály spojité.

Posunutí

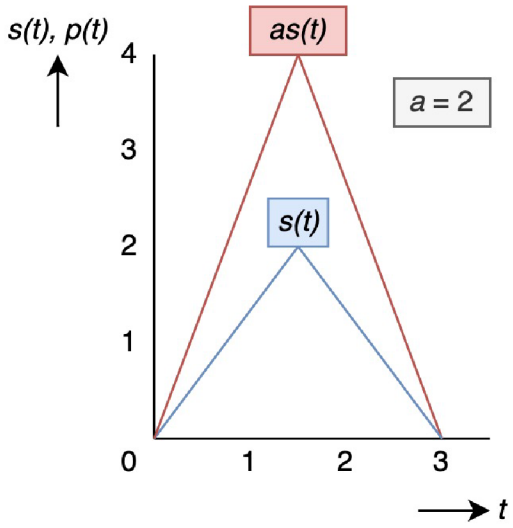
Časové posunutí je operace, která jednomu signálu $s(t)$ přiřadí signál $s(t-\tau)$, kde $\tau \in \mathbb{R}$. [2]



Obr. 1.4: Časové posunutí signálu $p(t) = s(t-1)$.

Zesílení a zeslabení

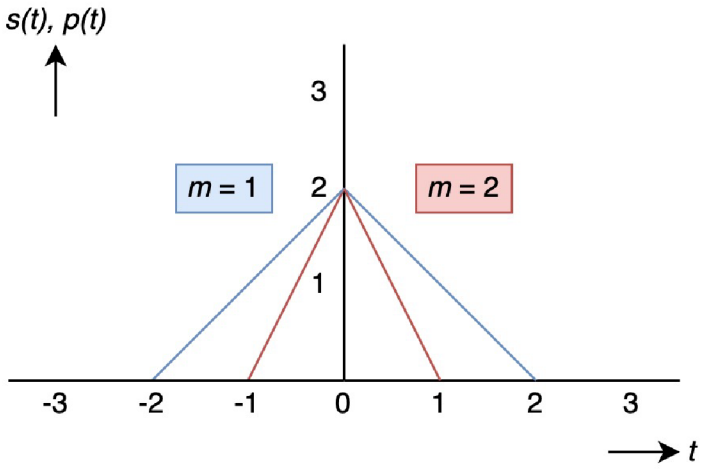
Změny velikosti signálu $s(t)$ lze dosáhnout pomocí násobení konstantou $a \in \mathbb{R}$, která signál zmenší, pokud $0 < a < 1$ nebo jej zvětší, pokud $a > 1$. Pro záporné a dochází k překlopení signálu okolo osy x. [2]



Obr. 1.5: Zesílení signálu $p(t) = 2s(t)$.

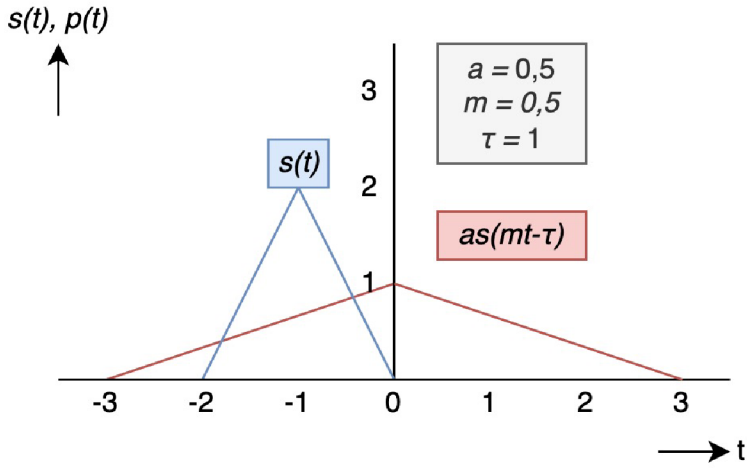
Změna časového měřítka

Změna časového měřítka je operace, při které je signál $s(t)$ nahrazen signálem $s(mt)$, $m \in \mathbb{R}$, $m \neq 1$. Pokud je $m < 1$, dochází k roztažení časového měřítka, při $m > 1$ dochází naopak k jeho smrsknutí. [2]



Obr. 1.6: Změna časového měřítka signálu $p(t) = s(2t)$.

Tyto základní operace lze libovolně kombinovat k dosažení požadovaného výsledného signálu.



Obr. 1.7: Kombinace tří základních operací se signály $p = \frac{1}{2}s(\frac{1}{2}t - 1)$.

1.1.3 Fourierova řada

Periodický signál

Signál $s(t)$ je periodický, pokud existuje číslo $T_1 \in \mathbb{R}$, které je kladné ($T_1 > 0$), a pro které platí, že pro všechna reálná t:

$$s(t + T_1) = s(t). \tag{1.3}$$

Nejmenší hodnota T_1, která splňuje tuto podmínku, se nazývá základní perioda. Frekvenční spektrum periodických signálů se skládá z harmonických funkcí. Pokud např. existuje signál, jehož průběh se opakuje s frekvencí 1000 Hz (perioda je tedy 1 ms), bude spektrum obsahovat první harmonickou složku právě na 1000 Hz, druhou na 2000 Hz, třetí na 3000 Hz atd. První z těchto harmonických složek se označuje jako fundament. Na spektrum lze nahlížet dvěma způsoby – je-li uvažováno spektrum spojité, hodnoty v tomto spektru jsou nulové všude kromě frekvencí harmonických složek. Druhý způsob předpokládá spektrum diskrétní, kdy je spektrum definováno pouze na frekvencích s přítomnou harmonickou složkou. Jinými slovy lze na frekvence mezi těmito harmonickým složkami nahlížet jako na nulové či neexistující. Tyto frekvence se nijak nepodílejí na původním periodickém signálu. [3]

Fourierova řada

Výpočet Fourierovy řady umožňuje aproximovat periodické funkce jako součet harmonických funkcí sinus a kosinus. [4] Její využití nalézáme např. v akustice a elektrotechnice.

Fourierova řada v goniometrickém tvaru:

$$f(t) \approx \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos n\omega t + b_n \sin n\omega t, \tag{1.4}$$

kde:

$$\omega = \frac{2\pi}{T}, n = 1, 2, 3, \ldots,$$

a pro jednotlivé koeficienty Fourierovy řady platí:

$$a_0 = \frac{1}{T} \int_0^T f(t) \mathrm{d}t,$$

$$a_n = \frac{2}{T} \int_0^T f(t) \cos n\omega t \mathrm{d}t,$$

$$b_n = \frac{2}{T} \int_0^T f(t) \sin n\omega t \mathrm{d}t.$$

Hodnota koeficientu a_0 je shodná se střední hodnotou signálu. Na hodnotu lze také nahlížet jako na kosinový signál s nulovou frekvencí (tedy jako na konstantu). V rámci výpočtu Fourierovy řady není uvažován žádný koeficient b_0, neboť by se jednalo o sinusový signál s nulovou frekvencí, tedy hodnota tohoto koeficientu by byla vždy nulová. [3]

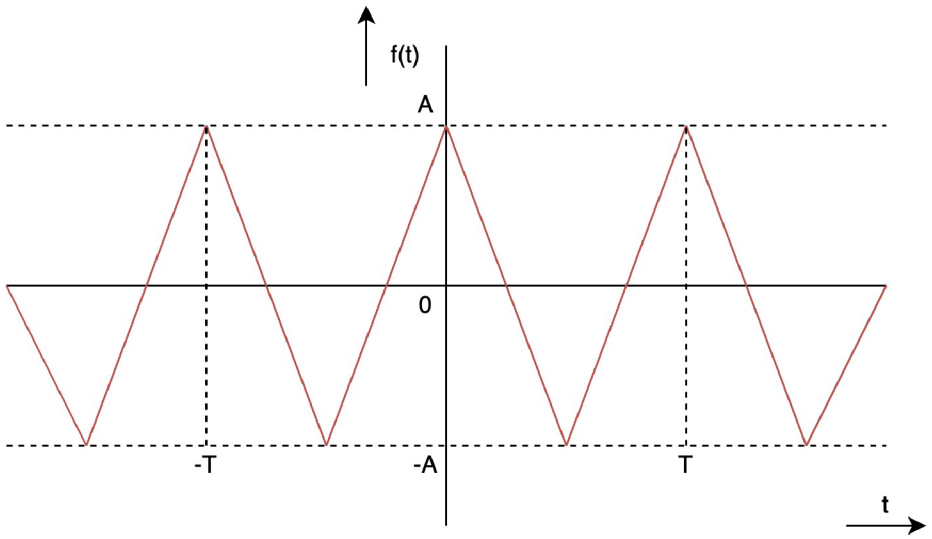
Výpočet Fourierovy řady pro určité typy periodických signálů lze zjednodušit na základě jejich parity. Pokud je funkce definující periodický signál lichá na intervalu $< -\frac{T}{2}, \frac{T}{2} >$, poté je signál redukován na sinusovou řadu. [4] Hodnoty b_1, b_2, b_3,. . . jsou amplitudami těchto jednotlivých sinusových signálů a jedná se o imaginární část frekvenčního spektra.

$$a_n = 0, b_n = \frac{4}{T} \int_0^{\frac{T}{2}} f(t) \sin n\omega t \mathrm{d}t. \tag{1.5}$$

Pokud je funkce definující periodický signál sudá na intervalu $< -\frac{T}{2}, \frac{T}{2} >$, poté je signál redukován na kosinovou řadu. Hodnoty a_1, a_2, a_3,. . . jsou amplitudami těchto jednotlivých kosinových signálů a jedná se o reálnou část frekvenčního spektra.

$$b_n = 0, a_n = \frac{4}{T} \int_0^{\frac{T}{2}} f(t) \cos n\omega t \mathrm{d}t. \tag{1.6}$$

Příklad výpočtu Fourierovy řady



Obr. 1.8: Příklad výpočtu Fourierovy řady – trojúhelníkový signál.

Uveďme příklad výpočtu Fourierovy řady. Vstupní trojúhelníkový signál je zadán pomocí obrázku 1.8. Jedná se o sudou funkci, tudíž koeficienty b_n jsou nulové. Z průběhu je také patrné, že střední hodnota signálu je nulová, tudíž koeficient a_0 je také nulový. Je tedy potřeba vypočítat kosinovou řadu koeficientů a_n. [5]
Mezi $t = 0$ a $t = \frac{T}{2}$ je funkce definována následovně:

$$f(t) = A - \frac{4A}{T}t.$$

Výpočet:

$$a_n = \frac{2}{T}\int_0^T f(t)\cos n\omega t\mathrm{d}t = \frac{4}{T}\int_0^{\frac{T}{2}} f(t)\cos n\omega t\mathrm{d}t = \frac{4}{T}\int_0^{\frac{T}{2}}\left(A - \frac{4A}{T}t\right)\cos n\omega t\mathrm{d}t =$$

$$= \frac{4A}{T}\left(\int_0^{\frac{T}{2}}\cos n\omega t\mathrm{d}t - \frac{4}{T}\int_0^{\frac{T}{2}} t\cos n\omega t\mathrm{d}t\right).$$

Po integraci získáme následující tvar:

$$a_n = \frac{4A}{T}\left(\frac{T\sin(\pi n)}{2\pi n} + \frac{4}{T}\frac{T^2(2\sin(\frac{\pi n}{2})^2 - \pi n\sin(\pi n))}{4\pi^2 n^2}\right).$$

Protože $\sin(\pi n) = 0$:

$$a_n = \frac{4A}{T}\frac{4}{T}\frac{T^2 2\sin(\frac{\pi n}{2})^2}{4\pi^2 n^2} = \frac{8A\sin(\frac{\pi n}{2})^2}{\pi^2 n^2}.$$

Protože při $n = 0, 1, 2, 3, \ldots$ platí, že $\sin(\frac{\pi n}{2})^2 = 0, 1, 0, 1, \cdots = \frac{1-(-1)^n}{2}$.

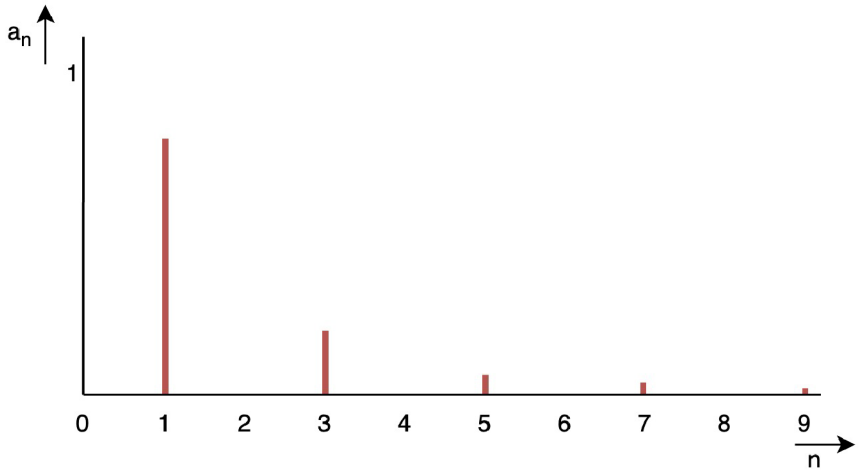
Získáváme následující vztah pro koeficienty a_n:

$$a_n = \begin{cases} 4A\frac{1-(-1)^n}{\pi^2 n^2}, & \text{pro sudé n} \\ 0, & \text{pro liché n} \end{cases}$$

Po dosazení $A = 1$ získáváme:

n	0	1	2	3	4	5	6	7	8	9
a_n	0	0,810	0	0,0901	0	0,0324	0	0,0165	0	0,0100

Výsledek výpočtu Fourierovy řady lze zobrazit následujícím grafem:



Obr. 1.9: Fourierova řada trojúhelníkového signálu.

1.1.4 Převod mezi jednotlivými tvary Fourierovy řady

Pro převod mezi goniometrickým a exponenciálním tvarem je využíván Eulerův vzorec:

$$\mathrm{e}^{\mathrm{j}x} = \cos x + \mathrm{j}\sin x. \tag{1.7}$$

A z něj následující odvozené vztahy:

$$\mathrm{e}^{-\mathrm{j}x} = \cos x - \mathrm{j}\sin x. \tag{1.8}$$

$$\cos x = \frac{\mathrm{e}^{\mathrm{j}x} + \mathrm{e}^{-\mathrm{j}x}}{2}. \tag{1.9}$$

$$\sin x = \frac{\mathrm{e}^{\mathrm{j}x} - \mathrm{e}^{-\mathrm{j}x}}{2\mathrm{j}}. \tag{1.10}$$

Převodem z goniometrického tvaru uvedeného v rovnici 1.4 lze s využitím vzorců 1.8, 1.9 a 1.10 dosáhnout exponenciálního tvaru rovnice Fourierovy řady. [2]

$$f(t) = \sum_{-\infty}^{\infty} c_n \mathrm{e}^{\mathrm{j}n\omega t}, \tag{1.11}$$

kde $n \in \mathbb{Z}, c_0 = a_0$ (střední hodnota). Při využití následujícího vzorce:

$$\cos(\alpha + \beta) = \cos\alpha \cos\beta - \sin\alpha \sin\beta, \tag{1.12}$$

lze získat následující tvar rovnice Fourierovy řady:

$$f(t) = \frac{C_0}{2} + \sum_{n=1}^{\infty} C_n \cos(n\omega_1 t + \varphi_n). \tag{1.13}$$

Pro jednotlivé koeficienty získáváme tvary:

$$a_n = C_n \cos\varphi_n.$$

$$b_n = -C_n \sin\varphi_n.$$

A platí rovnost:

$$C_n \cos(n\omega_1 t + \varphi_n) = a_n \cos n\omega_1 t + b_n \sin n\omega_1 t. \tag{1.14}$$

Přepočet mezi koeficienty a amplitudovým/fázovým spektrem je dán pomocí vztahů:

$$C_n = \sqrt{a_n^2 + b_n^2}, \tag{1.15}$$

$$\varphi_n = \arctan\left(\frac{-b_n}{a_n}\right). \tag{1.16}$$

Signál z časové oblasti lze tedy převést na frekvenční spektrum, které je složeno z modulového a fázového spektra.

1.1.5 Vliv operací se signálem na spektrum Fourierovy řady

Posunutí

Časové posunutí se projevuje ve fázovém spektru, modulové spektrum zůstává nepozměněno. Pokud je rovnice posunutého signálu:

$$s_{p\tau} = s_p(t - \tau), \tag{1.17}$$

poté platí vztah:

$$c_{\tau n} = c_n \mathrm{e}^{-\mathrm{j}n\omega\tau}, \tag{1.18}$$

kde $c_{\tau n}$ označuje koeficienty posunutého signálu a c_k označuje koeficienty původního signálu. [2]

Pro výsledné fázové (argumentové) spektrum tedy platí:

$$\arg c_{\tau n} = \arg c_n - n\omega\tau. \tag{1.19}$$

Přepočet posunu pro jednotlivé koeficienty je možné získat ze vztahů pro Fourierovu řadu následujícím způsobem. Nechť jsou a_n a b_n koeficienty Fourierovy řady funkce $f(t)$ v goniometrickém tvaru. Označme funkci obsahující posunutí jako $g(t)$:

$$g(t) = f(t - \tau),$$

a její Fourierovy koeficienty označme jako a'_n a b'_n. Pro zjednodušení výpočtu umístíme sinusovou komponentu do imaginární oblasti:

$$a'_n + \mathrm{j}b'_n = \frac{T}{2}\int_0^T g(t)\Big(\cos(n\omega t) + \mathrm{j}\sin(n\omega t)\Big)\mathrm{d}t =$$
$$= \frac{T}{2}\int_0^T g(t)\mathrm{e}^{\mathrm{j}n\omega t}\mathrm{d}t = \frac{T}{2}\int_0^T f(t-\tau)\mathrm{e}^{\mathrm{j}n\omega t}\mathrm{d}t.$$

Zavedeme substituci, v níž se nachází posunutí o τ:

$$|t' = t - \tau, \mathrm{d}t' = \mathrm{d}t|.$$

$$a'_n + \mathrm{j}b'_n = \frac{T}{2}\int_{-\tau}^{T-\tau} f(t')\mathrm{e}^{\mathrm{j}n\omega(t'+\tau)}\mathrm{d}t',$$

Zvolíme další substituci $K = \mathrm{e}^{\mathrm{j}n\omega\tau}$ s využitím $\mathrm{e}^{\mathrm{j}n\omega(t'+\tau)} = \mathrm{e}^{\mathrm{j}n\omega t'}\mathrm{e}^{\mathrm{j}n\omega\tau}$,

$$a'_n + \mathrm{j}b'_n = K\frac{T}{2}\int_{\tau}^{T+\tau} f(t')\mathrm{e}^{\mathrm{j}n\omega t'}\mathrm{d}t'.$$

Je možné posunout meze integrálu, neboť se jedná o součin periodických funkcí na intervalu jedné periody:

$$a'_n + \mathrm{j}b'_n = \frac{T}{2}\int_0^T g(t')\mathrm{e}^{\mathrm{j}n\omega t'}\mathrm{d}t' = K\frac{T}{2}\int_0^T f(t')\mathrm{e}^{\mathrm{j}n\omega t'}\mathrm{d}t' = K[a_n + \mathrm{j}b_n],$$

a protože $K = \cos(n\omega\tau) + \mathrm{j}\sin(n\omega\tau)$, získáváme vztah pro posunuté koeficienty:

$$a'_n = \cos(n\omega\tau)a_n - \sin(n\omega\tau)b_n, \tag{1.20}$$

$$b'_n = \sin(n\omega\tau)a_n + \cos(n\omega\tau)b_n. \tag{1.21}$$

Při vyjádření Fourierovy řady pomocí sinusové a kosinové řady dochází při posunutí signálu o určitou část periody k adekvátnímu posunu hodnot koeficientů podle otáčení v komplexní (Gaussově) rovině, kde osy reprezentují reálnou a imaginární složku signálu. Nejjednodušším příkladem může být sinusový signál, pro který jsou hodnoty koeficientů při posunu následující (pro amplitudu $A = 1$ a velikost kroku posunutí $\frac{T}{4}$):

posun	0	$\frac{T}{4}$	$\frac{T}{2}$	$\frac{3T}{4}$	T
b_1	1	0	-1	0	1
a_1	0	1	0	-1	0

Tabulka zobrazující hodnoty pokračuje do nekonečna se stejnými opakovanými hodnotami. Hodnota každého dalšího koeficientu se mění při stejném posunu n-krát rychleji než u prvního koeficientu, kde n je pořadí koeficientu, neboť n-tý koeficient reprezentuje sinusový nebo kosinový signál n-násobné frekvence prvního koeficientu.

Zesílení

Změna zesílení (tedy násobení signálu konstantou) vstupního signálu je úměrná změně zesílení modulového spektra Fourierovy řady, fázové spektrum není pozměněno. [2] Výpočet Fourierovy řady je lineárním zobrazením, tudíž vektorové operace sčítání a násobení konstantou zůstávají zachovány.

$$As_p(t) = A \sum_{n=-\infty}^{\infty} c_n \mathrm{e}^{\mathrm{j}n\omega t} = \sum_{n=-\infty}^{\infty} (Ac_n)\mathrm{e}^{\mathrm{j}n\omega t}. \tag{1.22}$$

Změna měřítka

Při změně časového měřítka zůstávají hodnoty koeficientů ve fázovém i modulovém spektru zachovány, mění se však časová i kmitočtová osa. Pokud je rovnice popisující signál s pozměněným měřítkem:

$$s_p(t) = s(mt), \tag{1.23}$$

kde $m > 0$, $m \in \mathbb{R}$, poté pro nový signál platí:

$$T_p = \frac{T}{m}, \omega_p = m\omega, \tag{1.24}$$

kde ω je úhlový kmitočet původního signálu a T je perioda původního signálu. Pro koeficienty platí:

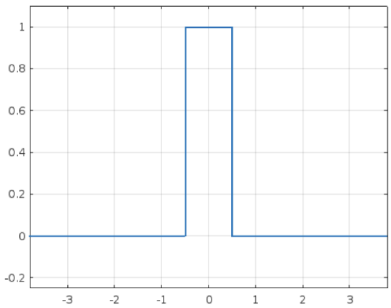
$$c_{pk} = c_k. \tag{1.25}$$

Velikost jednotlivých koeficientů se nezmění, mění se však frekvence, kterým jednotlivé koeficienty odpovídají. Při roztažení (expanzi) signálu v časové doméně dochází ke stlačení spektra do nižších kmitočtů, při stlačení (kompresi) signálu v časové doméně poté naopak dochází k roztažení spektra do vyšších kmitočtů. Poměrně intuitivní představou tohoto jevu je změna otáček na gramofonu – při přepnutí na nižší otáčky se zvukový signál přehrává pomaleji a v nižších frekvencích. [2]

1.1.6 Převzorkování signálu pomocí různých typů interpolací

Interpolace je metoda, kterou lze použít v případě, že je potřeba získat hodnotu v bodě, pro který daná hodnota není známa. Tento název pochází ze spojení latinských slov inter (mezi) a polire (leštit). [6] Pro nalezení těchto hodnot vycházíme z bodů, jejichž hodnoty jsou naopak známé. Pomocí různých numerických metod lze tuto neznámou hodnotu bodu určit. Toho je možné v 1D prostoru dosáhnout proložením známých bodů interpolační křivkou, ve 2D prostoru interpolační plochou atd. V rámci této diplomové práce se však pohybujeme zejména v 1D prostoru. Interpolační křivka může být také nazývána konvolučním jádrem. [7] Opačná metoda k interpolaci, tedy reprezentace stejného signálu pomocí menšího množství vzorků, se nazývá decimace. Interpolačními metodami a jejich názornou reprezentací se zabývá čtvrtá zadaná aplikace.

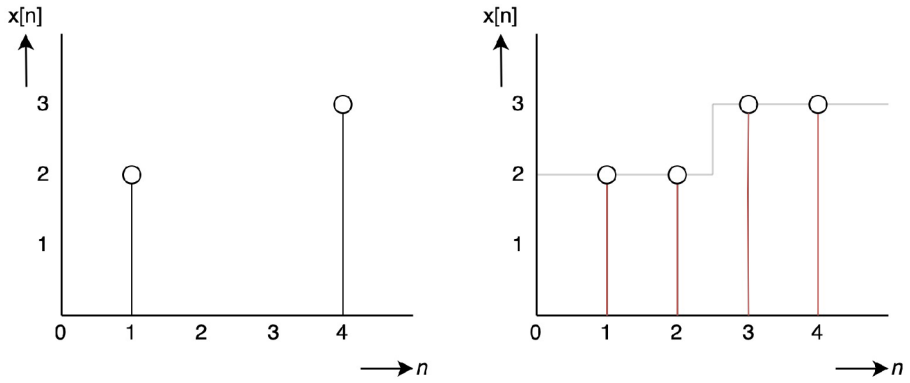
Metoda nejbližší soused



Obr. 1.10: Konvoluční jádro interpolace metodou nejbližšího souseda, vygenerováno v programu Matlab.

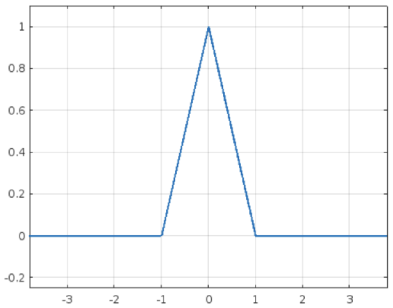
Při metodě nejbližšího souseda je v mezeře mezi dvěma body doplněna hodnota nejbližšího bodu (tzv. souseda). Jde tedy o doplnění hodnoty v neznámém bodě A o hodnotu jiného bodu B, jenž leží oproti ostatním bodům se známou hodnotou v nejkratší vzdálenosti právě od bodu A. Ačkoliv je metoda výpočetně velmi jednoduchá a efektivní, její výsledky mohou být velice nepřesné a zavádějící, neboť

nedochází k jakkoliv přirozeným přechodům mezi hodnotami, ale pouze k ostrým skokům.



Obr. 1.11: Signál interpolovaný metodou nejbližší soused – vlevo se nachází původní signál, vpravo jsou červeně označeny interpolované prvky, šedá reprezentuje průběh trendu.

Linerární interpolace

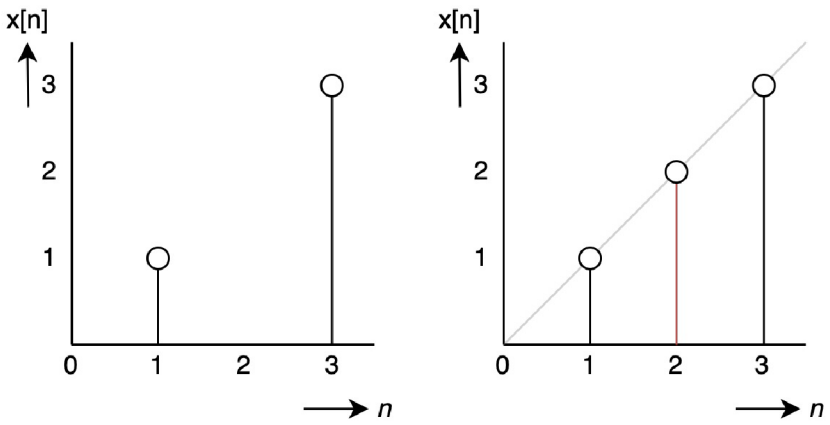


Obr. 1.12: Konvoluční jádro lineární interpolace, vygenerováno v programu Matlab

Lineární interpolace umožňuje nahradit průběh funkce mezi dvěma body úsečkou. Spojeny jsou vždy dva nejbližší body $x[n]$ a $x[n+1]$, kde $n = 1, 2, 3, \ldots$ Pro zjištění hodnoty neznámého bodu označeného s, kde $n < s < n + 1$, lze využít následující rovnici:

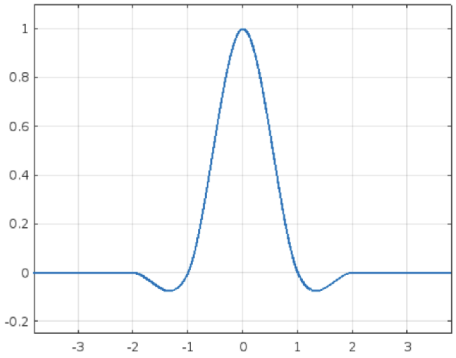
$$x(s) = x[n](s - n) + x[n + 1](n + 1 - s). \tag{1.26}$$

Pro vícerozměrný prostor nazýváme interpolaci také bilineární a trilineární. [8]



Obr. 1.13: Lineárně interpolovaný signál – šedá přímka reprezentuje průběh trendu, červeně je označen vzorek vzniklý interpolací.

Kubická interpolace



Obr. 1.14: Konvoluční jádro kubické interpolace, vygenerováno v programu Matlab.

Základní myšlenkou kubické interpolace je proložení každé mezery mezi dvěma sousedními body kubickým polynomem. V mnoha aplikacích je kubická interpolace dobrým kompromisem mezi výpočetní náročností a uspokojivými výsledky. V této metodě jsou dva sousední body spojeny pomocí polynomu, jehož stupeň je $m = 3$. Polynom (mnohočlen) třetího řádu má tvar [9]:

$$f(x) = a + bx + cx^2 + dx^3. \tag{1.27}$$

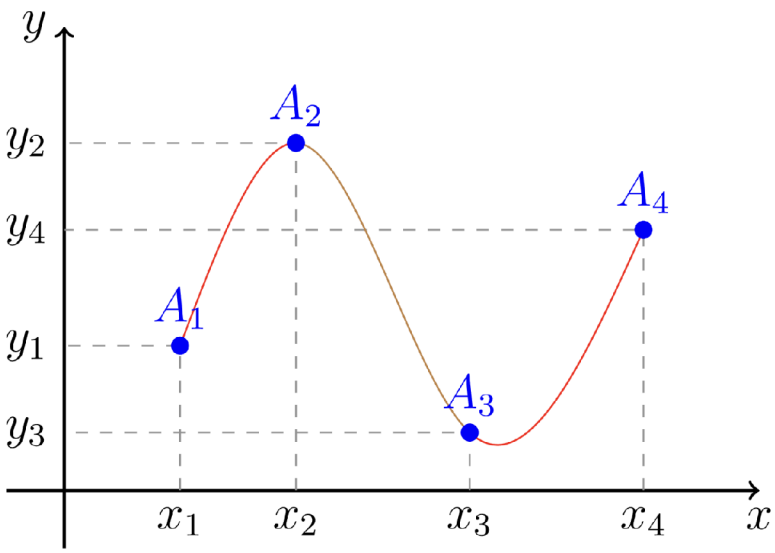
Funkci popisující interpolaci v celém rozsahu lze zapsat v následujícím tvaru [10]:

$$S(x) = \begin{cases} C_1(x), x_0 \leq x \leq x_1, \\ C_i(x), x_{i-1} \leq x \leq x_i, \\ C_n(x), x_{n-1} \leq x \leq x_n, \end{cases} \tag{1.28}$$

kde $i = 1, 2, \ldots, n$ označuje aktuální interval, n je celkový počet intervalů a pro C_i platí:

$$C_i(x) = a_i + b_i x + c_i x^2 + d_i x^3. \tag{1.29}$$

Hledaná výsledná funkce je tedy po částech polynomiální, kde každý interval mezi aktuálním bodem $A_i = [x_i; y_i]$ a následujícím bodem $A_{i+1} = [x_{i+1}; y_{i+1}]$ je právě jednou částí. Máme-li tedy deset bodů, mezi kterými chceme interpolovat, využijeme devět kubických polynomů místo jednoho polynomu desátého stupně. [11]



Obr. 1.15: Kubická interpolace, převzato z [6].

Pro každý interval (každé i) je nutné vypočítat koeficienty a_i, b_i, c_i a d_i, celkově se tedy jedná o $4n$ koeficientů. Na tyto funkce je kladena podmínka, že musí procházet body, které ji ohraničují. Po splnění této podmínky je funkce spojitá. [6]

$$\begin{aligned} C_i(x_i) &= y_i, \\ C_i(x_{i-1}) &= y_{i-1}. \end{aligned} \tag{1.30}$$

To je možné také interpretovat jako:

$$\begin{aligned} a_i + b_i x_{i-1} + c_i x_{i-1}^2 + c_i x_{i-1}^3 &= y_{i-1}, \\ a_i + b_i x_i + c_i x_i^2 + d_i x_i^3 &= y_i. \end{aligned}$$

Pro polynom stupně $m = 1$ je tento popis dostatečný – přímka je jasně dána a vzniká lineární interpolace – pro vyšší stupně je však potřeba doplnit další podmínky. V rámci kubické interpolace je tedy požadavkem na spoje (body) mezi funkcemi rovnost i mezi jejich derivacemi. V počítačové grafice se s těmito křivkami můžeme setkat pod označením splajn nebo spline. Rovnost první derivace funkce znamená, že směrnice grafů budou v daném bodě shodné. Druhá derivace zase zaručí, že v oblasti okolo navazujícího bodu budou tyto funkce stejně „vypouklé“.

$$\begin{aligned} C_i'(x_i) &= C_{i+1}'(x_i), \\ C_i''(x_i) &= C_{i+1}''(x_i). \end{aligned} \tag{1.31}$$

Po rozepsání těchto rovnic získáváme tvary:

$$\begin{aligned} b_i + 2c_i x_i + 3d_i x_i^2 &= b_{i+1} + 2c_{i+1} x_i + 3d_{i+1} x_i^2, \\ 2c_i + 6d_i x_i &= 2c_{i+1} + 6d_{i+1} x_i. \end{aligned}$$

V tuto chvíli je celková interpolace popsána pomocí $4n-2$ rovnic. Na krajích výsledné funkce je potřeba dodat ještě dvě podmínky (získat dvě chybějící rovnice), kdy může být zvolena např. možnost nulových druhých derivací – vznikají tzv. přirozené spliny.

$$C_1''(x_0) = C_n''(x_n) = 0. \tag{1.32}$$

Tato podmínka však umožňuje vytvořit pouze jeden konkrétní průběh kubické interpolace. Určit hodnotu derivací v bodech tak, aby byly nenulové, umožní v mnoha případech vytvořit takový průběh, který dané situaci lépe odpovídá, nebo je vhodnější pro dané využití. Výpočet interpolace mezi dvěma body je možné realizovat následovně – hledané polynomy $C_i(x)$ jsou spojnicí mezi dvěma sousedními body danými souřadnicemi $[x_0; y_0]$ až $[x_n; y_n]$. Tangenty, určující sklon okolo dvou sousedních bodů, označme k_1 a k_2. [12] Platí tedy, že:

$$\begin{aligned} C_i(x_1) &= y_1, \\ C_i(x_2) &= y_2, \\ C_i'(x_1) &= k_1, \\ C_i'(x_2) &= k_2. \end{aligned} \tag{1.33}$$

Na základě těchto podmínek získáváme čtyři rovnice o čtyřech neznámých:

$$\begin{aligned} a_i + b_i x_1 + c_i x_1^2 + d_i x_1^3 &= y_1, \\ a_i + b_i x_2 + c_i x_2^2 + d_i x_2^3 &= y_2, \\ b_i + 2c_i x_1 + 3d_i x_1^2 &= k_1, \\ b_i + 2c_i x_2 + 3d_i x_2^2 &= k_2. \end{aligned}$$

Pro zjednodušení výpočtu uvažujme případ na prvním intervalu, tedy:

$$\begin{aligned} x_1 &= 0, \\ x_2 &= 1. \end{aligned} \tag{1.34}$$

Soustava rovnic se zjednoduší na:

$$\begin{aligned} y_1 &= a_i, \\ y_2 &= a_i + b_i + c_i + d_i, \\ k_1 &= b_i, \\ k_2 &= b_i + 2c_i + 3d_i. \end{aligned}$$

Vyřešením soustavy rovnic získáme následující hodnoty koeficientů:

$$\begin{aligned} a_i &= y_1, \\ b_i &= k_1, \\ c_i &= 3(y_2 - y_1) - 2k_1 - k_2, \\ d_i &= k_1 + k_2 - 2(y_2 - y_1). \end{aligned} \tag{1.35}$$

Finální tvar je tedy:

$$C_i(x) = y_1 + k_1 x + (3y_2 - 3y_1 - 2k_1 - k_2)x^2 + (k_1 + k_2 - 2y_2 + 2y_1)x^3. \tag{1.36}$$

Tento tvar popisující kubický polynom v jednom intervalu není velmi praktický pro výpočty v samotné aplikaci. Zavedeme tedy následující tvar:

$$Q(x) = \Big(1 - t(x)\Big)y_1 + t(x)y_2 + t(x)\Big(1 - t(x)\Big)\Big((1 - t(x))u + t(x)v\Big), \tag{1.37}$$

kde:

$$\begin{aligned} t(x) &= \frac{x - x_1}{x_2 - x_1}, \\ u &= k_1(x_2 - x_1) - (y_2 - y_1), \\ v &= -k_2(x_2 - x_1) + (y_2 - y_1). \end{aligned} \tag{1.38}$$

Abychom dokázali, že je tento tvar $Q(x)$ ekvivalentní $C(x)$ při uvažování podmínek definovaných v 1.34, můžeme zjistit, zdali se každý člen polynomu $Q(x)$ (pro každou mocninu x) rovná, tedy jestli se shodují s koeficienty vypočítanými v 1.35:

$$\begin{aligned} &0 : y_1, \\ &1 : -y_1 + y_2 + a = -y_1 + y_2 + k_1 - y_2 + y_1 = k_1, \\ &2 : -2a + b = 3(y_2 - y_1) - 2k_1 - k_2, \\ &3 : a - b = -2y_2 + 2y_1 + k_2 + k_1. \end{aligned}$$

Tvary jsou tedy ekvivalentní.

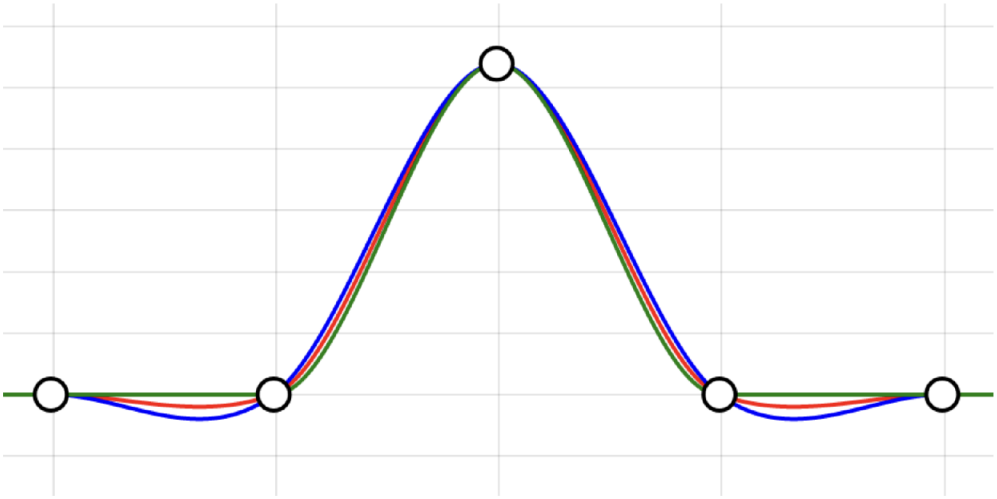
Metod získání hodnot k_1 a k_2 existuje vícero. Nejjednodušší je výše zmíněná metoda přirozeného kubického splinu, kdy za k_1 i k_2 je dosazena 0. Lze však také použít metodu konečných diferencí (konkrétně tříbodovou diferenci) [12], jež umožňuje nalézt tangentu pomocí rovnice:

$$k_i = \frac{1}{2}\Big(\frac{A_{i+1} - A_i}{x_{i+1} - x_i} + \frac{A_i - A_{i-1}}{x_i - x_{i-1}}\Big), \tag{1.39}$$

kde A_i označuje bod se souřadnicemi $[x_i; y_i]$. Tangenty získáváme v podstatě jako průměrný sklon mezi dvojicemi bodů $A_{i+1}A_i$ a A_iA_{i-1}. Rozdíl mezi body je vypočítán vektorově.

Další metodou nalezení tangent je metoda kardinálního splinu (taktéž označovaného jako kanonický spline), která přijímá také parametr označený jako c, nazývaný napětí (tension). Při dosazení hodnoty 1 je výsledná tangenta nulová a vzniká přirozený spline. Speciální případ, kdy je tato hodnota rovna 0,5 se nazývá Catmull–Rom spline. [13]

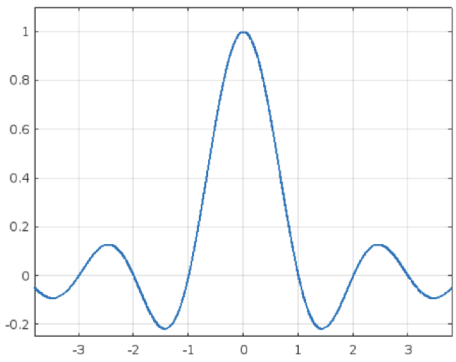
$$k_i = (1 - c)\frac{A_{i+1} - A_{i-1}}{x_{i+1} - x_{i-1}}. \tag{1.40}$$



Obr. 1.16: Porovnání jader metod výpočtu tangent kubických interpolací: modrá – konečné diference, červená – Catmull–Rom spline, zelená – přirozená.

Výše zmíněné metody používají okno o rozsahu tří bodů, což však není možné aplikovat na počáteční a koncový bod – je tedy potřeba zvolit ještě tyto okrajové podmínky. Intuitivně se nabízí např. vypočítat tangentu pouze mezi prvním a druhým (případně posledním a předposledním) bodem.

Metoda interpolace pomocí funkce sinc



Obr. 1.17: Konvoluční jádro interpolace pomocí funkce sinc, vygenerováno v programu Matlab.

Interpolace funkcí sinc je založena na využití Whittaker–Shannonova interpolačního vzorce. V praxi se s touto interpolací setkáváme při rekonstrukci analogového signálu z digitálních vzorků. Této rekonstrukce je možné dosáhnout proložením každého bodu diskrétního signálu právě funkcí sinc. [14]

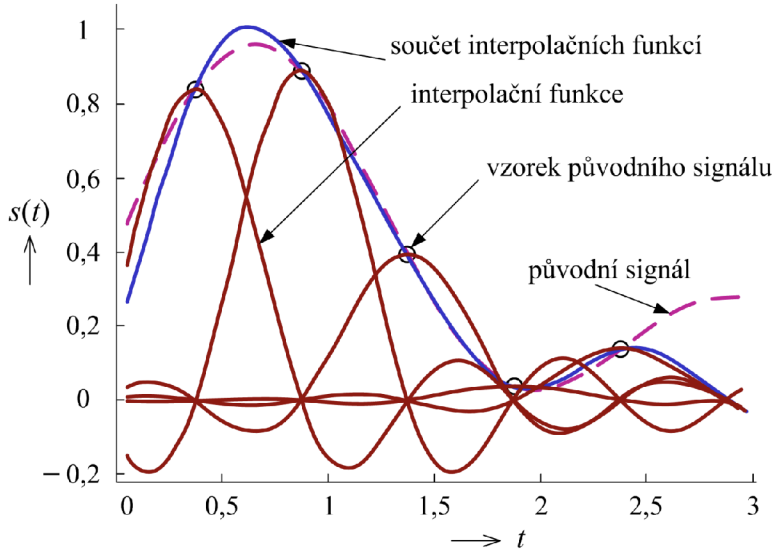
Funkce sinus cardinalis je definována takto:

$$\mathrm{sinc}(t) = \begin{cases} \frac{\sin t}{t}, & \text{pro } t \neq 0 \\ 1, & \text{pro } t = 0 \end{cases} \tag{1.41}$$

Pro interpolaci je poté využit vztah:

$$s(t) = \sum_{n=-\infty}^{\infty} s(n)\mathrm{sinc}\left(\frac{t - nT}{T}\right). \tag{1.42}$$

Při proložení jednotlivých vzorků touto funkcí je patrné, že funkce sinc má nenulové hodnoty v daném vzorku a v prostoru mezi vzorky. V dalších vzorcích je však hodnota nulová. Sečtením všech proložených interpolačních funkcí sinc je získána aproximace výsledného signálu. [14]



Obr. 1.18: Interpolace funkcí sinus cardinalis, převzato z [2].

1.2 Zvolené informační technologie

1.2.1 Jazyk JavaScript

JavaScript je velmi populární programovací jazyk, nejčastěji využívaný v oblasti webového vývoje. V době psaní této práce je podle indexu PYPL třetím nejpoužívanějším jazykem současnosti. [15] Prvotní verze JavaScriptu byla napsána v roce 1995 pro prohlížeč Netscape 2 a jeho autorem je Brendan Eich. JavaScript je spolu s WebAssembly podporován všemi současnými populárními prohlížeči. Základní vlastnosti JavaScriptu jsou, že je:

- Multiparadigmatový: k programování v tomto jazyce lze použít vícero přístupů či stylů psaní kódu. V samotné aplikaci je převážně využit imperativní přístup v kombinaci s funkcionálními metodami.
- Vysokoúrovňový: obsahuje silnou úroveň abstrakce nad vnitřními procesy počítače – nedochází např. k manuální alokaci paměti, manipulaci se zásobníkem volání či registry. Pro správu paměti je používán garbage collector. [16]
- Jednovláknový: Aplikace napsaná v jazyce JavaScript běží pouze na jednom vlákně. Pro práci s asynchronními operacemi je využívána tzv. „non-blocking event-loop“, volně přeloženo jako „neblokující smyčka událostí“, která je – velmi zjednodušeně řečeno – schopna vykonávat operace na pozadí bez nutnosti blokování hlavního chodu programu.
- Dynamicky typovaný: Jedná se o jazyk s dynamickými datovými typy, lze tedy datový typ proměnných za běhu měnit a upravovat. Tato vlastnost může často vést k problémům – jazyk totiž takto povoluje programátorovi napsat kód, který nepodléhá dobrým programátorským návykům a je náchylný na chyby. JavaScript je však extrémně populárním jazykem a v oblasti webových technologií se s ním lze setkat prakticky všude. Problém s dynamickým typováním lze řešit oblíbenou alternativou, která se nazývá TypeScript.

1.2.2 Vybrané kapitoly z jazyka JavaScript

V kódu diplomové práce je použito několik konstrukcí, které je vhodné popsat v textu, aby bylo možné mu plně porozumět. Cílem kapitoly je osvětlit čtenářům, kteří jsou jinak obeznámeni se základy programování, jak fungují některé častěji se vyskytující konstrukce v kódu aplikace, které nejsou všeobecně rozšířené pro jiné populární programovací jazyky. K jednotlivým konstrukcím jsou uvedeny příklady, po nichž následuje příkaz `console.log`, sloužící pro výpis do konzole.

Anonymní lambda funkce

V jazyce JavaScript se lze velmi hojně setkávat s touto konstrukcí nepojmenovaných (anonymních) funkcí, jež jsou zadány pomocí šipkové (arrow) notace. Jedná se o syntaktické zjednodušení umožňující jednodušší a přehlednější práci s funkcemi.

```
const arr = []
const jazyky = ['JavaScript', 'TypeScript', 'Haskell']
jazyky.forEach((x) => arr.push(x.toUpperCase()))
console.log(arr)
// ['JAVASCRIPT', 'TYPESCRIPT', 'HASKELL']
```

Syntax spread

Spread syntax umožňuje „rozbalit“ objekt nebo pole a předat jeho prvky jako argumenty funkce pomocí syntaxe tří teček. Také lze pomocí něj spojovat objekty a pole.

```
const obj1 = {a: 1}
const obj2 = {b: 2}
const obj3 = {...obj1, ...obj2}
console.log(obj3)
// {a: 1, b: 2}
```

Metoda map

Tuto metodu a další dvě následující (`filter` a `reduce`) lze volat pomocí tečkované notace nad libovolným polem. Metodu je možné volat na základě prototypové dědičnosti, konkrétně z objektu `Array.prototype`. Výhodou užívání těchto metod je, že nedochází k mutaci původního pole. [17]

Voláním metody `map` nad polem získáme nové pole tak, že předáme jako argument funkci, jež se má vykonat nad každým prvkem pole původního. V mnoha případech je `map` vhodnou alternativou k cyklu `for`, která zabere méně řádků kódu.

```
const arr = [1,2,3]
const newArr = arr.map(a => a + 1)
console.log(newArr)
// [2,3,4]
```

Metoda filter

Metoda `filter` umožňuje z původního pole smazat ty prvky, které nesplní podmínku, jež je předána jako výraz v argumentu.

```
const arr = [1,2,3]
const newArr = arr.filter(a => a > 1)
console.log(newArr)
// [2,3]
```

Metoda reduce

Metoda `reduce` umožňuje nad každým prvkem pole volat redukční funkci (reducer). Během redukce je k dispozici akumulátor a aktuální prvek, kdy návratová hodnota předchozího výpočtu je předána jako akumulátor pro následující výpočet.

```
const arr = [1,2,3]
const newArr = arr.reduce((akumulator, aktualniPrvek) =>
akumulator + aktualniPrvek)
console.log(newArr)
// 6
```

Do konstrukce s `reduce` je také možné jako druhý parametr zadat výchozí hodnotu, jež je přítomna v akumulátoru během výpočtu prvního prvku.

```
const arr = [4,5,6]
const vychoziHodnota = [1,2,3]
const newArr = arr.reduce((akumulator, aktualniPrvek) =>
[...akumulator, aktualniPrvek], vychoziHodnota)
console.log(newArr)
// [1,2,3,4,5,6]
```

1.2.3 Jazyk TypeScript

Jazyk TypeScript je rozšířením jazyka JavaScript o statické typování a další nástroje. TypeScript byl vytvořen firmou Microsoft, která i zajišťuje jeho správu. Kód napsaný v TypeScriptu se kompiluje do jazyka JavaScript a tento kód je tedy možné spustit kdekoliv, kde je JavaScript podporován. Kód je již při psaní kontrolován pomocí statické typové analýzy a chytrý editor programátorovi napovídá pomocí inteligentního dokončování kódu. Příkladem může být staticky typovaný objekt, k jehož klíčům můžeme přistoupit pomocí tečkované notace (dot notation) nebo výpis daného typu při najetí na definici či instanci kurzorem. [18]

Výhodu využití striktnějších typů lze demonstrovat na příkladu – zavedeme funkci `nastavSvetloSemaforu`. V jazyce JavaScript by obecně bylo možné volat tuto funkci s argumentem kteréhokoliv datového typu, což by způsobilo chybu při běhu aplikace. V některých jazycích se statickými typy by bylo možné omezit tento typ na libovolný řetězec. To však nemusí být dostačující. Optimální je omezit argument na datový typ `'zelena' | 'cervena' | 'oranzova'`. TypeScript by volání funkce s jiným řetězcem odhalil již při kompilaci a tím odchytil potenciální nepříjemnou chybu ještě dříve, než bude program spuštěn.

```
type BarvaSemaforu = 'zelena' | 'cervena' | 'oranzova'
const nastavSvetloSemaforu = (barva: BarvaSemaforu) =>
{ ... }
nastavSvetloSemaforu('ruzova')
// nastane chyba při kompilaci
```

1.2.4 Vybrané kapitoly z jazyka TypeScript

Obdobně jako v kapitole 1.2.2 se ve výsledném kódu objevuje několik opakujících se konstrukcí. V příkladech je využíván symbol `|`, jenž se chová jako logické nebo.

Deklarace typů

Jazyk TypeScript umožňuje deklaraci vlastních typů pomocí klíčového slova `type`.

```
type VstupniTvar = {
  A: number
  typ: string
  signal: number[]
}
```

Nyní je možné deklarovat objekt, který je typu `VstupniTvar`.

```
const signal: VstupniTvar = {
  A: 3
  typ: 'sinus'
  signal: [1,2,3]
}
```

V případě pokusu o uložení hodnoty nesprávného datového typu ke kterémukoliv klíči tohoto objektu by došlo k chybě již při kompilaci.

Klíčová slova keyof a typeof

Tato klíčová slova slouží k odvození (type inference) datových typů z proměnných. Klíčové slovo `typeof` vrací typ proměnné jako řetězec a `keyof` vytváří typ z klíčů daného objektu.

```
const a = 'retezec'
console.log(typeof a)
// 'string'

const obj = {a: 1, b: 2}
type NovyTyp = keyof obj
// NovyTyp je nyní typu 'a' | 'b'
```

Syntax as const

Pomocí syntaxe `as const` je možné odvodit konkrétnější typ, než by byl standardně odvozen.

```
const a = 'slovo'
console.log(typeof a)
// 'string'
const b = 'slovo' as const
console.log(typeof b)
// 'slovo'
```

Příklad vyskytující se ve zdrojovém kódu aplikace umožňuje ze seznamu možných signálů vytvořit datový typ a tím zamezit volání funkce, jež zpracovává daný signál s nesmyslnou hodnotou:

```
export const Rovnice = [
  'sinus',
  'obdelnik',
  'trojuhelnik',
  'pila'
] as const;


type TypRovnice = typeof Rovnice[number];
// typem je nyní: "sinus" | "obdelnik" | "trojuhelnik" | "pila"
```

Nyní je možné při deklaraci stavu aktuálního zobrazeného signálu omezit typ tak, aby nebylo možné přiřadit řetězec jiný, než které jsou předepsané v poli `Rovnice`.

```
const [rovnice, nastavRovnici] = useState<TypRovnice>('sinus')
```

Touto syntaxí pro deklaraci stavu a jeho manipulaci se zabývá kapitola 2.3.

1.2.5 Jazyk HTML

HTML (hypertext markup language) je značkovací jazyk standardně využívaný v prohlížečích pro vytvoření struktury webových stránek či aplikací. Stránky jsou provázány pomocí odkazů (hyperlink). Jazyk kromě definice struktury a obsahu stránky umožňuje i omezenou práci s formátováním textu. Základním stavebním kamenem tohoto jazyka jsou značky (tag), kdy obsah je předán mezi otevírající a uzavírající značku pomocí syntaxe: `<značka>obsah</značka>`. Tyto dvě značky lze také spojit v jednu, pokud není nutné vložit mezi značky obsah: `<značka />`. Značky lze

upravovat pomocí atributů, jež se zadávají ve stylu `<značka atribut='hodnota'>`. HTML soubor má svoji standardizovanou strukturu – na první řádek se vkládá informace o typu dokumentu pomocí `<!DOCTYPE html>` a další kód je obalen značkami `<html></html>`. Obsah těchto značek je dále rozdělen na hlavičku (head) a samotné tělo souboru (body). [19]

Výpis 1.1: Příklad struktury základního HTML souboru.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta
      name="viewport"
      content="width=device-width,initial-scale=1"
    >
    <title>Nazev stranky</title>
    <link rel="stylesheet" href="style.css">
  </head>
  <body>
    <h1>Moje stranka</h1>
    <p>Lorem ipsum</p>
    <script src="index.js"></script>
  </body>
</html>
```

1.2.6 CSS

Jedná se o jazyk využívaný ve webových aplikacích a stránkách pro úpravu jejich vzhledu. Oproti HTML disponuje tento jazyk velkou škálou možností, jak vzhled dané stránky upravovat. Jazyk funguje na základě deklarací pravidel, jež jsou potom pro jednotlivé HTML značky uplatňovány. Pomocí CSS lze měnit např. jak chování všech tlačítek `<button>` zároveň, tak i konkrétní tlačítko, kterému je přiřazena třída pomocí atributu `class`. Definice pravidel pro takovou třídu lze poznat pomocí tečky, která se před jméno třídy v souboru s příponou `.css` umisťuje. [20]
Změna vzhledu všech tlačítek:

```
button {
  font-size: 10px;
  color: #000000
}
```

Změna vzhledu konkrétního tlačítka:

```
.moje-tlacitko {
  font-size: 12px;
  color: #ffffff
}
```

Kdy je tlačítku přiřazena třída:

```
<button class='moje-tlacitko'>
  Klikni!
</button>
```

1.2.7 React

React je v současné době nejpoužívanější aplikační rámec (framework) pro tvorbu tzv. front–endu ve webových aplikacích. Laicky řečeno je front–end ta část aplikace, se kterou interaguje uživatel, tedy klientská část aplikace běžící v prohlížeči. Původním autorem knihovny React je Jordan Walke, v současné době jej spravuje samostatný team společnosti Meta.

React umožňuje tvořit aplikace s filozofií kompozice menších komponent, jež se skládají dohromady pro vytvoření větších celků. Pro React je typický deklarativní způsob psaní uživatelského rozhraní a také agnostický přístup ke zbytku využitých technologií – React je možné pohodlně kombinovat s kteroukoliv další knihovnou pro jazyk JavaScript a dokonce i TypeScript, který plně podporuje. Součástí knihovny je systém pro vytváření, udržování a manipulaci se stavem aplikace. Využíváno je syntaxe JSX, umožňující obstarávat logiku chování aplikace přímo s vizuálními prvky. Tam, kde by běžnou praxí bylo pro webového programátora mít tři soubory (s příponami `.html`, `.css` a `.js`) stačí pouze jeden soubor využívající právě této syntaxe. [21]

Výpis 1.2: Ukázka syntaxe JSX.

```
{interpolace.kubicka && (
  <div className="btn-div">
    {seznamKubickychInterpolaci.map(
      ([nazevInterpolace, jeZapnuta]) => (
      <button
        style={{
          backgroundColor: jeZapnuta ? '#B9B9B9' : '#F2F2F2',
        }}
        onClick={() =>
          nastavKubickeInterpolace((soucasnyStav) => ({
            ...soucasnyStav,
            [nazevInterpolace]: !jeZapnuta,
          }))
        }
      >
        {mapaKubickaInterpolaceNazev[nazevInterpolace]}
      </button>
    ))}
  </div>
)}
```

Kód je možné bez zabíhání do detailů číst následovně – pokud je zapnutá kubická interpolace, vykresli pro každý prvek pole `seznamKubickychInterpolaci` tlačítko. Pokud je tlačítko aktivní, změň jeho barvu. Po kliknutí na tlačítko změň celkový stav tak, aby se změnila právě pouze hodnota, kterou reprezentuje konkrétní tlačítko – aktivní tlačítko deaktivuj a naopak. Nakonec pomocí `mapaKubickaInterpolaceNazev` umísti do tlačítka vhodný text.

Na tomto příkladu dochází k využití typické vlastnosti syntaxe JSX – v jednom úryvku kódu měníme jak celkový model stránky pomocí HTML, tak jeho vzhled pomocí logiky specifikované kódem v jazyce JavaScript.

1.2.8 Chart.js

Jedná se o knihovnu umožňující pohodlnou práci s grafy pro jazyk JavaScript. Nachází se v ní mnoho šablon pro práci se standardními druhy grafů (sloupcový, spojnicový, kruhový,...). Okolo knihovny existuje také open source komunita využívající rozšiřitelnosti knihovny pomocí plugin systému, přinášející různá vylepšení a nové funkcionality. Kromě základních očekávaných funkcí nabízí `Chart.js` také např. animace, hybridní grafy kombinující více druhů grafů do jednoho, nebo také interaktivní prvky umožňující manipulovat s grafem pomocí myši atd. Knihovna je rozšiřitelná

o sadu datových typů pro podporu TypeScriptu a dokonce i o předpřipravené komponenty pro React v podobě knihovny `react-chartjs-2`. Kromě Reactu lze knihovnu použít i pro další JavaScriptové aplikační rámce (framework), jako Angular, Vue, Svelte atd. [22]

2 Programové řešení práce

2.1 Založení nové aplikace

Před vytvořením nové aplikace je třeba stáhnout balíčkovací manažer npm a také běhové (run-time) prostředí Node.js. Novou React aplikaci podporující TypeScript lze vytvořit příkazem:

```
npx create-react-app my-app --template typescript
```

Tento příkaz vytvoří základní React aplikaci, která se skládá z několika souborů a složky `public`, jež obsahuje soubor `index.html`, pomocí kterého je aplikace spuštěna prohlížečem, dále `favicon.ico`, reprezentující ikonu aplikace v záložce prohlížeče a několik dalších souborů. Mimo složku `public` je vytvořen také soubor `package.json`, který obsahuje např. seznam závislostí na externí knihovny, pomocí kterého jsou tyto knihovny instalovány, a také skripty umožňující aplikaci spustit při lokálním vývoji, spuštění testů, vystavění aplikace do statických souborů atd.

Na základě osobní preference byly pro vývoj zvoleny ještě developerské pomocné knihovny, které jsou v textu uvedeny pro úplnost a obhájení jejich existence ve zdrojovém kódu. Tyto knihovny jsou:

- prettier – upozornění programátora na problémy a chyby ve stávajícím kódu, konfigurované v souboru `.eslintrc.json`
- eslint – zejména automatizované formátování kódu konfigurované v souboru `.prettierrc`

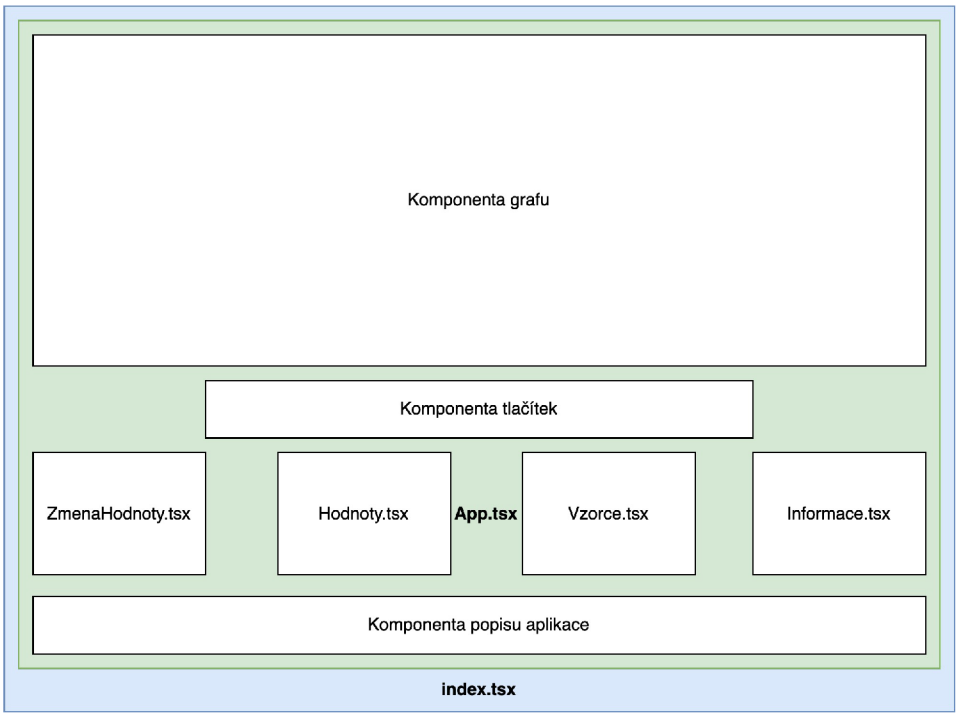
Posledním konfiguračním souborem přítomným v repozitáři je `tsconfig.json` umožňující konfiguraci TypeScriptového kompilátoru.

2.2 Zobrazení střední a efektivní hodnoty

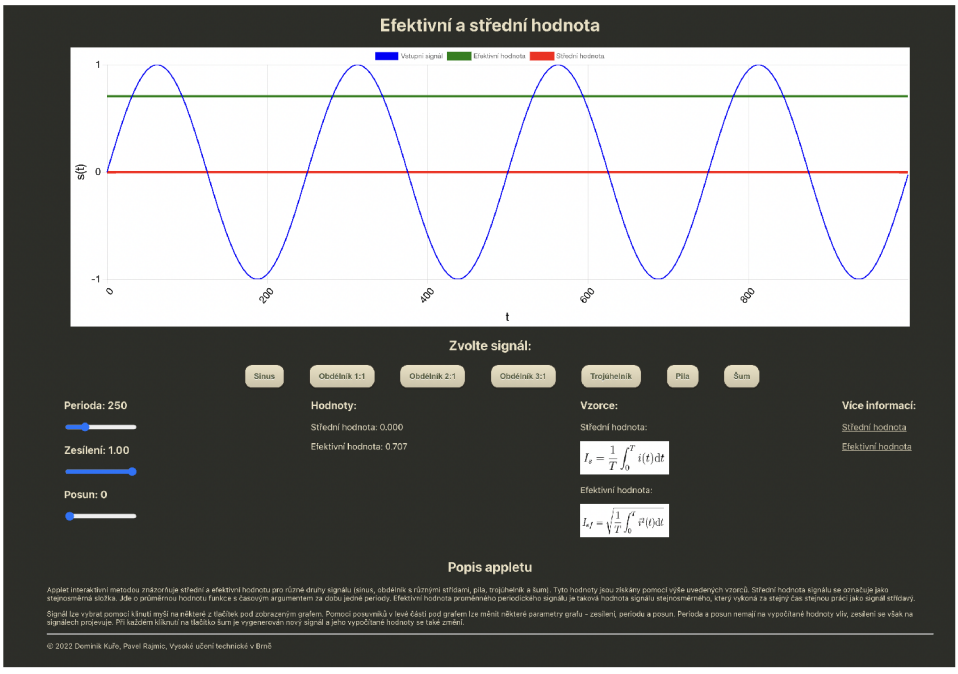
Následující stromová struktura je reprezentací části aplikace obsahující implementaci, tudíž se v ní nenacházejí konfigurační soubory, testovací soubory atd. Struktura bude velmi podobná i pro ostatní aplikace, odlišnosti se budou nacházet ve složkách `knihovna` a `komponenty`, neboť tyto jsou specifické vždy pro danou aplikaci.

/ ..... src
- knihovna
  - rovnice.ts
  - utility.ts
  - utilityGraf.ts
  - vypocetHodnot.ts
- komponenty
  - Hodnoty.tsx
  - Informace.tsx
  - Vzorce.tsx
  - ZmenaHodnoty.tsx
- obrazky
  - efektivni.png
  - stredni.png
- App.css
- App.tsx
- index.css
- index.tsx

V souborech `index.css` a `index.tsx` lze upravovat stránku na nejvyšší úrovni. Soubor `index.css` pro všechny aplikace vytvořené v rámci diplomové práce pouze definuje barvu pozadí, velikost stránky a font. Soubor `index.tsx` pouze renderuje aplikaci. Soubor `App.tsx` obsahuje hlavní komponentu reprezentující celou aplikaci. Tato komponenta je složena z menších komponent (např. `Hodnoty.tsx`, `Informace.tsx` atd.), které jsou uloženy v samostatné složce, případně nadefinovány přímo v `App.tsx`. Lepší grafickou představu je možné získat porovnáním obrázku schématu 2.1 s GUI výsledné aplikace 2.6.



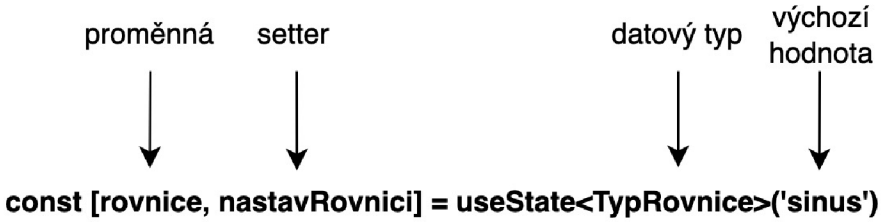
Obr. 2.1: Grafické schéma aplikace.



Obr. 2.2: GUI aplikace pro zobrazení střední a efektivní hodnoty.

2.2.1 Aplikace zobrazující střední a efektivní hodnotu signálu

Hlavní část aplikace – soubor `App.tsx` – může být rozdělen do několika částí. První část obsahuje importy komponent, funkcí, typů, objektů atd. z jiných částí aplikace. Následuje registrace pluginů pro knihovnu `Chart.js` a poté již hlavní komponenta aplikace. Pro práci se stavem se v knihovně React využívá tzv. `useState` hook, jehož syntax popisuje obrázek 2.3.



Obr. 2.3: Syntax useState.

Po definici stavových proměnných následuje definice funkcí, jež jsou spuštěny na základě změny vstupu uživatele (kliknutí na tlačítko, posuv posuvníku). Tyto funkce pomocí výše definovaných setterů (funkce sloužící k nastavení aktuálního stavu) upraví stav podle typu události a zadaných hodnot. Následující krátká sekce definuje graf jako takový, jenž obsahuje tři hodnotové kanály (vstupní signál, efektivní hodnota a střední hodnota) a také údaje o nastavení os. Graf je kromě tohoto místa konfigurován ještě v souboru `utilityGraf.ts`. Návratovou hodnotou celé komponenty je stromová struktura reprezentující zobrazení jednotlivých komponent pomocí syntaxe JSX.

Pro úplnost je třeba ještě uvést soubor `utlity.ts`, jenž obsahuje pomocné funkce, jako mapování hodnoty tlačítka na danou rovnici a výpočet náhodného čísla, které jsou dále využívány v jiných částech aplikace.

2.2.2 Komponenty

Aplikace obsahuje několik komponent, z nichž jsou čtyři uloženy do samostatných souborů. Tyto menší komponenty jsou v hlavní komponentě poskládány do většího celku. Samostatné komponenty jsou `Hodnoty`, `Informace`, `Vzorce` a `ZmenaHodnoty` a jsou v definovány ve stejnojmenných souborech s příponou `.tsx`.

Strukturu takové komponenty lze uvést na příkladu výpisu 2.1, v tomto případě jde o komponentu `Vzorce`.

Výpis 2.1: Ukázka komponenty Vzorce.tsx.

```
1  import efektivni from '../obrazky/efektivni.png'
2  import stredni from '../obrazky/stredni.png'
3
4  export default const Vzorce = () => {
5    return (
6      <div>
7        <h3>Vzorce:</h3>
8        <p>
9          <p>Střední hodnota:</p>
10         <img src={stredni} alt="stredni"
11         height="60px" width="160px" />
12         <p>Efektivní hodnota:</p>
13         <img src={efektivni} alt="efektivni"
14         height="60px" width="160px" />
15       </p>
16     </div>
17   )
18 }
```

Na řádku 1 a 2 dochází k importování obrázků z jiného místa v adresáři, následuje definice komponenty. Z kódu je patrné, že komponenta je funkce, jejíž návratová hodnota je JSX element. Komponenta musí vždy vracet pouze jeden takový element, který ale může uvnitř obsahovat libovolný počet dalších elementů. Značka `<h3>` označuje odstavec třetí úrovně, značka `<p>` odstavec textu, `<div>` označuje logickou část a značka `<img>` slouží k zobrazení obrázku. Tomuto obrázku je dále přiřazena velikost a klíč `alt`, sloužící k výpisu textu, pokud by nebyl obrázek nalezen. Funkce je při deklaraci označena spojením `export default`, které umožňuje komponentu importovat v jiné části aplikace s využitím syntaxe bez složených závorek.

Import bez využití default:

```
import { Vzorce } from './komponenty/Vzorce'
```

Import s využitím default:

```
import Vzorce from './komponenty/Vzorce'
```

Komponentám mohou být předány argumenty, v knihovně React označovány jako `props`. V aplikaci k tomu dochází např. v komponentě `ZmenaHodnoty`, která se v aplikaci volá třikrát, pokaždé s jinými argumenty (pro periodu, maximum a posun).

2.2.3 Rovnice vstupních signálů

Implementaci funkcí pro vykreslení vstupních signálů je možné nalézt v souboru `rovnice.tsx`. Pro jednotlivé signály jsou zadány hodnoty maxima, posunu a periody, kterými je možné upravovat vstupní signál.

Rovnice sinus

Pro úpravu sinusového signálu je v programu využita rovnice:

$$y(x) = A \sin\left(\frac{2\pi x}{T} + \varphi\right). \tag{2.1}$$

Posun p je v aplikaci zadáván v sekundách a jeho převod na fázový posun je realizován pomocí rovnice $\varphi = \frac{2\pi p}{T}$. V aplikaci je tedy výpočet implementován takto:

```
const sinusRovnice =
(arr: number[], A: number, T: number, p: number) =>
  arr.map((x) => Math.sin(
  (2 * Math.PI * x) / T + (2 * Math.PI * p) / T
  ) * A)
```

Rovnice trojúhelníka

K vytvoření trojúhelníkového signálu se zadaným maximem, posunem a periodou lze využít následující vztah:

$$y(x) = \frac{4A}{T}\left|\left(\left(x - \frac{T - 4p}{4}\right) \mod T\right) - \frac{T}{2}\right| - A, \tag{2.2}$$

kde A je maximum, p posun a T perioda. Implementace této funkce v jazyce TypeScript vypadá následovně:

```
const trojuhelnikRovnice =
(arr: number[], A: number, T: number, p: number) =>
  arr.map(
    (x) =>
      ((4 * A) / T) *
        Math.abs((((((x - (T - p * 4) / 4) % T) + T) % T) - T / 2) -
      A,
  )
```

Na první pohled může být matoucí rozdíl mezi matematickým zápisem a programovým řešením v dvojím použití operace modulo. Důvodem této změny je, že operátor

modulo v JavaScriptu (a mnoha dalších programovacích jazycích) počítá zbytek po dělení a tato operace se chová pro záporná čísla jinak, než matematická operace modulo. Definice operace modulo je následující:

$$(A \mod B) = A - \left\lfloor \frac{A}{B} \right\rfloor B, \tag{2.3}$$

kde zápis $\lfloor x \rfloor$ je funkce, která vrací nejbližší nižší celé číslo (tzv. floor function). Operátor modulo tedy vždy vrátí po výpočtu kladnou hodnotu:

$$-21 \mod 4 = 3.$$

Definice výpočtu zbytku, která však také bývá označována zkratkou „mod“, je následující:

$$(A \mod B) = A - B(\frac{A}{B}). \tag{2.4}$$

Při výpočtu zbytku tedy dostáváme jiný výsledek:

$$-21 \mod 4 = -1.$$

Tato druhá varianta reprezentuje chování operátoru mod v JavaScriptu a dalších jazycích, v rovnici pro trojúhelníkový signál je však uvažována první varianta. Převést operaci výpočtu zbytku `(A % B)` lze v JavaScriptu pomocí zápisu `(((A % B) + B) % B)`.

Rovnice šumu

Šum je v grafu vytvořen pomocí generátoru náhodného čísla v podobě funkce `Math.random()`, která vrací náhodné číslo mezi 0 a 1 s rovnoměrným rozdělením pravděpodobnosti. Tuto funkci lze využít ve větším celku pro výpočet náhodné hodnoty v určitém rozsahu:

```
const ziskejNahodneCeleCislo = (min: number, max: number) =>
    Math.floor(Math.random() * (max - min + 1)) + min
```

Rovnice pily

Pilový signál lze definovat následovně pro zvolenou amplitudu, periodu a posun:

$$y(x) = 2A\left(\frac{x+p}{T} - \left\lfloor \frac{1}{2} + \frac{x+p}{T} \right\rfloor\right). \tag{2.5}$$

Implementace funkce výpočtu pilového signálu tedy v programu vypadá takto:

```
const pilaRovnice =
(arr: number[], A: number, T: number, p: number) =>
    arr.map((x) =>
    2 * A * ((x + p) / T - Math.floor(1 / 2 + (x + p) / T)))
```

Rovnice obdélníka

Rovnice pro výpočet obdélníka narozdíl od ostatních signálů přijímá ještě informaci o střídě, označenou jako d podle anglického duty cycle. Tato hodnota označuje poměr času, ve kterém hodnota signálu nabývá maxima oproti minimu. Obdélníkový signál se střídou 3:1 tedy nabývá maximální hodnoty pro 75% doby trvání jedné periody a 25% této doby je hodnota minimální.

```
const obdelnikRovnice =
(d: number) => (arr: number[], A: number, T: number, p: number) =>
    arr.map((x) => (((((x + p + (T * d) / 2) % T) + T) % T)
    / T > d ? -A : A));
```

2.2.4 Rovnice pro výpočet středních a efektivních hodnot

V souboru `vypocetHodnot.tsx` se nachází funkce pro výpočet střední a efektivní hodnoty. Do funkcí `vypocitejEfektivniHodnotu` a `vypocitejStredniHodnotu` je na vstupu přijímám objekt obsahující maximum, označené jako A, typ signálu a zdrojový signál. Zdrojový signál je využit v rovnici pro výpočet šumu, kdy nelze výpočty hodnot zjednodušit a je potřeba jej počítat z diskrétních vzorků. Funkce pro výpočet střední a efektivní hodnoty šumu jsou zobrazeny ve výpisu 2.5.

Střední hodnota signálu s průběhem sinus, obdélník se střídou 1:1, trojúhelník a pila je nulová. Pro obdélník se střídou 2:1 je střední hodnota $\frac{A}{3}$, pro obdélník se střídou 3:1 potom $\frac{A}{2}$, kde A je maximum funkce.

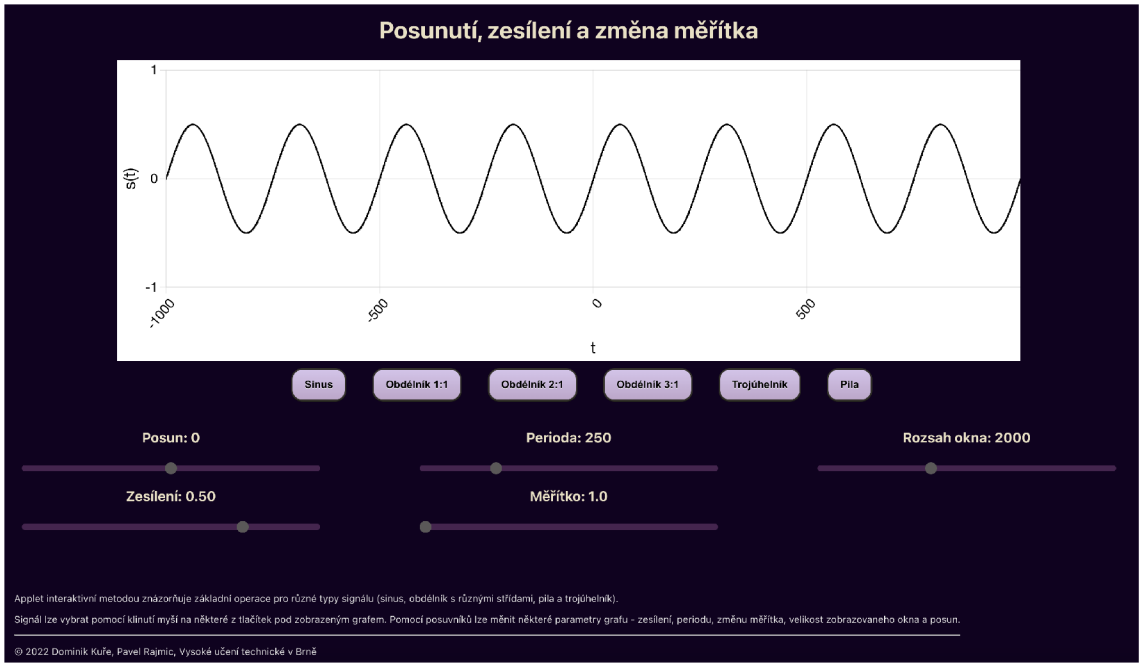
Efektivní hodnota obdélníkových signálů je rovna maximu. Pro sinusový signál je tato hodnota $\frac{A}{\sqrt{2}}$, pro pilový a trojúhelníkový signál $\frac{A}{\sqrt{3}}$.

Výpis 2.2: Funkce pro výpočet efektivní a střední hodnoty šumového signálu.

```
1   const integrate = (signal: number[]) =>
2   signal.reduce((acc, x) => acc + x, 0)
3
4   export const vypocitejEfektivniHodnotuSignalu =
5   (signal: number[]) =>
6   {
7     const signalUmocneny = signal.map((i) => i * i)
8     return Math.sqrt(
9       (1 / signal.length) * integrate(signalUmocneny)
10    )
11  }
12
13  export const vypocitejStredniHodnotuSignalu =
14  (signal: number[]) =>
15    (1 / signal.length) * integrate(signal)
```

Funkce `integrate` reprezentuje výpočet integrálu a je volána funkcemi `vypocitejEfektivniHodnotuSignalu` a `vypocitejStredniHodnotuSignalu`, které jsou pouze přepisem vzorců pro výpočet střední a efektivní hodnoty.

2.2.5 Aplikace pro základní operace se signálem



Obr. 2.4: GUI aplikace pro základní operace se signálem.

Druhá aplikace navazuje na první aplikaci, rozšiřuje ji však o operace změny měřítka a rozsahu zobrazovaných hodnot. Graf oproti první aplikaci zobrazuje i záporné hodnoty, průsečík os se tedy nachází uprostřed zobrazovaného okna.

Nově přidaná operace změny měřítka je provázána se změnou periody a její chování je blíže popsáno v teoretické části, v kapitole 1.1.2. Prakticky platí mezi periodou a měřítkem nepřímá úměra. Realizace této operace v kódu je následující:

```
const zpracujZmenuMeritka = (T: string) => {
  const noveMeritko = Number.parseFloat(T)
  nastavMeritko(noveMeritko)
  const novaPerioda = (meritko * perioda) / noveMeritko
  nastavPeriodu(novaPerioda > 10 ? novaPerioda : 10)
}
```

Obdobně je provázána i změna periody, zde je však měřítko omezeno z obou stran:

```
const zpracujZmenuPeriody = (T: string) => {
  const novaPerioda = Number.parseInt(T)
  nastavPeriodu(novaPerioda)
```

```
  const noveMeritko = (perioda * meritko) / novaPerioda
  nastavMeritko(noveMeritko < 10 ?
  (noveMeritko > 1 ? noveMeritko : 1) : 10)
}
```

Jde jednoduše o přenastavení jak měřítka, tak periody. Přenastavení je podmíněno tak, aby nedošlo k příliš nízké hodnotě periody, což by mohlo být v grafu matoucí.

Další změnou je absence šumového signálu, který nemá v této aplikaci své opodstatnění, vzhledem k tomu, že se nejedná o periodický signál a tudíž jej nelze manipulovat pomocí posuvníku periody a změny měřítka. Nový posuvník rozsah okna při změně hodnoty předává knihovně `Chart.js` informaci o maximální a minimální hodnotě osy x, která je poté nově vykreslena.

Struktura aplikace je obdobná jako u první aplikace, jsou zde však jiné pomocné soubory knihovny:

```
/ ..... src
├── knihovna
│   ├── rovnice.ts
│   └── utility.ts
├── komponenty
│   ├── Teorie.tsx
│   └── ZmenaHodnoty.tsx
├── App.css
├── App.tsx
├── index.css
└── index.tsx
```

Hlavní soubor `App.tsx` začíná opět importy a registracemi pro knihovnu `Chart.js`. První rozdíl je v absolutní délce celého grafu, která je dána nejvzdálenějším pohledem na graf pomocí změny rozsahu okna. Dalším rozdílem oproti první aplikaci jsou záporné hodnoty na ose x. Tohoto je v aplikaci docíleno následujícím způsobem – nejprve deklarace pomocné funkce pro vytvoření pole celých čísel od nuly po zadané číslo:

```
const vytvorPoleCelychCisel = (delka: number) =>
Array.from(Array(delka).keys())
```

Poté získání záporné části:

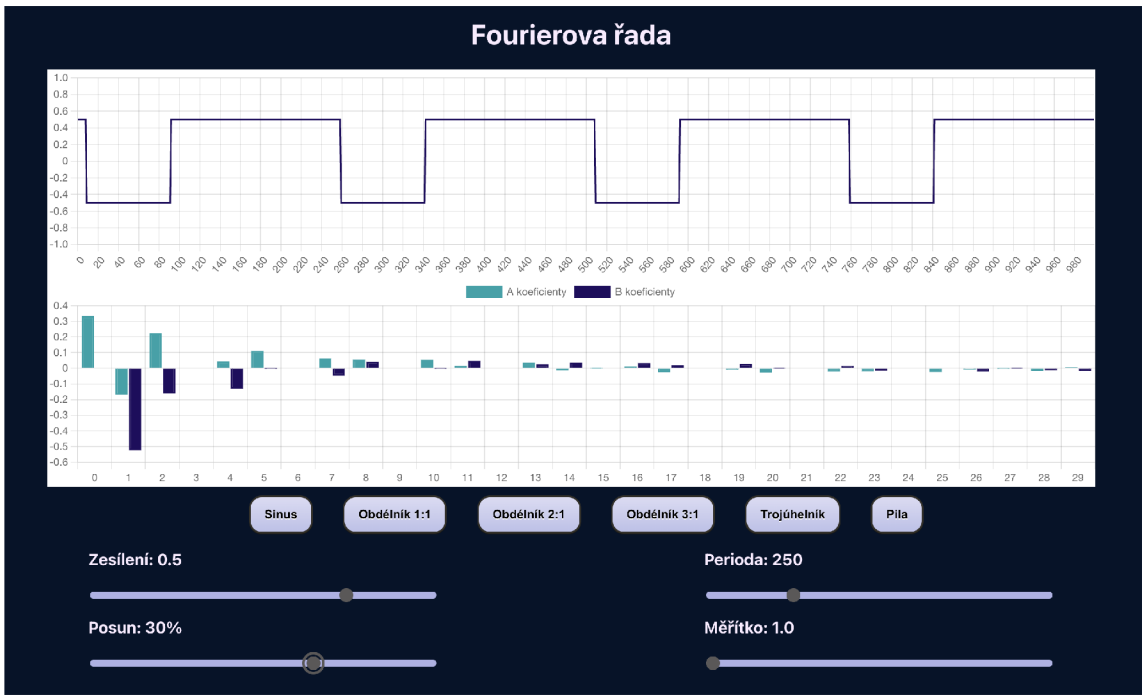
```
const maxRozliseniZaporne =
vytvorPoleCelychCisel(MAXIMALNI_DELKA_GRAFU)
  .map((x) => -x)
  .reverse()
  .slice(0,-1)
```

Pole bylo převedeno na záporné hodnoty a do opačného pořadí. Z tohoto nového pole je také potřeba odebrat nulu, neboť by se ve výsledném spojeném poli nacházela dvakrát.

Vytvoření nového pole a spojení těchto polí:

```
const maxRozliseni = maxRozliseniZaporne.concat(
  vytvorPoleCelychCisel(MAXIMALNI_DELKA_GRAFU),
)
```

2.2.6 Aplikace zobrazující vliv operací se signálem na Fourierovu řadu



Obr. 2.5: GUI aplikace zobrazující vliv operací se signálem na Fourierovu řadu.

Třetí aplikace rozšiřuje předešlé aplikace o nový graf zobrazující spektrum signálu pomocí výpočtu Fourierovy řady. V této aplikaci je tedy možné pozorovat vliv jednotlivých signálových operací a jejich kombinací na spektrum signálu.

Nový graf je implementován jako sloupcový `Chart.js` graf, kdy pro každou pozici jsou přítomny dva údaje – vždy pro dané hodnoty sinusové a kosinové složky, resp. a_n a b_n koeficientů, popsaných v teoretické části, konkrétně v kapitole 1.1.3.

Jednou z možností, jak získat hodnotu jednotlivých koeficientů spočívá v přímém výpočtu vycházejícím z obecných tvarů pro výpočet koeficientů. Funkce pro výpočet a_n koeficientů by tedy mohl vypadat následovně:

```
const ziskejKoeficientyA = (arr: number[], max: number, n: number) => {
  return (
    (2 / max) *
    integrace(arr.map((ft, t) => ft
    * Math.cos(n * ((2 * Math.PI) / max) * t)))
  );
};
```

Nevýhodou tohoto řešení by však byla značná nepřesnost v případě nízké periody, neboť výsledek by byl závislý pouze na velmi omezeném množství prvků. Vhodnějším řešením je tedy pro každý typ signálu koeficienty počítat zvlášť. Vzhledem k tomu, že operaci posunutí je možné implementovat obecně bez ohledu na původní signál, stačí pro každý signál vypočítat výsledný tvar pouze pro nejjednodušší případy. Jde tedy o případy kdy je signál sudý či lichý.

Sinusový signál

Nejjednodušším případem je sinusový signál, jehož b_1 koeficient je roven amplitudě a všechny ostatní koeficienty jsou nulové. Neboť v aplikaci je reprezentováno zesílení, které je uvažováno jako celý rozsah mezi maximem a minimem signálu, je potřeba tuto hodnotu podělit dvěma.

```
...
{
  a: new Array(arr.length).fill(0),
  b: [0, A / 2, ...new Array(arr.length - 2).fill(0)],
},
...
```

Výraz u klíče *a* lze přečíst jako „vytvoř pole stejné délky, jako je pole s názvem `arr` a naplň jej nulami“. Tato konstrukce bude využita i pro všechny následující signály. Pro klíč *b* se v podstatě jedná o to samé, jen je na pozici s indexem 1 (tedy na druhý prvek pole) dosazena hodnota amplitudy.

Obdélníkový signál

Uvažujeme-li obdélníkový signál jako sudou funkci, rovnice popisující jeho Fourierovu řadu jsou následující:

$$a_0 = A(2d - 1), \tag{2.6}$$

$$a_n = \frac{2A}{n\pi} \sin(n\pi d), \tag{2.7}$$

kde n je pořadí prvku, A je zesílení a d střída. Praktická implementace vypadá tedy následovně:

```
...
{
  b: new Array(arr.length).fill(0),
  a: arr.map((n, i) =>
    i === 0
      ? A * (2 * d - 1)
      : ((2 * A) / (n * Math.PI)) * Math.sin(n * Math.PI * d),
  ),
},
...
```

Trojúhelníkový signál

Kompletní výpočet rovnice trojúhelníkového signálu je možné najít v teoretické části diplomové práce. Výsledkem tohoto příkladu je následující rovnice:

$$a_n = 4A\frac{1-(-1)^n}{\pi^2 n^2}. \tag{2.8}$$

V kódu vypadá výpočet následovně:

```
  ...
  {
    b: new Array(arr.length).fill(0),
    a: arr.map(
      (n) =>
        (4 * (A / 2) * (1 - Math.pow(-1, n))) /
        (Math.pow(n, 2) * Math.pow(Math.PI, 2)),
    ),
  },
  ...
```

Pilový signál

Rovnice pro pilový signál je následující:

$$b_n = \frac{-A}{n\pi}. \tag{2.9}$$

Pilový signál je v tomto tvaru lichá funkce, tudíž je tentokrát nenulový b koeficient. V programu je pilový signál reprezentován následovně:

```
...
{
  b: arr.map((n) => -A / (n * Math.PI)),
  a: new Array(arr.length).fill(0),
},
...
```

Posunutí signálu

Vliv operace posunutí na Fourierovu řadu je pro jednotlivé signály společná a je zajištěna funkcí, která má na vstupu pět parametrů:

- a: hodnota koeficientu a pro danou harmonickou složku
- b: hodnota koeficientu b pro danou harmonickou složku
- p: posunutí
- ω: úhlová rychlost
- n: index dané harmonické složky

Výpočet nové hodnoty je dán rovnicemi:

$$a_n = a\cos(-\omega pn) - b\sin(-\omega pn). \tag{2.10}$$

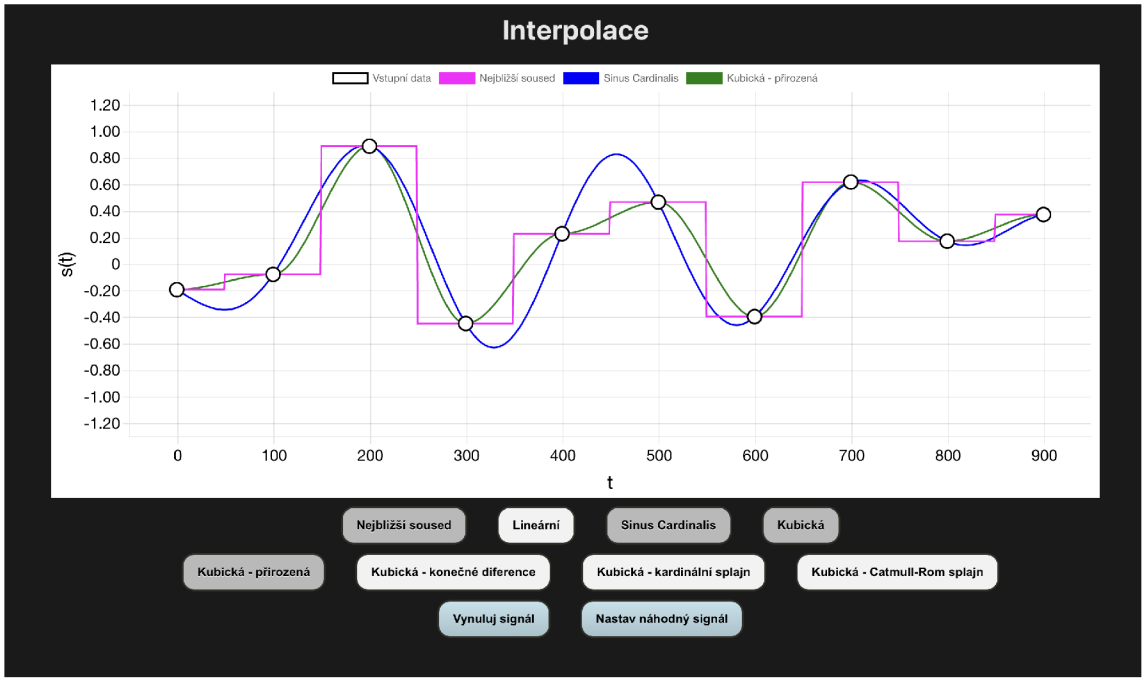
$$b_n = a\cos(-\omega pn) + b\sin(-\omega pn). \tag{2.11}$$

Což je v kódu realizováno následovně:

```
...
{
a: a * Math.cos(-w * p * n) - b * Math.sin(-w * p * n),
b: b * Math.cos(-w * p * n) + a * Math.sin(-w * p * n),
}
...
```

Oproti vzorcům 1.20 a 1.21 je argument funkcí sinus a kosinus záporný, neboť ve vizualizaci je uvažován posun v opačném směru.

2.2.7 Aplikace zobrazující interpolaci signálu



Obr. 2.6: GUI aplikace zobrazující interpolaci signálu.

Čtvrtá aplikace zaměřená na interpolaci nabízí uživateli deset vzorků a možnost nastavit hodnotu těchto jednotlivých vzorků. Vzorky jsou interpolovány pomocí uživatelem nastavené interpolační funkce (nejbližší soused, lineární, kubická a sinc). Rozložení této aplikace je velmi podobné jako u předchozích aplikací. Knihovna `Chart.js` ve výchozím stavu nepodporuje možnost posouvat s danými body pomocí myši – existuje však rozšiřující knihovna `chartjs-plugin-dragdata`, která toto chování umožňuje.

Pro přehlednost je funkcionalita aplikace rozdělena na dvě části. V jedné je samostatně zpracována kubická interpolace a v druhé všechny ostatní (nejbližší soused, sinc, lineární). Grafy se zobrazují podle stavu zadefinovaného pomocí `useState` syntaxe:

```
const [interpolace, nastavInterpolace] = useState(
  vychoziStavInterpolace
)
const [kubickeInterpolace, nastavKubickeInterpolace] = useState(
  vychoziStavKubickeInterpolace
)
```

kde výchozí stavy jsou pouze mapy Boolean hodnot.

```
  export const vychoziStavKubickeInterpolace = {
  kubickaPrirozena: false,
  kubickaKonecneDiference: false,
  kubickaKardinalniSplajn: false,
  kubickaCatmullRomSplajn: false,
}

export const vychoziStavInterpolace = {
  nejblizsiSoused: false,
  linearni: true,
  kubicka: false,
  sinc: false,
}
```

Ve výchozím stavu jsou tedy všechny grafy kromě lineární interpolace vypnuté a kliknutím na tlačítko se změní hodnota pro daný klíč objektu udržujícího stav na opačnou. Změna je zaznamenána také v proměnných `datasety` a `kubickaDatasety`, které jsou generovány ze seznamů možných hodnot. Je-li je při procházení seznamu nalezena ve stavu pravdivá hodnota, dochází k vytvoření nového datasetu (jenž se projevuje v aplikaci jako graf) a např. pro nekubické interpolace má následující tvar:

```
  {
    label: mapaInterpolaceNazev[interpolace],
    data: namapujVstupNaInterpolaci(
      interpolace as InterpolaceBezKubicke,
      vstupniSignal,
    ),
    dragData: false,
    radius: 0,
    borderColor: mapaInterpolaceBarva[interpolace],
    backgroundColor: mapaInterpolaceBarva[interpolace],
    borderWidth: 2,
  }
```

V pomocném souboru `mapy.ts` se nachází informace o tom, jakou má mít daný graf barvu nebo jak se má zobrazovat jeho název. K výsledku výpočtu dané interpolace lze přistoupit pomocí klíče `data`. Funkce `namapujVstupNaInterpolaci` slouží k vybrání dané funkce na základě názvu interpolace:

```
export const namapujVstupNaInterpolaci = (
  nazev: InterpolaceBezKubicke,
  signal: Bod[],
) =>
  ({
    nejblizsiSoused: nejblizsiSousedInterpolace,
    linearni: linearniInterpolace,
    sinc: sincInterpolace,
  }[nazev](signal))
```

Funkce vyresInterpolaci

Každá z těchto funkcí kromě interpolace metodou sinc využívá společné jádro, funkci `vyresInterpolaci`, která vypadá následovně:

Výpis 2.3: Funkce pro výpočet interpolace mezi dvěma body.

```
1  const vyresInterpolaci = (
2    signal: Bod[],
3    interpolacniFunkce: (a: Bod, b: Bod, x: number) => number,
4  ) =>
5    signal.slice(1).reduce(
6      ([predchoziBod, akumulator], aktualniBod) => {
7        const noveBody = vytvorPoleCelychCisel
8          (POCET_BODU_MEZI_BODY)
9          .map((i) => predchoziBod.x + i)
10         .map((x) => ({
11           x,
12           y: interpolacniFunkce(predchoziBod, aktualniBod, x),
13         }))
14       return [aktualniBod, [...akumulator, ...noveBody]]
15       as const
16     },
17     [signal[0], [] as readonly Bod[]] as const,
18   )[1]
```

Funkce přijímá signál, který je reprezentován pomocí bodů, které jsou datového typu objekt s klíči x a y. Druhým argumentem je interpolační funkce, která přijímá dva sousední body a a b a také aktuální x souřadnici. Interpolace je tedy realizována po částech tak, že jsou mezi dvěma sousedními body vypočítány hodnoty pro každou souřadnici a poté se počítá prostor mezi následujícími dvěma sousedními body, dokud není naplněn celý graf.

JavaScriptová metoda `reduce`, uvedena v kapitole 1.2.2, umožňuje na začátku definovat výchozí hodnotu určující tvar, do kterého se bude funkce postupně redukovat, což je v tomto případě uspořádaná dvojice (tuple), kde první prvek je předchozí bod a druhý prvek akumulátor výsledků (ve výpisu řádek 18). Pomocí metody `slice` je nejprve ze signálu odebrán první bod (řádek 5), který je však předán ve výchozí hodnotě. Konstanta `POCET_BODU_MEZI_BODY` určuje rozlišení mezi každými dvěma sousedními body, v aplikaci se pracuje s hodnotou 100.

Funkce `vytvorPoleCelychCisel(POCET_BODU_MEZI_BODY)` tedy vytvoří pole ve tvaru [0,1,2,...,99], které je v následující mapovací funkci (řádek 9) posunuto na správné místo na základě přičtení souřadnice x předchozího bodu. Dále je na toto pole aplikována interpolační funkce, která počítá výsledné hodnoty na ose y. V návratové hodnotě je předán na první pozici aktuální bod (který se v další iteraci stává předchozím bodem) a také akumulátor obohacený o nově vypočítané body. Po iteraci přes všechny dvojice bodů vracíme pouze druhý prvek uspořádané dvojice, který obsahuje pole všech hodnot (akumulátor).

Interpolace metodou nejbližšího souseda

Interpolace metodou nejbližšího souseda je realizována pomocí následujícího kódu:

```
const nejblizsiSousedInterpolace = (signal: Bod[]) =>
  vyresInterpolaci(signal, (a, b, x) =>
  (x < (a.x + b.x) / 2 ? a.y : b.y))
```

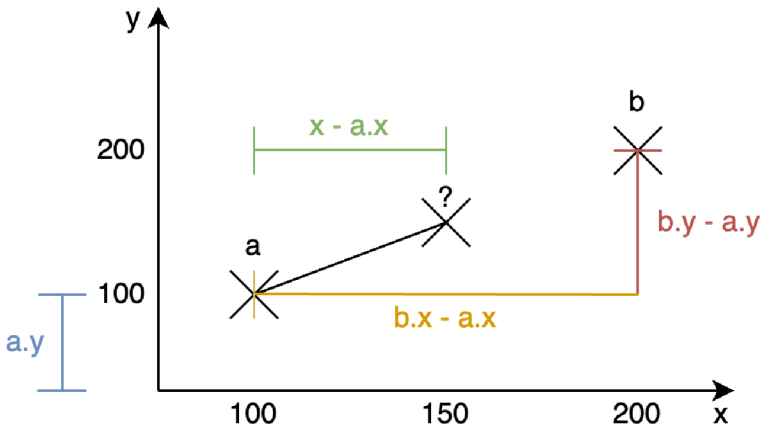
Na základě podmínky dochází ke zjištění, zdali se aktuální hodnota osy x nachází v první nebo druhé polovině intervalu mezi dvěma sousedními body a a b.

Lineární interpolace

Lineární interpolace je realizována následovně:

```
const linearniInterpolace = (signal: Bod[]) =>
  vyresInterpolaci(
    signal, (a, b, x) =>
    a.y + ((x - a.x) * (b.y - a.y)) / (b.x - a.x),
  )
```

Jednotlivé vzdálenosti pro hledaný bod označený symbolem ? jsou znázorněny na následujícím grafu:



Obr. 2.7: Znázornění vzdáleností při výpočtu lineární interpolace.

Interpolace pomocí funkce sinc

Funkci sinc lze v kódu implementovat následovně:

```
const sinc = (x: number) => (x !== 0 ? Math.sin(x) / x : 1)
```

Výpočet interpolace pomocí funkce sinc není možné realizovat pomocí funkce `vyresInterpolaci`, neboť se při něm uvažují pouze dva sousední body. Pro interpolaci pomocí funkce sinc je potřeba každý bod proložit funkcí sinc. Tímto způsobem vznikne pro každý bod pole hodnot a tato pole je potřeba dohromady sečíst.

```
const sincInterpolace = (signal: Bod[]) => {
  const poleNulovychHodnot = vytvorPoleCelychCisel(
    POCET_BODU_MEZI_BODY * (signal.length - 1),
  ).map((_, i) => ({ x: i, y: 0 }))
  return signal.reduce(
    (akumulator, aktualniBod) =>
      akumulator.map((b) => ({
        x: b.x,
        y: b.y + aktualniBod.y
           * sinc((Math.PI / POCET_BODU_MEZI_BODY)
           * (b.x - aktualniBod.x)),
      })),
    poleNulovychHodnot,
  )
}
```

Zde je nejprve vytvořeno pole délky celého grafu s nulovými hodnotami pro osu y. Toto pole je následně použito jako výchozí hodnota pro redukci. Při jednotlivých iteracích zůstává x nepozměněno, k jednotlivým y hodnotám je přičítána aktuální hodnota funkce sinc pro aktuální bod. Je přitom nutné upravit rozlišení (vzorkovací frekvenci) tak, aby odpovídalo reálné vzdálenosti mezi body.

Kubická interpolace

Výpočet kubické interpolace je v aplikaci realizován čtyřmi způsoby, hlavní část výpočtu je však společná. Odlišnosti jsou dané výpočtem tangent okolo jednotlivých bodů. Společné jádro je implementováno následovně:

```
const kubickaInterpolace =
  (tangenty: Record<number, number>) => (signal: Bod[]) =>
    vyresInterpolaci(signal, (a, b, x) => {
      const k1 = tangenty[a.x]
      const k2 = tangenty[b.x]
      const t = (x - a.x) / (b.x - a.x)
      const U = k1 * (b.x - a.x) - (b.y - a.y)
      const V = -k2 * (b.x - a.x) + b.y - a.y

      return (1 - t) * a.y + t * b.y + t
             * (1 - t) * (U * (1 - t) + V * t)
  })
```

Jedná se v podstatě jen o implementaci vzorců 1.37 a 1.38. Nejjednodušším případem je přirozená kubická interpolace, kdy hodnoty jednotlivých tangent jsou nulové:

```
{
...
kubickaPrirozena: kubickaInterpolace(
      signal.reduce((akumulator, curr) =>
      ({ ...akumulator, [curr.x]: 0 }), {}),
    ),
...
}
```

Pro výpočet tangent dalšími metodami je vhodné zavést pomocnou funkci, která ji umožní vypočítat ze dvou bodů. V této funkci lze také jednoduše zajistit, že nedojde k dělení nulou.

```
const vypocitejTangentu = (a: Bod, b: Bod) => {
  const dx = a.x - b.x
  const dy = a.y - b.y
  if (dx === 0 || dy === 0) return 0
  return dy / dx
}
```

Výpočet kubické interpolace pomocí metody konečných diferencí [12], resp. výpočet jednotlivých tangent, popisuje vzorec 1.39. Jeho implementace je realizována následovně:

Výpis 2.4: Funkce pro výpočet tangent bodů kubické interpolace metodou konečných diferencí.

```
1  const vytvorTangentyKonecneDiference = (
2    signal: Bod[],
3  ): Record<number, number> => {
4    const tangenty = {} as Record<number, number>
5    for (let i = 1; i < signal.length - 1; i++) {
6      const py =
7        (1 / 2) *
8        (vypocitejTangentu(signal[i + 1], signal[i]) +
9          vypocitejTangentu(signal[i], signal[i - 1]))
10     tangenty[signal[i].x] = py
11   }
12   // okrajove podminky
13   tangenty[signal[0].x] =
14   vypocitejTangentu(signal[0], signal[1])
15   const d = signal.length - 1
16   tangenty[signal[d].x] =
17   vypocitejTangentu(signal[d], signal[d -1])
18   return tangenty
19 }
```

Nejprve je definován prázdný objekt tangenty, jehož klíče budou pozice jednotlivých bodů na ose x, proto tedy typová definice `Record<number, number>`. For cyklus je omezen tak, aby bylo možné vždy zpracovávat trojici po sobě jdoucích bodů, protože to je interval, ze kterého jsou jednotlivé tangenty vypočítány. Pro každou trojici bodů je do objektu `tangenty` pod daným číselným klíčem uložena hodnota. Neboť tato metoda využívá vždy trojici bodů, je třeba zvolit okrajové podmínky. V tomto případě jsou tangenty počítány pouze ze dvou sousedních bodů na začátku a konci grafu.

Další metodou výpočtu tangent je metoda kardinálního splinu, která přijímá také parametr napětí (tension) označovaný jako c, viz 1.40. Její implementace je podobná, tangenty se však nepočítají mezi dvěma sousedním body, ale pro každý bod z předcházejícího a následujícího bodu. Počítáme-li tedy tangenty pro druhý bod, do vzorce vstupuje bod první a třetí.

Výpis 2.5: Funkce pro výpočet tangent bodů kubické interpolace metodou kardinálního splinu.

```
const vytvorTangentyKardinalniSplajn = (
    signal: Bod[],
    c: number,
  ): Record<number, number> => {
    const tangenty = {} as Record<number, number>
    for (let i = 1; i < signal.length - 1; i++) {
      const py = (1 - c) *
      vypocitejTangentu(signal[i + 1], signal[i - 1])
      tangenty[signal[i].x] = py
    }
    tangenty[signal[0].x] = (1 - c) *
    vypocitejTangentu(signal[0], signal[1])
    const d = signal.length - 1
    tangenty[signal[d].x] = (1 - c) *
    vypocitejTangentu(signal[d], signal[d - 1])
    return tangenty
}
```

Speciálním případem kardinálního splinu je tzv. Catmull–Rom spline, kdy je funkce `vytvorTangentyKardinalniSplajn` zavolána s parametrem napětí o hodnotě 0,5.

Závěr

V rámci diplomové práce byla nejprve vypracována teoretická část zahrnující střední a efektivní hodnotu signálu, základní operace se signály, jejich vliv na Fourierovu řadu signálu a také různé druhy interpolací diskrétních signálů. V druhé polovině teoretické části byly poté uvedeny a popsány zvolené webové technologie, které byly použity pro vytvoření výsledných aplikací. Kromě jazyka JavaScript pojednává diplomová práce i o jeho rozšíření o statické typy v podobě jazyka TypeScript, knihovně pro tvorbu front–endových aplikací React, webových technologiích HTML a CSS a také o knihovně zaměřené na grafickou vizualizaci dat Chart.js.

Zadání diplomové práce zahrnuje čtyři funkční interaktivní aplikace pro výukové účely, které byly v rámci tvorby práce implementovány. Tyto aplikace budou brzy dostupné na adrese `https://www.utko.fekt.vut.cz/~rajmic/applets` a umožní studentům si osahat několik konceptů, se kterými se během studia potkají.

Interaktivní webové aplikace mohou ze své podstaty studentům či kterýmkoliv dalším zájemcům o témata těchto aplikací poskytnout unikátní perspektivu na danou problematiku, která může pomoci látku názorně demonstrovat a následně ji pochopit a vytvořit si pro ni i určitou intuici. Tato diplomová práce je příspěvkem do tohoto vzdělávacího procesu využívajícího moderní technologie s možností přístupu z kteréhokoliv počítače připojenému k internetu.

Literatura

[1] BRANČÍK, Lubomír: *Elektrotechnika 1.* Vysoké učení technické v Brně, 2004. ISBN: 80-214-2607-1.

[2] SMÉKAL, Zdeněk: *Analýza signálu a soustav.* Vysoké učení technické v Brně, 2012. ISBN: 978-80-214-4453-9.

[3] SMITH, Steven W.: *The scientist and engineer's guide to digital signal processing.* San Diego: California Technical Pub., 1997. ISBN: 09-660-1763-3.

[4] KOLÁŘOVÁ, Edita: *Matematika 2.* Vysoké učení technické v Brně, 2007. ISBN: 978-80-214-3442-4.

[5] CHEEVER, Erik: *Linear Physical Systems Analysis* [online] [cit. 10.11.2022] Dostupné z URL: <https://lpsa.swarthmore.edu/>.

[6] KALVODA, Tomáš: *Kubická interpolace.* [online] [cit. 22.10.2022] Dostupné z URL: <https://marast.fit.cvut.cz/cs/blog_posts/16>.

[7] HEJDOVÁ, Martina: *Interpolace obrazů.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2010. 36 s. Vedoucí bakalářské práce doc. Ing. Radim Kolář, Ph.D.

[8] RAJMIC, Pavel; SCHIMMEL, Jiří: *Moderní počítačová grafika.* Vysoké učení technické v Brně, 2013. ISBN: 978-80-214-4906-0.

[9] LEHMANN, T. M., GÖNNER, C., SPITZER, K. *K. Survey: Interpolation Methods in Medical ImageProcessing.* IEEE Transactions on Medical Imaging. 1999, vol. 18, No. 11.

[10] YOUNG, Todd, MOHLENKAMP, Martin J. *Introduction to Numerical Methods and Matlab Programming for Engineers* [online] [cit. 22.4.2023] Dostupné z URL: <http://www.ohiouniversityfaculty.com/youngt/IntNumMeth/book.pdf>.

[11] FARIN, Gerald. *Curves and Surfaces for CAGD: A Practical Guide (5th ed.).* Morgan Kaufmann, 2002, 520 s. ISBN: 9781558607378.

[12] BURDEN, Richard, FAIRES, Douglas. *Numerical Analysis (10th ed.)* Cengage Learning, 2015, 912 s. ISBN: 9781305253674.

[13] HOUSE, Donald H., *Spline Curves* [online] [cit. 15.4.2023] Dostupné z URL: <https://people.computing.clemson.edu/~dhouse/courses/405/notes/splines.pdf>.

[14] MALCHER, Tomáš: *Webové aplikace pro podporu výuky audiosignálů*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2022, 57 s. Bakalářská práce. Vedoucí práce: doc. Mgr. Pavel Rajmic, Ph.D.

[15] CARBONNELLE, Pierre. *PYPL PopularitY of Programming Language*. [online] [cit. 19.11.2022] Dostupné z URL: <https://pypl.github.io/PYPL.html>.

[16] FLANAGAN, David: *JavaScript: The Definitive Guide, Seventh Edition*. O'Reilly Media, Inc, USA, 2020, 600 s. ISBN: 978-1-491-95202-3.

[17] Mozilla Corporation. *JavaScript* [online] [cit. 30.4.2023] Dostupné z URL: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>.

[18] Kolektiv autorů: *TypeScript Handbook*. [online] [cit. 22.10.2022] Dostupné z URL: <https://www.typescriptlang.org/assets/typescript-handbook.pdf>.

[19] KUŘE, Dominik. *Interaktivní vyhledávání v on-line archivu obrazových a audiovizuálních děl*. Brno, 2020, 66 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: doc. Ing. Jiří Schimmel, PhD.

[20] JANOVSKÝ, Dušan. *CSS*. [online] [cit. 22.10.2022] Dostupné z URL: <https://www.jakpsatweb.cz/css>.

[21] Meta Platforms. *Learn React* [online] [cit. 10.4.2023] Dostupné z URL: <https://react.dev/learn>.

[22] Kolektiv autorů. *Dokumentace Chart.js* [online] [cit. 19.11.2022] Dostupné z URL: <https://www.chartjs.org/docs/latest/>.

Seznam symbolů a zkratek

|	logické nebo pro typové výrazy v jazyce TypeScript
1D	jednodimenzionální
2D	dvoudimenzionální
arr	array – pole
const	konstanta
CSS	Cascading Style Sheets – jazyk pro popis stylizace webové stránky
GUI	Graphical User Interface – grafické uživatelské rozhraní
HTML	Hypertext Markup Language – jazyk pro tvorbu webových stránek
JS	JavaScript
JSX	syntaktické rozšíření kombinující HTML, CSS a JavaScript
obj	objekt
PYPL	PopularitY of Programming Language – žebříček popularity programovacích jazyků
RMS	Root mean square – efektivní hodnota, viz 1.1.1
sinc	Sinus cardinalis, viz 1.1.6
TSX	JSX pro TypeScript
UI	User Interface – Uživatelské rozhraní

A Uživatelská příručka

Na lokalním počítači je potřeba mít nainstalováno prostředí Node.js, dostupné z `https://nodejs.org/en/` a balíčkovací manažer yarn, dostupný z `https://classic.yarnpkg.com/lang/en/docs/install`. Pomocí příkazového řádku vstupte do kořenového adresáře. Příkazem `yarn install` se nainstalují požadované závislosti. Balíčkovací manažer yarn umožňuje pracovat s jednotlivými aplikacemi zvlášť pomocí klíčového slova `workspace`. Jednotlivé aplikace lze tedy spustit pomocí příkazu `yarn workspace NAZEV_BALICKU start`. Pro vystavění aplikace do statických souborů použijte příkaz `yarn workspace NAZEV_BALICKU build`.

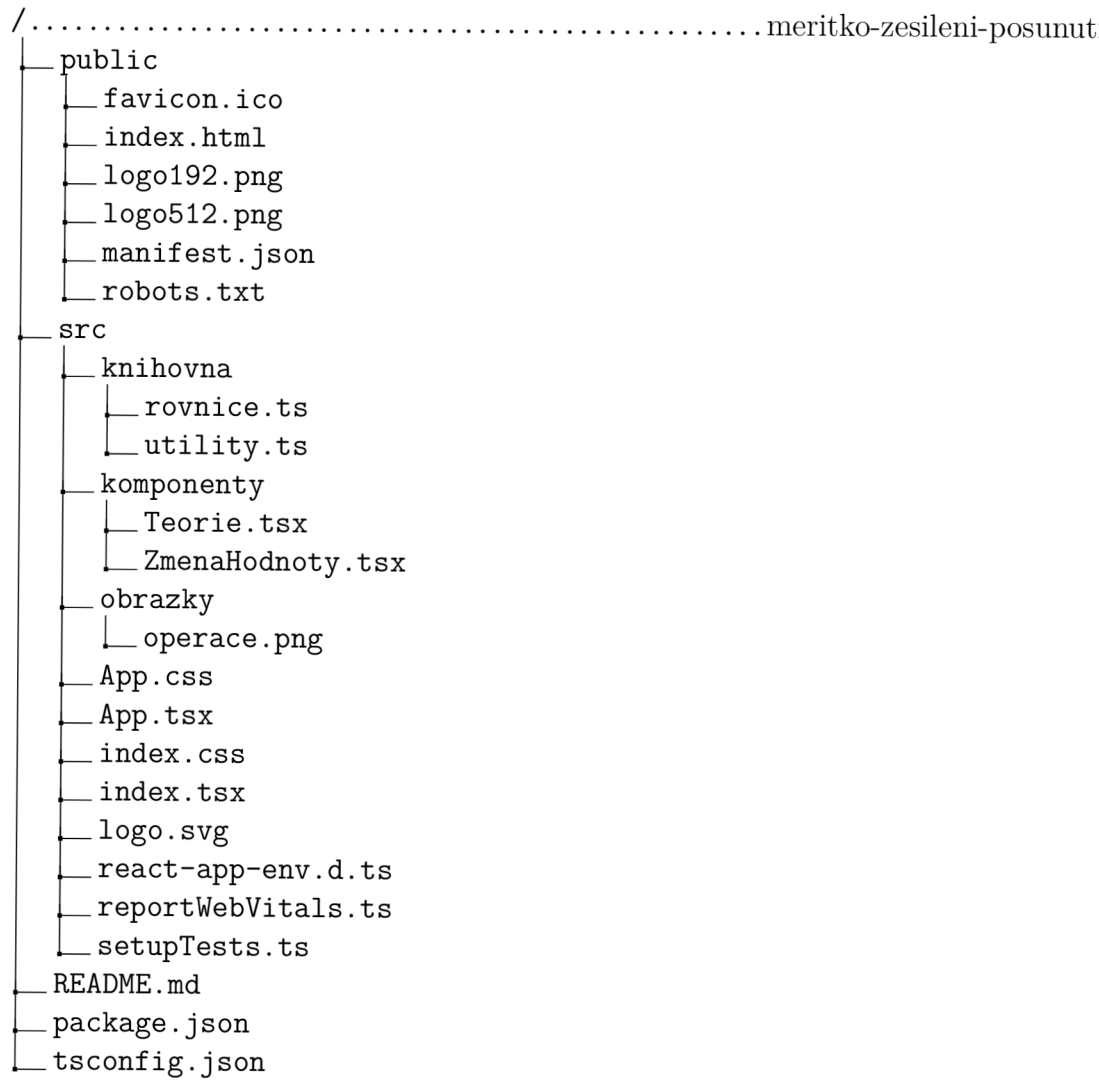
Za `NAZEV_BALICKU` je možné dosadit:

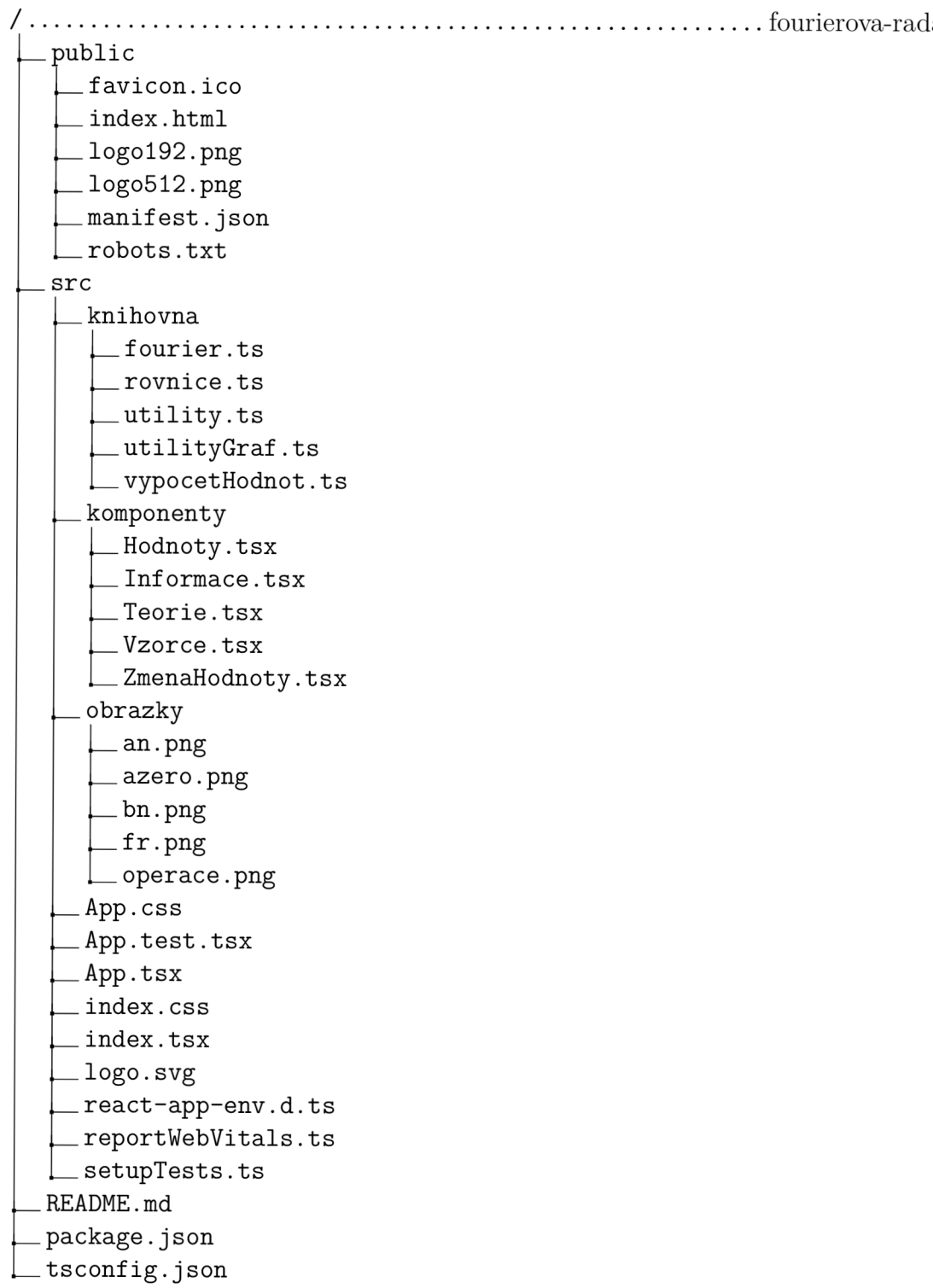
- `efektivni-stredni`
- `meritko-zesileni-posunuti`
- `fourierova-rada`
- `interpolace`

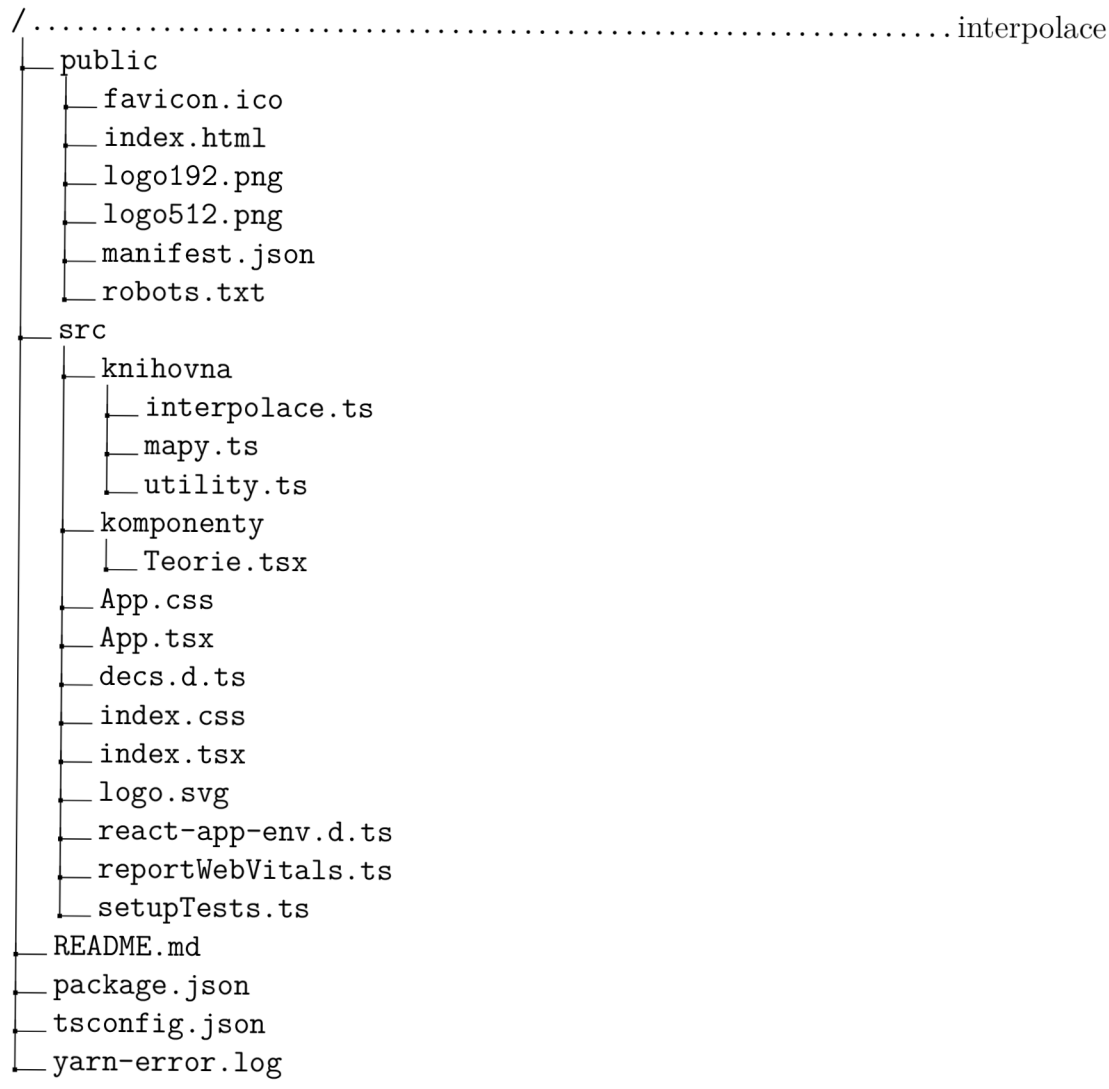
B Zdrojový kód

Příloha obsahuje text diplomové práce ve formátu pdf, složku se zdrojovým kódem a kopii uživatelské příručky. Aplikace se nacházejí ve složce `packages`. Obsah přílohy je popsán detailněji v textu práce. Struktura přílohy je následující:

```
/ ..... efektivni-stredni
├── public
│   ├── favicon.ico
│   ├── index.html
│   ├── logo192.png
│   ├── logo512.png
│   ├── manifest.json
│   └── robots.txt
├── src
│   ├── knihovna
│   │   ├── rovnice.ts
│   │   ├── utility.ts
│   │   ├── utilityGraf.ts
│   │   └── vypocetHodnot.ts
│   ├── komponenty
│   │   ├── Hodnoty.tsx
│   │   ├── Informace.tsx
│   │   ├── Vzorce.tsx
│   │   └── ZmenaHodnoty.tsx
│   ├── obrazky
│   │   ├── efektivni.png
│   │   └── stredni.png
│   ├── App.css
│   ├── App.tsx
│   ├── index.css
│   ├── index.tsx
│   ├── react-app-env.d.ts
│   ├── reportWebVitals.ts
│   └── setupTests.ts
├── README.md
├── efektivni-stredni-build.zip
├── package.json
├── tsconfig.json
└── yarn-error.log
```


/ meritko-zesileni-posunuti
public
favicon.ico
index.html
logo192.png
logo512.png
manifest.json
robots.txt
src
knihovna
rovnice.ts
utility.ts
komponenty
Teorie.tsx
ZmenaHodnoty.tsx
obrazky
operace.png
App.css
App.tsx
index.css
index.tsx
logo.svg
react-app-env.d.ts
reportWebVitals.ts
setupTests.ts
README.md
package.json
tsconfig.json


/ ..... fourierova-rada
public
favicon.ico
index.html
logo192.png
logo512.png
manifest.json
robots.txt
src
knihovna
fourier.ts
rovnice.ts
utility.ts
utilityGraf.ts
vypocetHodnot.ts
komponenty
Hodnoty.tsx
Informace.tsx
Teorie.tsx
Vzorce.tsx
ZmenaHodnoty.tsx
obrazky
an.png
azero.png
bn.png
fr.png
operace.png
App.css
App.test.tsx
App.tsx
index.css
index.tsx
logo.svg
react-app-env.d.ts
reportWebVitals.ts
setupTests.ts
README.md
package.json
tsconfig.json


/ interpolace
public
favicon.ico
index.html
logo192.png
logo512.png
manifest.json
robots.txt
src
knihovna
interpolace.ts
mapy.ts
utility.ts
komponenty
Teorie.tsx
App.css
App.tsx
decs.d.ts
index.css
index.tsx
logo.svg
react-app-env.d.ts
reportWebVitals.ts
setupTests.ts
README.md
package.json
tsconfig.json
yarn-error.log

/ ..... kořenový adresář
- packages
- package.json
- tsconfig.json
- yarn.lock
- .eslintrc.json