MASARYKOVA UNIVERZITA
Přírodovědecká fakulta
Ústav matematiky a statistiky

Diplomová práce

Brno 2025 Bc. Jan Macharáček

MASARYKOVA
UNIVERZITA
Přírodovědecká fakulta
Ústav matematiky a statistiky

Komplexní dynamické systémy

Diplomová práce

Bc. Jan Macharáček

Vedoucí práce: RNDr. Veronika Eclerová, Ph.D. **Brno 2025**

Bibliografický záznam

Autor:	Bc. Jan Macharáček Přírodovědecká fakulta, Masarykova univerzita Ústav matematiky a statistiky
Název práce:	Komplexní dynamické systémy
Studijní program:	Matematika
Studijní obor:	Finanční a pojistná matematika
Vedoucí práce:	RNDr. Veronika Eclerová, Ph.D.
Akademický rok:	2024/2025
Počet stran:	$x + 102$
Klíčová slova:	Fraktál; Juliovy množiny; Komplexní dynamické systémy; Mandelbrotova množina; Hyperbolická komponenta; Iterace; Vizualizační metody; Algoritmus únikového času; Modifikovaná metoda inverzní iterace; Metoda detekce hran; Metoda kontroly obdélníků; Webová aplikace; JavaScript

Bibliographic Entry

Author:	Bc. Jan Macharáček Faculty of Science, Masaryk University Department of Mathematics and Statistics
Title of Thesis:	Complex dynamical systems
Degree Programme:	Mathematics
Field of Study:	Financial and acturial mathematics
Supervisor:	RNDr. Veronika Eclerová, Ph.D.
Academic Year:	2024/2025
Number of Pages:	$x + 102$
Keywords:	Fractal; Julia sets; Complex dynamical systems; Mandelbrot set; Hyperbolic components; Iteration; Visualisation methods; Escape time algorithm; Modified inverse iteration method; Edge detection method; Rectangle checking method; Web application; JavaScript

Abstrakt

Diplomová práce pokračuje ve studiu komplexních dynamických systémů, konkrétně Juliovy nebo Mandelbrotovy množiny. Rozebírá jejich vlastnosti a nové možnosti vizualizace. Práce se zaměřuje zejména na generalizaci těchto fraktálních struktur ve smyslu rozšíření na transcendentní funkce a polynomické funkce s obecným racionálním exponentem. Následně se věnuje pojmu periodické komponenty zahrnující jejich formální definici, matematické vlastnosti a možnosti vizualizace. Cílem práce je také poskytnout hlubší matematický vhled do studované problematiky a to především v oblasti komplexní analýzy. Praktická část diplomové práce navazuje na výsledky z bakalářské práce a rozšiřuje webové rozhraní o nové funkce, umožňující pokročilou vizualizaci a interaktivní analýzu komplexních dynamických systémů. Práce tak propojuje teoretické poznatky z oblasti komplexní analýzy s grafickou vizualizací, čímž přispívá k lepšímu porozumění rozebíraného tématu.

Abstract

The thesis continues the study of complex dynamical systems, specifically Julia and Mandelbrot sets. It analyzes their properties and explores new visualization possibilities. The work focuses particularly on the generalization of these fractal structures by extending them to transcendental functions and polynomial functions with a general rational exponent. Subsequently, it delves into the concept of periodic components, including their formal definition, mathematical properties, and visualization methods. The aim of the thesis is also to provide a deeper mathematical insight into the studied topic, primarily in the field of complex analysis. The practical part builds upon the results of the bachelor's thesis and extends the web interface with new features that enable advanced visualization and interactive analysis of complex dynamical systems. Thus, the thesis connects theoretical knowledge from the field of complex analysis with graphical visualization, contributing to a better understanding of the studied topic.

MUNI
SCI

MASARYKOVA UNIVERZITA
PŘÍRODOVĚDECKÁ FAKULTA
KOTLÁŘSKÁ 2, 611 37 BRNO
IČ: 00216224
DIČ: CZ00216224

ZADÁNÍ DIPLOMOVÉ PRÁCE

Akademický rok: 2024/2025

Ústav:	Přírodovědecká fakulta
Student:	Bc. Jan Macharáček
Program:	Aplikovaná matematika
Specializace:	Finanční a pojistná matematika

Ředitel *ústavu* PřF MU Vám ve smyslu Studijního a zkušebního řádu MU určuje diplomovou práci s názvem:

Název práce:	Komplexní dynamické systémy
Název práce anglicky:	Complex dynamical systems
Jazyk práce:	čeština

Oficiální zadání:
Student bude pokračovat ve studiu vlastností Juliových množin a jejich vizualizacích. Konkrétně zaměří (i) na generalizací pojmu Juliova množina a (ii) na tzv. ”periodic bubbles” - jejich formální definici a vlastnosti, ale také možnosti vizualizace. Cílem práce je také poskytnout hlubší matematický vhled do studované problematiky a to především v oblasti komplexní analýzy. Praktická část práce bude rozšířením webového rozhraní vytvořeného ve studentově bakalářské práci.

Literatura: CARLESON, Lennart; GAMELIN, Theodore W. Complex dynamics. Springer Science & Business Media, 2013.

BEARDON, Alan F. Iteration of rational functions: Complex analytic dynamical systems. Springer Science & Business Media, 2000.

Vedoucí práce:	RNDr. Veronika Eclerová, Ph.D.
Datum zadání práce:	9. 9. 2022
V Brně dne:	1. 1. 2025

Zadání bylo schváleno prostřednictvím IS MU.

Bc. Jan Macharáček, 17. 10. 2022
RNDr. Veronika Eclerová, Ph.D., 18. 10. 2022
RNDr. Jan Vondra, Ph.D., 25. 10. 2022

Poděkování

Na tomto místě bych rád poděkoval RNDr. Veronice Eclerové, Ph.D. za odborné vedení, cenné rady a připomínky po celou dobu zpracovávání diplomové práce. Děkuji také své rodině za podporu během celého studia. V neposlední řadě patří velké poděkování mé přítelkyni za trpělivost, podporu a motivaci, které významně přispěly k napsání této práce.

Prohlášení

Prohlašuji, že jsem svoji diplomovou práci vypracoval samostatně pod vedením vedoucího práce s využitím informačních zdrojů, které jsou v práci citovány.

Brno 2. ledna 2025

.....

Bc. Jan Macharáček

Obsah

Přehled použitého značení

Pro snazší orientaci v textu zde čtenáři předkládáme přehled základního značení, které se v celé práci vyskytuje.

$\mathbb{C}$	množina všech komplexních čísel
$\mathbb{R}$	množina všech reálných čísel
$\mathbb{Q}$	množina všech racionálních čísel
$\mathbb{N}$	množina všech přirozených čísel
$\mathbb{Z}$	množina všech celých čísel
$\mathbb{Z}^+$, $\mathbb{Z}^-$	množina všech kladných, záporných celých čísel
$z \mapsto z^d + c$,$f(z) = z^d + c$	zobrazení, funkce
$z_n = f^n(z_0)$	iterační předpis
$\frac{d}{dz} f$	derivace f podle z
$\arg(z)$, θ	orientovaný úhel (argument) z
$\bar{c}$	komplexně sdružené číslo k c
$f^k(z_0)$	k-tá iterace funkce f pro počáteční hodnotu z_0
f^{-1}	inverzní funkce k funkci f
$f'(z)$	derivace funkce f v bodě z
$P_c(f)$	prisoner set
$E_c(f)$	escape set
J, $J_c(f)$, J_c	Juliova množina
K_c, $K_c(f)$	Filled Juliova množina
M,M_d	Mandelbrotova množina
B_c	non-escape množina
C_c	množina souvislosti
$\lambda_s, \lambda_e, \lambda_k$	hlavní, zárodečný, kompozitní hlavní lalok
∂K	hranice množiny K
$\overline{K}$	uzávěr množiny K
z^*	pevný bod
$P_n(c), Q_n(c)$	polynomy s kořeny hyperbolické komponenty periody n
U_n	počet hyperbolických komponent s periodou n

$N(z_n)$	Newtonova funkce
$A(x)$	oblast přitažení bodu x
$\tilde{f}(z)$	konjugovaná funkce k f
$\lfloor d \rfloor$	dolní celá část d
$\mathrm{Re}(z)$	reálná část čísla z
$\mathrm{Im}(z)$	imaginární část čísla z
$\mathbf{v}$	vektor
$\Delta\mu$	gradient μ

Úvod

Diplomová práce navazuje na bakalářskou práci na téma Juliovy množiny. Práce bude mít dvě hlavní části. Jako první se zaměříme na generalizaci Juliovy a Mandelbrotovy množiny. Následně rozebereme pojem periodické komponenty. Jejich formální definici, vlastnosti a také možnosti vizualizace. Cílem práce je také poskytnout hlubší matematický vhled do studované problematiky a to především v oblasti komplexní analýzy. Praktická část práce bude rozšířením webového rozhraní vytvořeného v bakalářské práci.

Práce je rozdělena do šesti kapitol. První kapitola slouží jako přípravná a poskytne čtenáři potřebné základní vědomosti ohledně elementárních funkcí komplexní proměnné, jako jsou exponenciální a trigonometrické funkce.

Druhá kapitola detailně rozebere Mandelbrotovu množinu. Nejprve shrneme poznatky o Mandelbrotově množině kvadratického zobrazení. Následně se zaměříme na její zobecnění. Uvedeme formální definici a její vlastnosti. Závěr druhé kapitoly zavede pojem komplexní odmocniny z jedné a její transformaci na bifurkační body na hranici kardioidu a komponenty s periodou dva.

Třetí kapitola se věnuje pojmu periodické komponenty. Jako první připomeneme geometrickou strukturu Mandelbrotovy množiny. Následně zadefinujeme periodické komponenty na základě chování orbity parametru c. Poté budeme zkoumat jejich pozici v Mandelbrotově množině a možnosti, jak vypočítat jejich středy. V poslední části kapitoly uvedeme Newtonovu metodu jako jednu z variant pro výpočet. Poskytneme také zjednodušení pro volbu počátečních hodnot pro iterační proces Newtonovy metody.

Čtvrtá kapitola se věnuje Juliovým množinám. Jako první zadefinujeme pojem normální rodiny. Poté stejně jako ve druhé kapitole stručně shrneme, co platí pro Juliovy množiny kvadratického zobrazení. Následně budeme zkoumat, které z těchto vlastností platí pro zobecněné Juliovy množiny polynomické funkce $z^d + c$, kde $d \in \mathbb{Q}$. Navíc se zaměříme na problém tzv. „skoků“ v množině, při volbě neceločíselného d. Navážeme transcendentními funkcemi ($\lambda e^z, \lambda \sin(z), \lambda \cos(z)$). Představíme také tzv. Ishikawovu iteraci, pomocí které budeme vizualizovat Juliovu množinu pro funkci $f(z) = \lambda e^z$.

Poslední dvě kapitoly popíšou webovou aplikaci. V první z nich vysvětlíme strukturu a jednotlivé části aplikace. V druhé pak představíme jednotlivé vizualizační metody, algoritmy a jejich optimalizace. Text podpoříme také úryvky zdrojového kódu a jeho vysvětlením. U každé metody poskytneme i výslednou vizualizaci a ve většině případů také diskuzi o volbě parametrů.

Diplomová práce je vysázena v systému LaTeX. Všechny algoritmy a metody

jsou naprogramovány v jazyce JavaScript a ve spojení s HTML a CSS tvoří ucelenou webovou aplikaci. Veškerý kód byl programován ve vývojářském prostředí Microsoft Visual Studio. Většina obrázků je vygenerována pomocí JavaScriptu, matematické aplikace GeoGebra nebo statistického softwaru R.

Kapitola 1

Elementární funkce komplexní proměnné

Hlavním cílem této úvodní kapitoly je čtenáři poskytnout potřebný vědomostní základ o elementárních funkcích komplexní proměnné. Vzhledem k dalšímu využití v průběhu této práce připomeneme, jak jsou funkce definovány na $\mathbb{C}$ a následně představíme základní operace s nimi. Jako první začneme exponenciální funkcí a na základě ní pak zadefinujeme i vybrané ostatní. Celá kapitola čerpá primárně ze zdrojů [2] a [16].

1.0.1 Exponenciální funkce

Exponenciální funkce e^z, kde $z = x + iy$ je definována vztahem:

$$e^z = e^x e^{iy},$$

což se s využitím Eulerova vzorce dá zapsat jako:

$$e^z = e^x(\cos(y) + i\sin(y)), \tag{1.1}$$

přičemž y je v radiánech. Toto vyjádření bude později klíčové ve vizualizaci Juliovy množiny exponenciální funkce. Z definice navíc plyne, že e^z se redukuje na běžnou exponenciální funkci používanou v $\mathbb{R}$ oboru, pokud $y = 0$.

Definice exponenciální funkce také zřejmě vyhovuje vlastnosti aditivity:

$$e^{x_1} e^{x_2} = e^{x_1 + x_2},$$

která se v komplexní analýze rozšiřuje na:

$$e^{z_1} e^{z_2} = (e^{x_1} e^{iy_1})(e^{x_2} e^{iy_2}) = (e^{x_1} e^{x_2})(e^{iy_1} e^{iy_2}).$$

Protože x_1 a x_2 jsou reálná čísla a platí $e^{iy_1} e^{iy_2} = e^{i(y_1+y_2)}$, dostáváme:

$$e^{z_1} e^{z_2} = e^{(x_1+x_2)} e^{i(y_1+y_2)} = e^{z_1+z_2}.$$

Z této vlastnosti plyne i vztah pro rozdíl:

$$\frac{e^{z_1}}{e^{z_2}} = e^{z_1 - z_2},$$

kde speciálním případem pro $e^0 = 1$ je poté:

$$\frac{1}{e^z} = e^{-z}.$$

Další důležitou vlastností exponenciální funkce e^z je její derivace:

$$\frac{d}{dz} e^z = e^z,$$

což znamená, že funkce e^z je všude komplexně diferencovatelná, tedy celá (viz 4.4).

Vlastnost absolutní hodnoty $|e^z|$ vyplývá ze vztahu:

$$|e^z| = |e^{x+iy}| = |e^x \cdot e^{iy}| = |e^x| \cdot |e^{iy}| = |e^x| \cdot 1 = e^x,$$

jelikož exponenciální funkce je pro reálné x vždy kladná a pro $|e^{iy}|$ platí:

$$|e^{iy}| = |\cos y + i \sin y| = \sqrt{\cos^2 y + \sin^2 y} = \sqrt{1} = 1.$$

Tedy absolutní hodnota, nebo též velikost, nezávisí na imaginární složce z. Naopak argument $\arg(e^z)$ je závislý pouze na imaginární složce a platí:

$$\arg(e^z) = y + 2n\pi,\ n \in \mathbb{Z}.$$

Navíc podle následujícího vztahu vidíme, že e^z je periodická s periodou $2\pi i$:

$$e^{z+2\pi i} = e^{x+i(y+2\pi)} = e^x \left(\cos(y + 2\pi) + i \sin(y + 2\pi)\right) = e^x \left(\cos y + i \sin y\right) = e^z.$$

1.0.2 Trigonometrické funkce

Konkrétně se podíváme pouze na komplexní funkce $\cos(z)$ a $\sin(z)$. Zbylé nejsou pro tuto práci tolik důležité. Definice těchto funkcí je založena na jejich exponenciálním vyjádření. Pro komplexní číslo z definujeme komplexní kosinus a sinus následujícími vztahy:

$$\cos(z) = \frac{e^{iz} + e^{-iz}}{2}, \quad \sin(z) = \frac{e^{iz} - e^{-iz}}{2i}.$$

Tyto definice vycházejí z Eulerova vzorce, jehož definice je v předchozí části o exponenciálních funkcích (viz 1.1). Díky této definici můžeme snadno vyjádřit hodnoty $\cos(z)$ a $\sin(z)$ pro komplexní argumenty.

Komplexní funkce $\cos(z)$ a $\sin(z)$ jsou periodické s periodou 2π, podobně jako v $\mathbb{R}$. To znamená, že:

$$\cos(z + 2\pi) = \cos(z), \quad \sin(z + 2\pi) = \sin(z) \quad \text{pro všechna } z \in \mathbb{C}.$$

Co se týče diferencovatelnosti, tak se jedná opět o funkce celé, tedy diferencovatelné na celém $\mathbb{C}$. Jejich derivace jsou dány vztahy:

$$\frac{d}{dz}\cos(z) = -\sin(z), \quad \frac{d}{dz}\sin(z) = \cos(z).$$

Jednou z vlastností, která je odlišná od reálného případu, je identita s hyperbolickými funkcemi $\cosh(z)$ a $\sinh(z)$[1]. Ta je dána vztahy:

$$\cos(iz) = \cosh(z), \quad \sin(iz) = i\sinh(z).$$

Důležitými vztahy, kterých bude využito při generování Juliových množin, jsou:

$$\cos(x+iy) = \cos(x)\cosh(y) - i\sin(x)\sinh(y),$$
$$\sin(x+iy) = \sin(x)\cosh(y) + i\cos(x)\sinh(y).$$

Tímto způsobem dokážeme jednoduše rozdělit komplexní $\sin a \cos$ na jejich reálnou a imaginární část, což nám výrazně ulehčí celý výpočet.

[1] Více informací ohledně hyperbolické sinu a kosinu zde [2].

Kapitola 2

Mandelbrotova množina

Kapitola se zaměřuje na pojem Mandelbrotova množina a její zobecnění z kvadratické funkce $f(z) = z^2 + c$ na polynomickou $f(z) = z^d + c$, pro $d \in \mathbb{Q}$. Využívá poznatků z [1], [15],[19] a [23]. Ve druhé části pak vycházíme z [4], [11] a také [30]. Ve výše zmíněných zdrojích jsou také uvedeny důkazy většiny vět. Pokud se tedy u věty nevyskytuje důkaz, nebylo nutné ho zde uvádět, ale v případě zájmu si jej může čtenář dohledat v poskytnuté literatuře.

Připomeňme jako první definici a základní vlastnosti Mandelbrotovy množiny funkce $f(z) = z^2 + c$. Mandelbrotova množina M je podmnožinou komplexní roviny definována jako množina všech bodů $c \in \mathbb{C}$, pro které zůstává kritická orbita funkce $f(z) = z^2 + c$ po n iteracích, kde $n \to \infty$ omezená. Formálně:

Definice 2.1. Mandelbrotova množina M je množina všech parametrů $c \in \mathbb{C}$, pro které platí

$$M = \{c \in \mathbb{C} \mid \{f^n(z_0)\}_{n=0}^{\infty} \nrightarrow \infty\} \quad z_0 = 0.$$

2.1 Základní vlastnosti

Mandelbrotova množina má zajímavé a současně i velmi důležité vlastnosti, které si nyní představíme.

2.1.1 Ohraničenost

Mandelbrotovu množinu M je možné celou ohraničit kruhem o poloměru 2 se středem v počátku. To znamená, že neexistuje bod c, který by měl velikost větší než dva a náležel Mandelbrotově množině. Matematicky $M \subset \{c \in \mathbb{C} : |c| \leq 2\}$.

Tento fakt nám mimo jiné velmi šetří čas při grafické vizualizaci. Ještě před první iterací vyloučíme velké množství bodů, které zaručeně do M patřit nebudou, a následně při každé další iteraci tuto podmínku testujeme znovu. Jelikož pokud jednou hodnota orbity bude větší jak dva, můžeme automaticky říct, že daný bod c v množině nebude. Dostáváme tak limitní poloměr divergence r_c pro Mandelbrotovu množinu, který má hodnotu 2.

2.1.2 Souvislost

Vlastnost souvislosti budeme zkoumat i pro zobecněnou Mandelbrotovu množinu. Proto bude vhodné mít ji zde připomenutou i pro kvadratický případ. Vycházíme z Douady-Hubbard věty.

Věta 2.1.1 (Douady-Hubbard)**.** *Mandelbrotova množina je souvislá.*

Důkaz. Viz [19]. □

2.1.3 Symetrie

Další vlastností, která bude dopodrobna zkoumána pro zobecněné Mandelbrotovy množiny, je symetrie, kterou můžeme pomocí věty zavést takto:

Věta 2.1.2. *Mandelbrotova množina je symetrická podle reálné osy, tedy*

$$c \in M \iff \bar{c} \in M.$$

Platnost věty je zřejmá například při pohledu na obrázek 6.10 v podkapitole 6.4 o obarvovacích metodách. Pro tuto vlastnost není nutné uvádět obrázek, protože v následujících částech textu je již k dispozici dostatek a akorát by se opakovaly.

2.1.4 Mapa Juliových množin

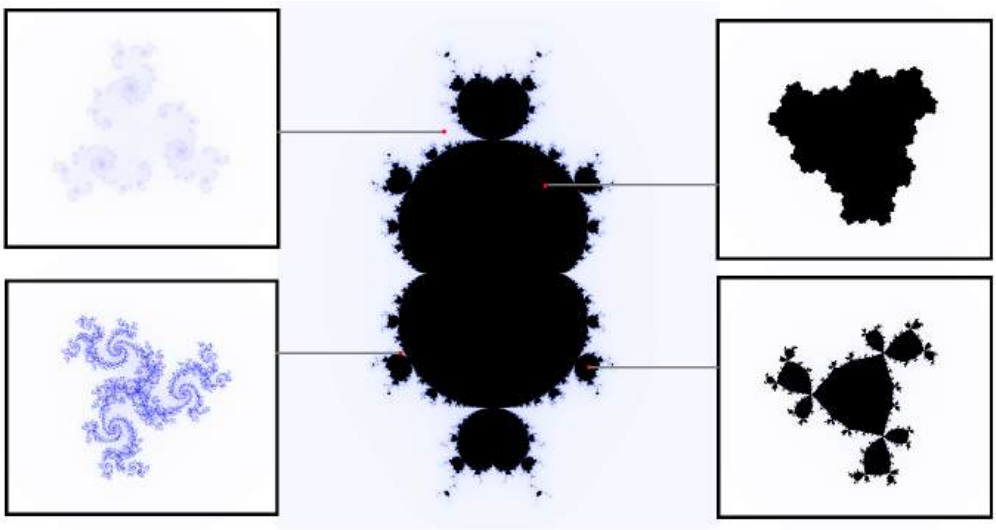
Mandelbrotova množina má úzký vztah s Juliovými množinami J, o kterých bude řeč níže v kapitole 4. Platí totiž, že Mandelbrotova množina ovlivňuje spojitost a celkově strukturu a tvar Juliových množin. Tato vlastnost je definována jako *základní dualita* a její přesnou definici najdete v kapitole o Juliových množinách. Je také zajímavé pozorovat, jak se mění složitost a tvar Juliových množin v závislosti na volbě c z M. Čím blíž jsme hranici M, tím víc vykazuje Juliova množina fraktální strukturu na své hranici. Naopak pro c hluboko uvnitř M je J strukturou jednodušší. Dokonce pro $c = 0$ je Juliova množina přesný kruh. Obecně řečeno, Mandelbrotova množina funguje jako mapa Juliových množin. Pro více informací a grafických vizualizací k této vlastnosti viz [23].

2.2 Multibrotova množina

Nyní už můžeme přejít ke zobecněnému případu Mandelbrotovy množiny. Zadefinujme si dva typy podmnožin parametrů c komplexní roviny.

Definice 2.2. *Non-escape* množina B_c je množina parametrů $c \in \mathbb{C}$, pro které platí $f_d^n(0) \not\to \infty$ při $n \to \infty$.

Definice 2.3. Množina *souvislosti* C_c je množina parametrů $c \in \mathbb{C}$, pro které je K_c souvislou množinou.



Obrázek 2.1: Juliovy množiny v závislosti na poloze c v Multibrotově množině funkce $f(z) = z^3 + c$.

Připomeňme, že K_c v této práci označuje filled Juliovu množinu, což je množina obsahující body $z_0 \in \mathbb{C}$, jejichž orbita nediverguje. Tedy,

$$K_c = \{z_0 \in \mathbb{C} \mid \{f^n(z_0)\}_{n=1}^{\infty} \not\to \infty\}.$$

Pro speciální případ, kde $d \in \mathbb{N}$ tyto množiny splývají a nazývají se Mandelbrotova množina, kterou značíme jako M, případně M_d v závislosti na kontextu. V mnoha literaturách se tento název množiny pro obecné d nepoužívá a můžeme se setkat častěji s názvem *Multibrotova* množina. Pokud tedy d bude přirozené číslo, bude Multibrotova množina opět fungovat i jako *mapa Juliových množin* s odpovídajícím exponentem d, stejně jako tomu bylo pro $d = 2$. Multibrotovu množinu jako mapu Juliových množin funkce $f(z) = z^3 + c$ můžeme vidět na obrázku 2.1.

Mnoho vlastností Mandelbrotovy množiny pro $d = 2$ lze přímo zobecnit pro Multibrotovy množiny. Stejně jako v předešlé kapitole ale musíme být velmi obezřetní při dalším zobecňování, jelikož jak jsme viděli, můžou nastat určité problémy. Například zde pro $d \notin \mathbb{N}$ nejsou množiny B_c a C_c ekvivalentní.

Pro lepší představu struktury Multibrotovy množiny viz 2.2. Jsou zde zobrazeny množiny vygenerované pro různé hodnoty exponentu d.

2.2.1 Vlastnosti

Pojďme se podívat na některé z vlastností Multibrotových množin. Vysvětlíme si je a následně většinu z nich interpretujeme pomocí grafických vizualizací.

Symetrie

Začneme s tvrzením, které jsme již v podobné formě viděli v sekci 4.3.1 kapitoly o Juliových množinách.

Tvrzení 2.2.1. *Multibrotova množina M_d, kde d je kladné celé číslo, je symetrická vzhledem k rotaci o úhel $2\pi/(d-1)$ kolem počátku. Tj. obsahuje $d-1$ identických oblastí, z nichž každá je obsažena v sektoru s úhlovou šířkou $2\pi/(d-1)$.*

Zkusme toto tvrzení nyní rozebrat. Víme, že bod c_0 je v M_d, právě když kritická orbita zůstává omezená. To je ekvivalentní s tvrzením, že existuje přitahující nebo neutrální cyklus pro $f_{c_0}(z)$. Proto musí derivace příslušné iterace f_{c_0}, označme ji $f_{c_0}^m$, vyhodnocená v bodě z_0 v cyklu mít absolutní hodnotu menší nebo rovnu 1. Platí tedy:

$$\left|\left(f_{c_0}^m\right)'(z_0)\right| = \prod_{i=0}^{m-1}\left|\left(f_{c_0}^i\right)'(z_i)\right|,$$

kde cyklus je $\{z_0, z_1 = f_{c_0}(z_0), z_2 = f_{c_0}(z_1), \ldots, z_{m-1} = f_{c_0}(z_{m-1})\}$.

Protože derivace f je tvaru $(f_{c_0})'(z) = dz^{d-1}$, násobení každé hodnoty z číslem $w_k = e^{2\pi ki/(d-1)}$ zachovává hodnotu derivace. Musíme ukázat, že bod $w_k z_0$ je periodický bod (s periodou m) pro $f_{w_k c_0}(z) = z^d + w_k c_0$. Pro pevný bod ($m = 1$) toho dosáhneme přímým výpočtem:

$$\begin{aligned}
f_{w_k c_0}(w_k z_0) &= (w_k z_0)^d + w_k c_0 \\
&= \left(e^{\frac{2\pi ki}{d-1}} z_0\right)^d + w_k c_0 \\
&= \left(e^{\frac{2\pi ki}{d-1}}\right)^{d-1+1} z_0^d + w_k c_0 \\
&= \left(e^{\frac{2\pi ki}{d-1}}\right)^{d-1} \cdot e^{\frac{2\pi ki}{d-1}} z_0^d + w_k c_0 \\
&= w_k z_0^d + w_k c_0 \\
&= w_k(z_0^d + c_0) \\
&= w_k z_0.
\end{aligned}$$

Podobně pro periodický bod s periodou m. Výpočet by se pouze zobecnil tak, že by se použila m-tá iterace $f_{w_k c_0}^m(z)$ místo samotného $f_{w_k c_0}(z)$.

Dále, jak je i z obrázku 2.2 zřejmé, Mandelbrotova, resp. Multibrotova množina je symetrická podle imaginární osy pro $d \in \mathbb{N}$. Navíc pro všechna lichá $d \in \mathbb{N}$ je Multibrotova množina symetrická i podle reálné osy, což plyne i z tvrzení 2.2.1, kde říkáme, že pro určité d množina obsahuje $d-1$ identických oblastí. Tedy pro lichá d existuje sudý počet identických oblastí, což je to, co chceme.

Symetrie pro záporný exponent d

Tvrzení 2.2.1 jde zobecnit pro všechna nenulová celočíselná d. Jelikož stejně jako u Juliových množin tyto funkce mají stejný typ symetrie jako pro $d \in \mathbb{Z}$. Stačí opět pouze nahradit exponent ve funkci $f(z)$ záporným číslem, aniž bychom porušili platnost 2.2.1. Avšak na rozdíl od kladného d, kde se vyskytovalo $d-1$ identických částí, se pro záporná d bude vyskytovat částí $|d|+1$. Dostáváme potom následující tvrzení.

Tvrzení 2.2.2. *Multibrotova množina M_d, kde d je záporné celé číslo, je symetrická vzhledem k rotaci o úhel $2\pi/(|d|+1)$ kolem počátku. Tj. obsahuje $|d|+1$ identických oblastí, z nichž každá je obsažena v sektoru s úhlovou šířkou $2\pi/(|d|+1)$.*

Pravdivost tvrzení 2.2.1 a 2.2.2 jde opět názorně vidět na obrázku 2.2, kde i pro záporné exponenty d dostáváme znovu rotační symetrii a navíc v již zmíněných případech i symetrii podle reálné nebo imaginární osy.

Symetrie pro racionální exponent d

Přejděme k dalšímu zobecnění d, konkrétně k Multibrotovým množinám s racionálním exponentem d. Následující dvě tvrzení vychází z [15]. Autoři zde využívají rozložení exponentu $d = \mu + \varepsilon$, kde $\mu \in \mathbb{Z}$ a pro $\varepsilon \in \mathbb{Q}$ platí $0 < \varepsilon < 1$. Symbol $\Omega(\lambda)$, kde λ je množina, označuje úhlovou velikost oblasti obsahující λ. Dále autoři také využívají následujících pojmenování, která označují vždy určité oblasti Multibrotovy množiny.

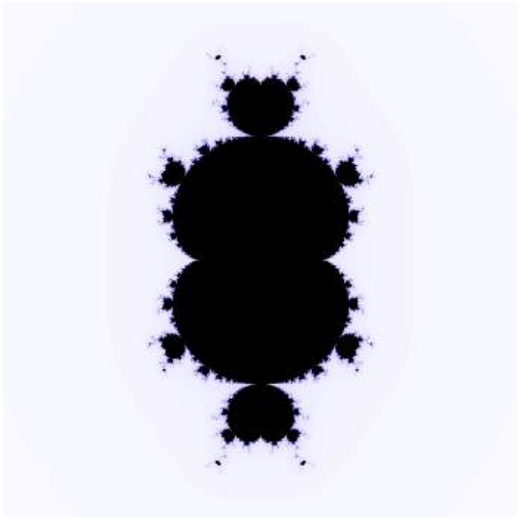
- **Hlavní lalok** (*major lobe*) - označován λ_s představuje velkou dominantní část množiny podobnou ostatním velkým částem množiny.
- **Zárodečný lalok** (*embryonic lobe*) - označován λ_e je menší část, která má menší úhlovou velikost než hlavní lalok. Vypadá jako zploštělá verze hlavního laloku.
- **Kompozitní hlavní lalok** (*composite major lobe*) - označován λ_k je část, která má větší úhlovou velikost než hlavní lalok. Vypadá jako dva hlavní laloky sloučené dohromady.

Nyní už můžeme bez problémů uvést následující tvrzení týkající se symetrie Multibrotovy množiny v závislosti na paritě d.

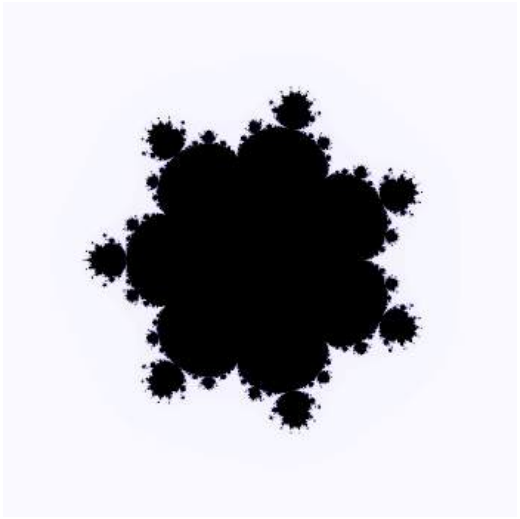
Tvrzení 2.2.3. *Transformace $z \mapsto z^d + c$, pro $d = \mu + \varepsilon$ a μ liché kladné celé číslo, se zadanou konstantou z_0, vytváří fraktální obraz v komplexní rovině parametrů c složený z Λ soběpodobných hlavních laloků, kde $\Lambda = (\mu - 1)\lambda_s$, s $\Omega(\lambda_s) = \frac{2\pi}{\mu+\varepsilon-1}$, a jednoho zárodečného laloku λ_e s $\Omega(\lambda_e) = \frac{2\pi\varepsilon}{\mu+\varepsilon-1}$. Obraz je symetrický vzhledem k reálné ose a λ_e se nachází v levé polovině komplexní roviny parametrů c na záporné reálné ose.*

Jinými slovy, Multibrotova množina pro liché μ má $\mu - 1$ velkých stejných částí a jednu menší nekompletní část, která se nachází v oblasti záporné části reálné osy.

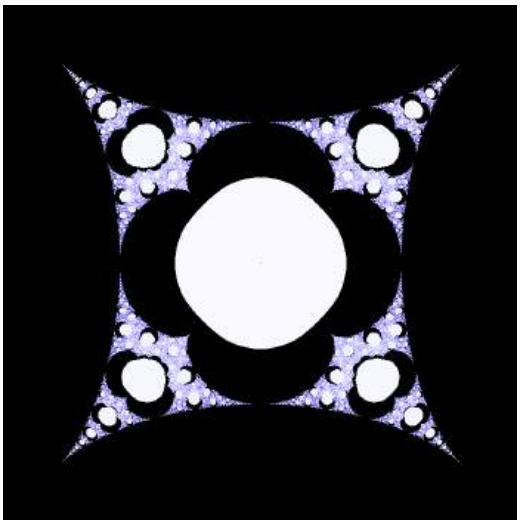
Tvrzení 2.2.4. *Transformace $z \mapsto z^d + c$, pro $d = \mu + \varepsilon$ a μ sudé kladné celé číslo, se zadanou konstantou z_0, vytváří fraktální obraz v komplexní rovině parametrů c složený z Λ soběpodobných hlavních laloků, kde $\Lambda = (\mu - 2)\lambda_s$, s $\Omega(\lambda_s) = \frac{2\pi}{\mu+\varepsilon-1}$, a jednoho kompozitního hlavního laloku λ_k s $\Omega(\lambda_k) = \frac{2\pi(1+\varepsilon)}{\mu+\varepsilon-1}$. Obraz je symetrický vzhledem k reálné ose a λ_k se nachází v levé polovině komplexní roviny parametrů c na záporné reálné ose.*



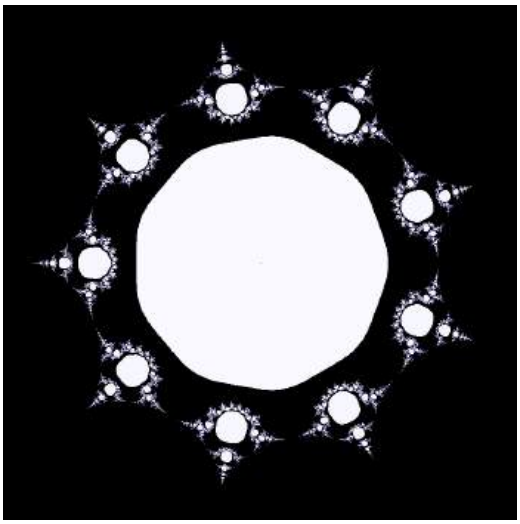
(a) M pro $d = 3$.



(b) M pro $d = 8$.

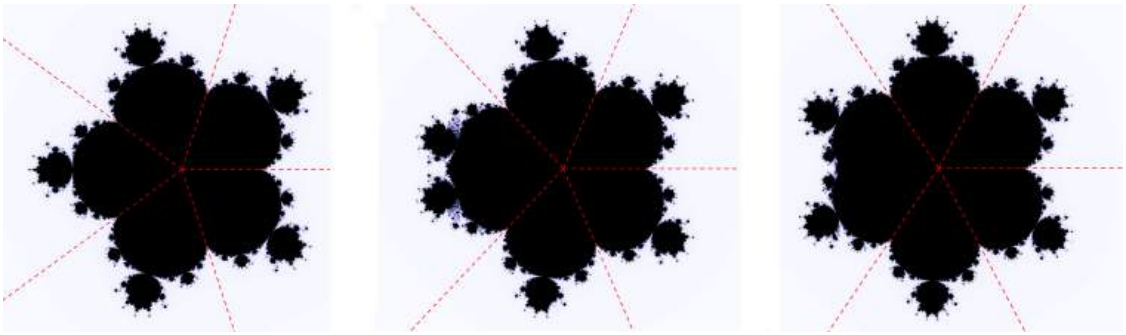


(c) M pro $d = -3$.

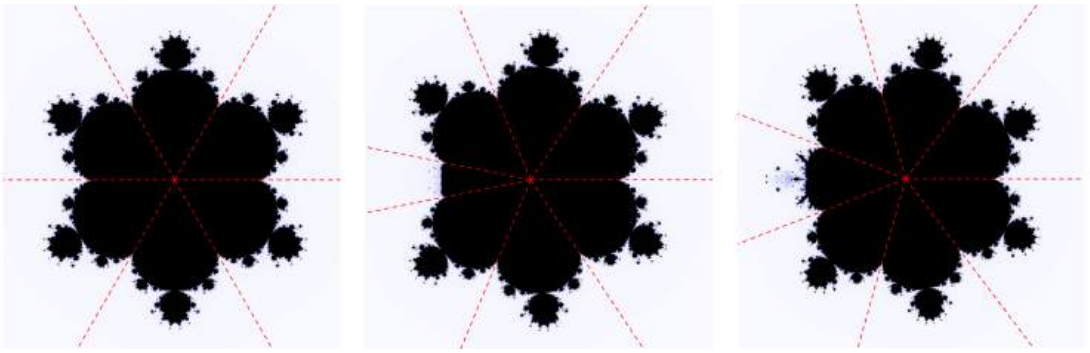


(d) M pro $d = -8$.

Obrázek 2.2: Mandelbrotova množina $z^d + c$ pro různé hodnoty $d \in \mathbb{Z}$.



(a) Postupná tvorba kompozitního hlavního laloku λ_k pro $\mu = 6$ a $\varepsilon = 0; 0{,}4$ a $0{,}8$.



(b) Postupná tvorba zárodečného laloku λ_e pro $\mu = 7$ a $\varepsilon = 0; 0{,}4$ a $0{,}8$.

Obrázek 2.3: Multibrotova množina $z^{\mu+\varepsilon} + c$ pro pevné $\mu \in \mathbb{Z}$ a postupně rostoucí hodnoty $\varepsilon \in \mathbb{Q}$.

Druhé tvrzení je velmi podobné prvnímu. Multibrotova množina je stejně jako v předešlém případě symetrická podle reálné osy. Pro μ obsahuje $\mu - 2$ shodných oblastí a jednu větší oblast, která se opět nachází v místech záporné části reálné osy.

Úplně zjednodušeně tato tvrzení můžeme chápat následovně. Uvědomme si, že pro $d \in \mathbb{Z}$ máme $\mu - 1$, jelikož $\varepsilon = 0$. A nyní je rozdíl v tom, jestli d je sudé, nebo liché. Pro sudé d leží jeden z $\mu - 1$ hlavních laloků přímo na záporné části reálné osy. Tedy pokud nyní připustíme, aby $\varepsilon > 0$, pak se tento lalok na záporné části reálné osy začne symetricky rozšiřovat a tvořit tak větší oblast, než je hlavní lalok. Naopak pro d liché je Multibrotova množina nejen symetrická podle reálné osy, ale také podle počátku. Tedy na záporné části reálné osy se nachází spoj mezi laloky. Takže pokud se opět připustí $\varepsilon > 0$, oblast, která se zde začne symetricky vytvářet, se bude zvětšovat od nulové velikosti až pro $\varepsilon \to 1$ k velikosti blížící se velikosti hlavních laloků, ale nikdy této velikosti nedosáhne. Po zkoumání Multibrotovy množiny pro d záporné navíc můžeme předešlá tvrzení zobecnit na celé $\mathbb{Q}$.

Tuto vlastnost můžeme pozorovat lépe na obrázku 2.3. Nalezneme zde vývoj tvorby jak kompozitního hlavního laloku v části 2.3a, tak vývoj tvorby zárodečného laloku v části 2.3b, pro postupně se zvětšující hodnotu ε. Jednotlivé laloky jsou ohraničeny červenými osami, aby byl názorněji vidět rozdíl ve velikosti hlavních laloků oproti kompozitnímu hlavnímu laloku, resp. zárodečného laloku.

Ve skutečnosti jsou tyto různé struktury laloků způsobeny větvovým řezem, o

kterém bude řeč v sekci 4.3.1.

Věta 2.2.1. *Multibrotova množina funkce $f(z) = z^d + c$ je invariantní vůči rotaci o úhel $\frac{2\pi}{d-1}$ kolem počátku, ale je omezena větvením.*

Jak vidíme, tak tato věta mimo jiné odpovídá i 2.2.3 a 2.2.4. Než si potvrdíme platnost předchozí věty, bude dobré uvést ještě lemma, které nám následně pomůže.

Lemma 2.2.1. *Platí $f_{w_k c_0}(w_k z) = w_k f_{c_0}(z)$, kde $w_k = e^{\frac{2\pi k i}{d-1}}$.*

Pravdivost lemmatu ukážeme přímým výpočtem takto:

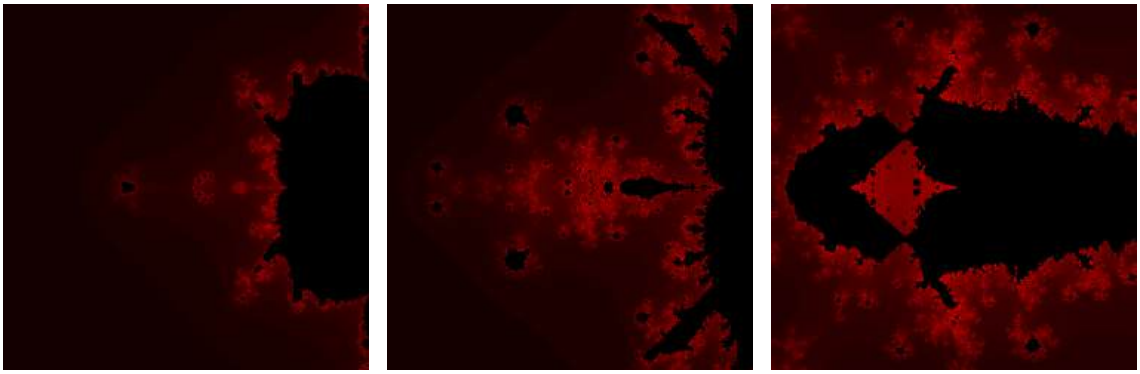
$$\begin{aligned} f_{w_k c_0}(w_k z) &= (w_k z)^d + w_k c_0 \\ &= \left(e^{\frac{2\pi k i}{d-1}} z\right)^d + w_k c_0 \\ &= \left(e^{\frac{2\pi k i}{d-1}}\right)^{d-1+1} z^d + w_k c_0 \\ &= \left(e^{\frac{2\pi k i}{d-1}}\right)^{d-1} \left(e^{\frac{2\pi k i}{d-1}}\right) z^d + w_k c_0 \\ &= w_k z^d + w_k c_0 \\ &= w_k (z^d + c_0) \\ &= w_k f_{c_0}(z). \end{aligned}$$

Nyní už zpět k větě 2.2.1. Bod c_0 náleží Mandelbrotově množině, pokud jeho kritická orbita funkce $f_{c_0}(z)$ je omezená. Zvolme tedy bod c_0 a sledujme kritickou orbitu funkce $f_{w_k c_0}(z)$, kde w_k je definováno v lemmatu 2.2.1. Jistě platí $f_{c_0}(0) = c_0$ a $f_{w_k c_0}(w_k \cdot 0) = w_k c_0 = w_k f_{c_0}(0)$, tedy $\left|f^1_{w_k c_0}(0)\right| = \left|f^1_{c_0}(0)\right|$.

Podle lemmatu 2.2.1 je při každé iteraci funkcí $f_{c_0}(z)$ a $f_{w_k c_0}(z)$ zachována rovnost velikostí jejich hodnot. Tedy obecně pak při každém kroku v iteračním cyklu platí $\left|f^n_{w_k c_0}(0)\right| = \left|f^n_{c_0}(0)\right|$. A to je rovnost, které jsme chtěli dosáhnout. Jednoduše tím říkáme, že pokud Multibrotovu množinu otočíme o úhel $\frac{2\pi}{d-1}$, na výsledku to nic nezmění. Kdybychom původní a tu otočenou o daný úhel překryli přes sebe, vznikla by opět původní množina.

Souvislost

Na symetrii Multibrotovy množiny v případě $d \in \mathbb{Q}$ navazují vlastnosti souvislosti, resp. nesouvislosti v oblasti záporné části reálné osy. Vzhledem k tomu, že Multibrotova množina „vyrůstá“ symetricky do obou směrů od reálné osy, může se stát, že v určitém případě se zde budou nacházet nesouvislosti. Například můžou z reálné osy začít „vyrůstat“ antény Multibrotovy množiny, které vytvoří izolovaný „ostrůvek“, jak můžeme pozorovat na obrázku 2.4, čímž přestane platit vlastnost jednoduché souvislosti množiny, která pro celočíselné exponenty d platí (viz [8]). Tyto anomálie jsou samozřejmě možné pouze v případě pro neceločíselný exponent d. Pokud $z \in \mathbb{Z}$, pak se množiny chovají „hezky“ a podle předpokladů.



(a) M pro $d = 3{,}75$. **(b)** M pro $d = 7{,}8$. **(c)** M pro $d = 5{,}853$.

Obrázek 2.4: Výřez na nesouvislé části Multibrotovy množiny v oblasti záporné části reálné osy pro různá d.

2.2.2 Bifurkační body hyperbolických komponent

V poslední sekci této kapitoly se zaměříme na bifurkační body hyperbolických komponent Mandelbrotovy množiny pro $d = 2$. Předně ale zadefinujme pojem *komplexní odmocniny z jedné*. Využijeme k ní Moivrovu větu, která říká, jak vypadá komplexní číslo v exponenciálním tvaru.

Věta 2.2.2 (Moivrova věta)**.** *Nechť $z = re^{i\theta}$ je komplexní číslo v exponenciálním tvaru, kde r je jeho absolutní hodnota a θ je argument. Pro libovolné celé číslo n platí*

$$z^n = r^n e^{in\theta}.$$

A následně už zadefinujeme pojem komplexní odmocniny z jedné. Komplexní n-té odmocniny z jedné jsou čísla, která při umocnění na určitou celou kladnou mocninu n dávají výsledek 1. Formálně tedy platí:

$$z^n = 1, \quad z \in \mathbb{C}, \quad n \in \mathbb{N}.$$

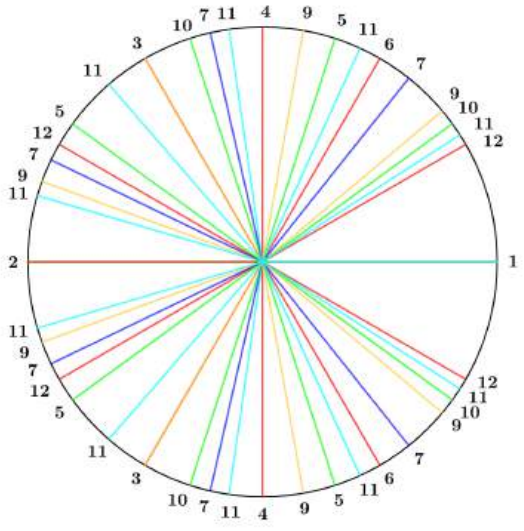
Také zle n-tou odmocniny z jedné definovat jako číslo z, které splňuje rovnici $z^n - 1 = 0$.

Důležitým faktem je, že komplexní odmocniny z jedné jsou rovnoměrně rozmístěné na jednotkové kružnici, tj.

$$z_l = e^{i\frac{2l\pi}{n}}, \quad l = 0, 1, 2, \ldots, n-1.$$

Z toho plyne, že n-té odmocniny z jedné tvoří n vrcholů pravidelného n-úhelníku vepsaného do jednotkové kružnice. Rovnoměrné rozmístění odmocnin z jedné můžeme pozorovat na obrázku 2.5. Pro lepší orientaci jsou jednotlivé odmocniny odlišeny barevně.

Všimněme si, že některé různě barevné úsečky vedoucí k požadované odmocnině se vzájemně překrývají. Obecně totiž platí, že do hodnot řešení n-tých odmocnin z jedné patří i hodnoty k-tých odmocnin z jedné, kde $k|n$ a zároveň



Obrázek 2.5: Vybrané komplexní odmocniny z jedné a jejich pozice na jednotkovém kruhu.

$k < n$. Například pro šestou komplexní odmocninu z jedné platí, že v množině řešení jsou i třetí, druhé a první odmocniny z jedné. Tento fakt vychází přímo z definice. Totiž třeba pro $l = 2$ se hodnota šesté odmocniny z jedné $e^{i\frac{2\cdot 2\pi}{6}}$ rovná hodnotě třetí odmocniny z jedné pro $l = 1$, tedy $e^{i\frac{2\cdot 1\pi}{3}}$. Po úpravě dostaneme v obou případech hodnotu $e^{i\frac{2\cdot 1\pi}{3}}$. Tento problém řeší následující pojem, který si zadefinujeme.

Primitivní odmocniny z jedné

Komplexní n-tá odmocnina z jedné z je označena jako *primitivní* n-tá odmocnina z jedné, pokud platí, že $z^k \neq 1$ pro všechna $k < n$. Formálně:

$$z^n = 1 \quad \text{a současně} \quad z^k \neq 1, \quad \forall k \in \{1, 2, \ldots, n-1\}.$$

Příklad 1. Pro $n = 4$ jsou čtvrté mocniny z jedné:

$$z_l = e^{i\frac{2l\pi}{4}}, \quad l = 0, 1, 2, 3.$$

Tyto kořeny jsou:

$$z_0 = 1, \quad z_1 = i, \quad z_2 = -1, \quad z_3 = -i.$$

Geometricky tvoří vrcholy čtverce na jednotkové kružnici.

Nyní již uvedeme samotnou Douady-Hubbard-Sullivan větu, která je klíčová při hledání bifurkačních bodů Mandelbrotovy množiny.

Věta 2.2.3. *Pokud W je hyperbolická komponenta „vnitřku" Mandelbrotovy množiny M, pak multiplikátor ρ[1] přitažlivého cyklu $f(z) = z^2 + c$, kde $c \in W$ zobrazuje W konformně*

[1] V našem případě definujeme multiplikátor ρ jednoduše jako derivaci $f'(z)$.

na otevřený jednotkový kruh Δ. To lze rozšířit spojitě na hranici ∂W, což pak zobrazuje $\overline{W}$ homeomorfně na uzávěr kruhu $\overline{\Delta}$.

Důkaz. viz [4]. □

Spojení „konformní zobrazení“ znamená, že zobrazení je spojité a zachovává úhly. Spojením „homeomorfní zobrazení“ se myslí vzájemně jednoznačné a oboustranně spojité zobrazení.

Větu 2.2.3 můžeme využít společně s n-tými odmocninami z jedné. Můžeme tvrdit, že všechny primitivní odmocniny z jedné jsou zobrazeny v odpovídajícím pořadí na hranici každé hyperbolické komponenty Mandelbrotovy množiny jako daná hustá posloupnost bifurkačních bodů.

Po nalezení odpovídajících odmocnin z jedné využijeme transformaci na kardioid, čímž zapříčiníme, že se body n-té odmocniny z jedné na jednotkovém kruhu pomocí transformace zobrazí na hranici kardiodu. Navíc tyto body budou odpovídat přímo bifurkačním bodům hyperbolických komponent Mandelbrotovy množiny pro dané n.

Z hlediska teorie transformaci kruhu na kardioid, tedy odmocnin z jedné na bifurkační body provádíme následovně. Každá n-tá odmocnina má podle definice určitý úhel θ na jednotkovém kruhu. Pomocí tohoto úhlu θ spočítáme bifurkační body odpovídajících daným komplexním odmocninám jedné. Vztahy pro tento výpočet jsou následující:

$$z_x(\theta) = \frac{1}{2}\cos(\theta) - \frac{1}{4}\cos(2\theta),$$
$$z_y(\theta) = \frac{1}{2}\sin(\theta) - \frac{1}{4}\sin(2\theta),$$

kde z_x a z_y jsou reálná a imaginární část z. Podobným způsobem lze transformaci provést i pro oblast c hodnot s periodou dva. V tomto případě to bude ještě jednodušší, jelikož stačí pouze posunout střed kruhu do -1 a změnit poloměr na $\frac{1}{4}$. Takto jsme schopni opět transformovat komplexní odmocniny z jedné na bifurkační body. Jen s tím rozdílem, že dané bifurkační body pro n-té odmocniny z jedné odpovídají bifurkačním bodům oblastí c hodnot s periodou $2n$. Tedy pokud provedeme transformaci z jednotkového kruhu na hranici oblasti s periodou dva, pak například pro $n = 2$ najdeme ve skutečnosti bifurkační body oblastí c hodnot s periodou čtyři a podobně. Tuto transformaci provedeme pomocí následujících vztahů:

$$z_x(\theta) = -1 + \frac{1}{4}\cos(\theta),$$
$$z_y(\theta) = \frac{1}{4}\sin(\theta).$$

Kapitola 3

Hyperbolické komponenty

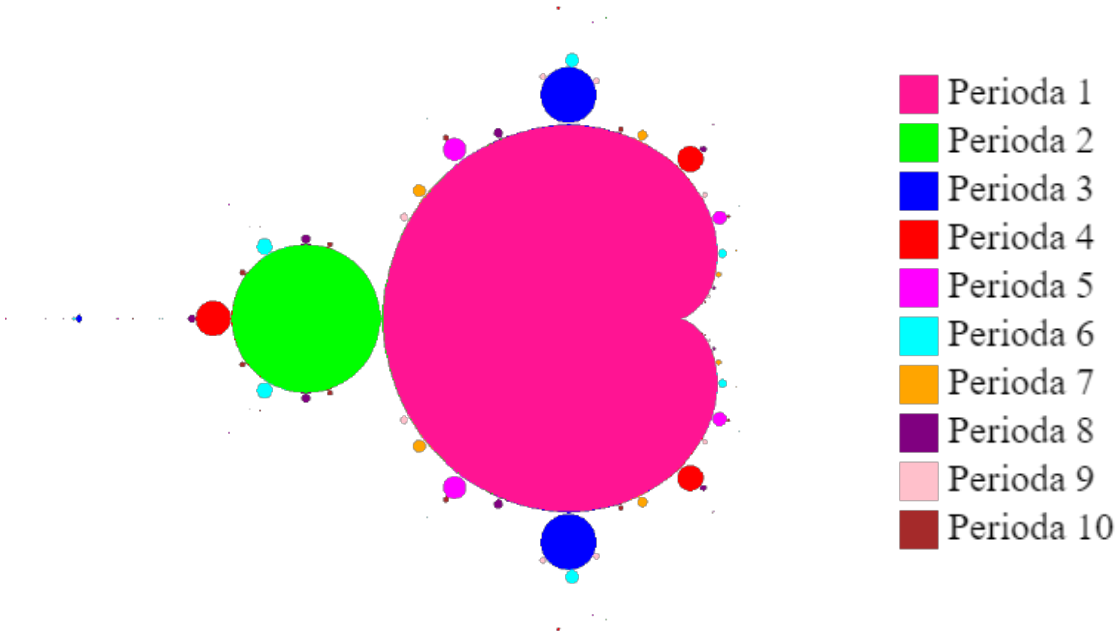
V této kapitole se zaměříme na problematiku hyperbolických komponent a jakým způsobem je hledat. Budeme čerpat z literatury [4], [22], [25], [30] a [35]. V sekci zabývající se Newtonovou metodou pak využijeme [32] a [34]. V textu bude používán termín hyperbolické komponenty (z anglického *hyperbolic components*). V různých literaturách se také můžeme setkat s termínem periodické bubliny (*periodic bubbles*).

3.1 Geometrie Mandelbrotovy množiny

Na začátek je vhodné stručně připomenout geometrii Mandelbrotovy množiny:

- Velká centrální část **kardioid**, obsahující parametry c, pro které iterační posloupnost $z_{n+1} = z_n^2 + c$ konverguje (perioda kritické orbity je jedna).
- K hlavnímu kardioidu je připojeno spočetně nekonečné množství **hyperbolických komponent**, přičemž každá odpovídá podmnožině parametrů c, které generují kritické orbity se stejnou periodou.
  - Největší z těchto komponent se nachází vlevo od kardioidu a obsahuje takové parametry c, pro které má kritická orbita periodu dva.
  - Další velké komponenty se nacházejí přímo nad a pod kardioidem obsahují parametry c, pro které má kritická orbita periodu tři.
- Kolem množiny se také nachází nekonečné množství **satelitů**, což jsou menší kopie hlavního těla množiny.

Obrázek 3.1 potvrzuje předchozí popis struktury Mandelbrotovy množiny. Jsou zde znázorněny podmnožiny parametrů c ze $\mathbb{C}$, pro které kritická orbita konverguje k periodě 1 až 10. Pro jednotlivé periody jsou podmnožiny pro lepší názornost odlišeny různými barvami. Vidíme, že už jen pro prvních deset period připomíná celé seskupení komponent Mandelbrotovu množinu a kdybychom se neomezili pouze na maximální velikost periody 10, vygenerovali bychom tímto způsobem celou Mandelbrotovu množinu, která je celá složena z těchto komponent pro periodu $n \to \infty$.



Obrázek 3.1: Hyperbolické komponenty parametrů c pro $n = 1,\ 2,\ 3, \ldots 10$.

3.2 Studie hyperbolických komponent

Cílem této podkapitoly je získat analytickou metodu, která nám pomůže rozhodnout, který z parametrů c v $\mathbb{C}$ patří do Mandelbrotovy množiny, což také povede k určení, kde se nacházejí jednotlivé hyperbolické komponenty.

Pro větší přehlednost v textu a zachování jednotnosti budeme využívat zápisu

$$z_{n+1} = f_c(z_n),$$

s počáteční hodnotou $z_0 = 0$, kde

$$f_c(z) = z^2 + c.$$

Dále pak tedy platí

$$z_k = f^k(z_0), \tag{3.1}$$

což nám generuje posloupnost hodnot $z_0, z_1, z_2, \ldots z_n$. Připomeňme také, že aby vygenerovaná posloupnost konvergovala k cyklu s periodou p, musí po určitém počtu iterací l platit $z_l = f^p(z_l)$. Pro tuto rovnost mohou nastat tři situace v závislosti na počátečním bodu iterace, který nazýváme:

1. **Preperiodický**, pokud $l > 0$.
2. **Periodický**, pokud $l = 0$.
3. **Pevný**, pokud $p = 1$.

Je také zřejmé, že jakmile rovnost začne platit jednou, bude platit pro nekonečně mnoho iteračních kroků.

Před další definicí ještě zadefinujme pojem *kvadratické konvergence*, která se nám bude nadále hodit.

Definice 3.1. Nechť $z_0, z_1, z_2, \dots$ je posloupnost, která konverguje k z^*, a nechť $e_k = z_k - z^*$. Když existuje číslo p a konstanta $C \neq 0$ taková, že

$$\lim_{k\to\infty} \frac{|e_{k+1}|}{|e_k|^p} = C,$$

pak p se nazývá řád konvergence posloupnosti a C je chybová konstanta. Speciálně říkáme, že:

- konvergence je **lineární**, když $p = 1$ a $C < 1$,
- konvergence je **superlineární**, když $p > 1$,
- konvergence je **kvadratická**, když $p = 2$.

Řekneme, že daná metoda je řádu p, jestliže všechny konvergentní posloupnosti získané touto metodou mají řád konvergence větší nebo rovný p, a nejméně jedna z těchto posloupností má řád konvergence přesně p.

Dále můžeme využít podmínku, která je dána větou o pevném bodě, aby k tomuto bodu iterační posloupnost konvergovala.

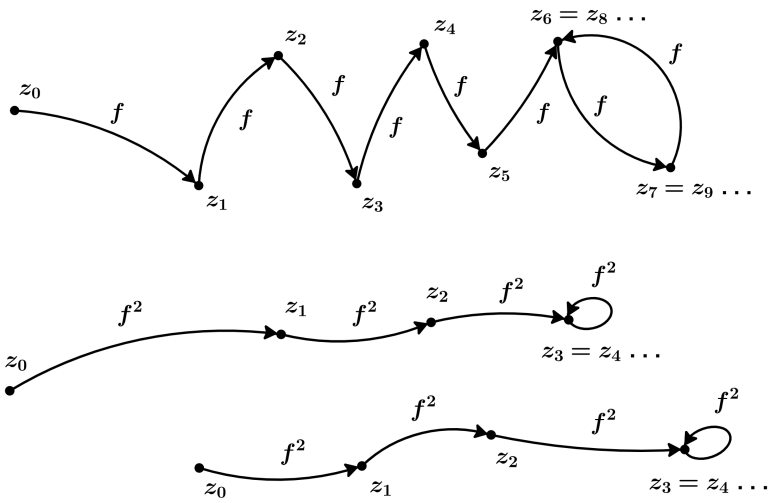
Definice 3.2. Nechť $z^* \in \mathbb{C}$ je pevný bod analytické funkce f. Pak z^* je:

1. **Super-přitahující**, jestliže $f'(z^*) = 0$ a posloupnost hodnot $z_0, z_1, \dots$ má alespoň kvadratickou konvergenci.
2. **Přitahující**, jestliže $0 < |f'(z^*)| < 1$.
3. **Odpuzující**, jestliže $|f'(z^*)| > 1$.
4. **Racionálně indiferentní**, jestliže $f'(z^*)$ je komplexní odmocnina z jedné.
5. **Iracionálně indiferentní**, jestliže $|f'(z^*)| = 1$, ale $f'(z^*)$ není komplexní odmocnina z jedné.

Nás bude konkrétně nejvíce zajímat vlastnost 2. Chceme-li ji využít při hledání parametru c, který způsobuje periodické chování orbity, využijeme následujícího faktu. Mějme iterační posloupnost s periodou n tvaru

$$\dots, z_k, f(z_k), f^2(z_k), f^3(z_k), \dots f^n(z_k) = z_k, \dots$$

Pak posloupnost vytvořená použitím transformace f n–krát na každou hodnotu z_k iterační posloupnosti generuje konstantní posloupnost této hodnoty, tedy jinými slovy konvergentní orbitu.



Obrázek 3.2: V horní části schematicky zakreslena preperiodická orbita transformace f pro daný parametr c. V dolní části pak pro stejné c orbita transformace f^2.

Podobně pro preperiodickou orbitu s konečnou periodou n, kde pouze „přibude“ několik hodnot počátečních iterací, než orbita dokonvertuje k cyklu. Zřejmě s využitím transformace f můžeme z preperiodické orbity sestrojit n konvergentních posloupností, které konvergují ke každému z bodů původního cyklu. Obecně se tak tyto body stávají pevnými body, což je přesně výsledek, který chceme dosáhnout pro využití definice 3.2. Pokud tedy pomocí této definice najdeme oblast, pro kterou konverguje kterákoliv z těchto posloupností generovaná f^n, získáme pak i oblasti, pro které transformace f generuje periodické orbity s periodou n. Tuto vlastnost a také vlastnost z předchozího odstavce schematicky ilustrujeme na obrázku 3.2.

3.2.1 Vztah chovaní orbity parametru c

Pro zjištění, zda-li orbita z_k konverguje k cyklu s periodou n, upravíme podmínku 2. z 3.2 takto:

$$|f^{n\prime}(x)| \leq 1. \tag{3.2}$$

Připustili jsme rovnost, tedy i body na hranici komponent budou vyhovovat podmínce a zároveň jsme se omezili pouze na pravou část nerovnosti, což je pro naše účely postačující.

Abychom byli schopni tuto nerovnost ověřovat, potřebujeme získat derivaci f^n. Jak již víme $f^n(z) = f(f^{n-1}(z))$. Můžeme proto využít vzorec pro derivaci složené funkce tvaru:

$$f^{n\prime}(z) = f'(f^{n-1}(z)) \cdot f^{n-1\prime}(z). \tag{3.3}$$

Pro $f(z) = z^2 + c$ je derivace podle z $f'(z) = 2z$. Dosazením do (3.3) a úpravou dostáváme tedy

$$f^{n\prime}(z) = 2 \cdot f^{n-1}(z) \cdot f^{n-1\prime}(z).$$

Předchozí rovnici také můžeme rozepsat následovně:

$$\begin{aligned} f^{n\prime}(z) &= 2 \cdot f^{n-1}(z) \cdot f^{n-1\prime}(z) \\ f^{n\prime}(z) &= 2 \cdot f^{n-1}(z) \cdot 2 \cdot f^{n-2}(z) \cdot f^{n-2\prime}(z) \\ &\vdots \\ f^{n\prime}(z) &= 2^n \cdot f^{n-1}(z) \cdot f^{n-2}(z) \cdot f^{n-3}(z) \cdots f(z) \cdot z \end{aligned}$$

Pro splnění (3.2) musí platit, že z je bod z komponenty konvergující k cyklu s periodou n a tedy pak $f^{n-1}(z), f^{n-2}(z) \ldots$ nejsou nic jiného, než hodnoty periodické orbity s periodou n. S využitím vztahu (3.1) můžeme pak předchozí rovnost zjednodušeně zapsat jako

$$f^{n\prime}(z) = 2^n \cdot \prod_{i=1}^{n} z_i.$$

Aplikováním (3.2) pak dostaneme

$$\left| 2^n \cdot \prod_{i=1}^{n} z_i \right| \leq 1. \tag{3.4}$$

Výsledek už pouze upravíme do vhodnějšího tvaru

$$\left| \prod_{i=1}^{n} z_i \right| \leq \frac{1}{2^n}. \tag{3.5}$$

Jen doplňme, že tedy pro všechny parametry c, které budou splňovat tuto nerovnost, platí, že se nacházejí v určité komponentě Mandelbrotovy množiny. A naopak ty, které tuto rovnost nesplní, negenerují periodickou orbitu a nebudou tak vůbec náležet Mandelbrotově množině.

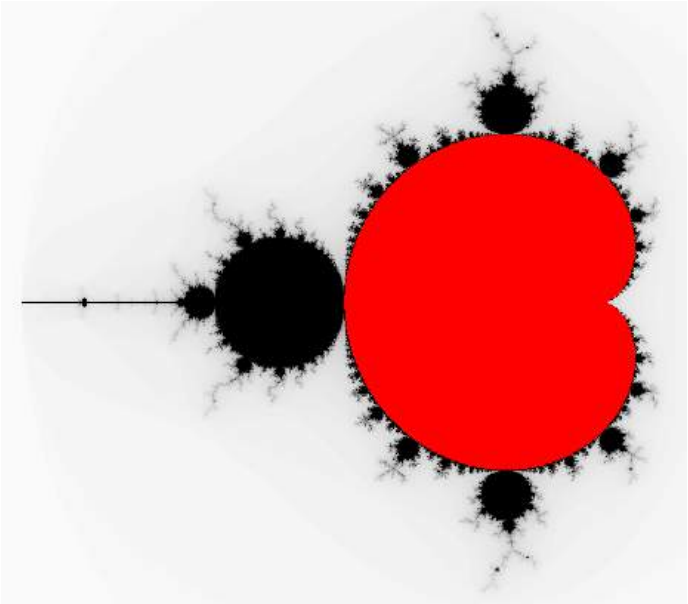
3.2.2 Pozice hyperbolické komponenty

V předchozí podkapitole jsme odvodili vztah, podle kterého jsme schopni určit chování orbity pro parametr c. V této sekci se zaměříme na polohu komponenty v $\mathbb{C}$ tvořených z c, které vykazují stejné chování, tedy mají stejnou periodu orbity. Uvedeme si příklad pouze pro $n = 1, 2$ a 3, jelikož výpočet vyšších period je výrazně časově náročnější a pro ilustraci logiky výpočtu to již není nutné uvádět. Ve výpočtech je pro jednoznačnost kořenů používán index u c. Tedy například pro tři kořeny polynomu dostaneme pro c kořeny značené jako c_1, c_2, c_3.

Komponenta pro $n = 1$

Po (3.5) aplikaci okamžitě vidíme, že

$$|z_1| \leq \frac{1}{2}. \tag{3.6}$$



Obrázek 3.3: Červeně zvýrazněná hyperbolická komponenta pro $n = 1$.

To nám říká, že pevný bod pro orbity generované c z komponenty s periodou 1 má modulo menší nebo rovno $\frac{1}{2}$. Nyní bude potřeba spočítat hodnoty c. V případě $n = 1$ to nebude vůbec složité. Uvědomíme-li si, že $z_{n+1} = z_n$. To znamená, že $z_n = z_n^2 + c$ a to není závislé na čísle iterace. Můžeme tedy číslo iterace zanedbat a dostáváme rovnost

$$c_1 = z - z^2. \tag{3.7}$$

Pro zjednodušení výpočtu nejprve uvažujme v (3.6) pouze rovnost, tedy c na hranici komponenty, pro které tedy platí $|z| = \frac{1}{2}$. Převodem této rovnosti do exponenciálního tvaru[1] pak dostaneme $z = \frac{1}{2}e^{i\theta}$. Dosazením do (3.7) získáme

$$c_1 = \frac{1}{2}e^{i\theta} - \frac{1}{4}e^{i2\theta} \quad \text{pro} \quad \theta \in [0, 2\pi)\,. \tag{3.8}$$

Takto jsme získali parametrický předpis křivky jako funkci θ. Vykreslíme-li si body splňující tuto rovnost pro všechny θ, získáme přesný tvar kardioidu. Zobecněním (3.8) na nerovnost pak dostaneme celou oblast parametrů c, pro které má orbita periodu jedna. Výsledek ilustrujme na obrázku 3.3.

Komponenta pro $n = 2$

Budeme pokračovat obdobně jak v předchozím případě. Hledáme oblast parametrů c, pro které bude orbita konvergovat k cyklu délky 2. Využijeme jednu z vlastností komplexních čísel $|a \cdot b| = |a| \cdot |b|$. Opět s využitím výsledku (3.5) z

[1] $z = |z|e^{i\theta}$, kde θ je $\arg(z)$. Pro více podrobností viz [2] nebo [16].

předchozí kapitoly dostáváme

$$|z_1 \cdot z_2| \leq \frac{1}{2^2},$$
$$|z_1| \cdot |z_2| \leq \frac{1}{4}.$$

Znovu je potřeba si uvědomit, že konkrétně pro tento případ bude platit $z_{n+2} = z_n$. To tedy znamená, že

$$z = f(f(z))$$
$$z = (z^2 + c)^2 + c$$
$$z = z^4 + 2z^2c + c^2 + c$$

Úpravou na tvar kvadratické rovnice dostáváme

$$c^2 + c(1 + 2z^2) + (z^4 - z) = 0.$$

Běžným řešením této rovnice pro c pomocí diskriminantu a upravením na co nejjednodušší tvar dostáváme

$$c_1 = z - z^2, \tag{3.9}$$
$$c_2 = -1 - z - z^2. \tag{3.10}$$

Jak si lze všimnout, tak c_1 koresponduje s řešením pro $n = 1$. Nastává zde problém, že orbita s periodou jedna je periodická i s jakoukoliv další periodou. Dá se totiž lehce ukázat pomocí vlastností ze sekce 3.2, že pokud aplikujeme transformaci f^3 na bod generující orbitu s periodou jedna, bude vyhovovat rovnosti $z = f^3(z)$ stejně dobře jako bod generující orbitu s periodou tři. Navíc tohle neplatí pouze pro periodu jedna, ale také obecně pro všechny dělitele periody n. Tento fakt ale rozebereme podrobně později v 3.2.3, nyní zpět k řešení. Výsledek pro c_1 proto nebudeme do řešení zahrnovat a zaměříme se pouze na c_2, které odpovídá pozici komponenty s periodou dva.

Nyní z rovnice pro c_2 spočítáme z pomocí diskriminantu, z kterého nám vyjdou dvě možná řešení pro z. Přesněji tedy z_1 a z_2, které dosadíme do (3.9):

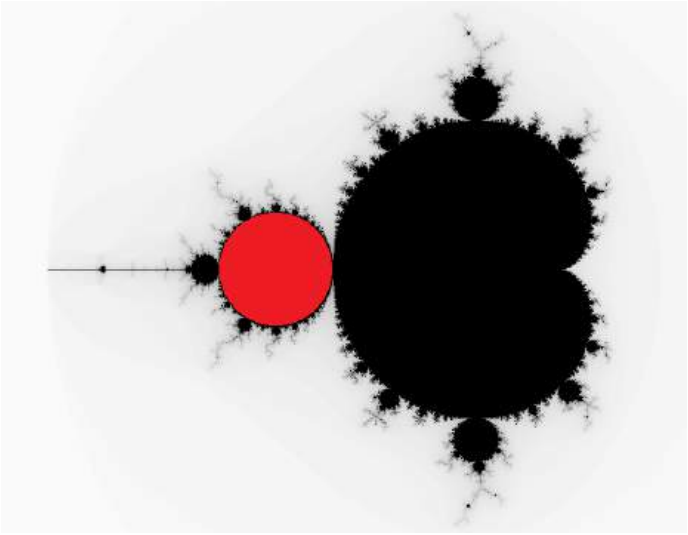
$$\left| \frac{-1 + \sqrt{-3 - 4c_2}}{2} \cdot \frac{-1 - \sqrt{-3 - 4c_2}}{2} \right| \leq \frac{1}{4},$$

kde po úpravách dostáváme

$$|1 + c_2| \leq \frac{1}{4}. \tag{3.11}$$

Jak je z výsledku patrné, dostáváme kruh o poloměru $\frac{1}{4}$ se středem v bodě -1. Limitní případ (3.11), v tomto případě kružnici, můžeme zapsat konzistentně s řešením pro $n = 1$ v exponenciálním tvaru jako

$$c_2 = -1 + \frac{1}{4}e^{i\theta} \quad \text{pro} \quad \theta \in [0, 2\pi)\,.$$



Obrázek 3.4: Červeně zvýrazněná hyperbolická komponenta pro $n = 2$.

Výsledek opět můžeme interpretovat graficky na obrázku 3.4.

Pokud porovnáme pozice zvýrazněných oblastí na obrázcích 3.3 a 3.4, všimneme si, že mají nejspíš společný bod. Ověřme tuto domněnku analyticky. Spočítejme nyní průsečík c_1 z (3.7) a c_2 z (3.10). Tedy

$$z - z^2 = -1 - z - z^2,$$
$$z = -\frac{1}{2}.$$

Z výsledku plyne, že $c = \frac{3}{4}$. To také zároveň znamená, že průsečík je pouze bod dotyku mezi komponentami pro $n = 1$ a $n = 2$, jelikož nám vyšlo pouze jedno možné řešení. Pro lepší představu výsledek interpretujeme na obrázku 3.6.

Komponenty pro $n = 3$

Opět s využitím výsledku (3.5) víme, že

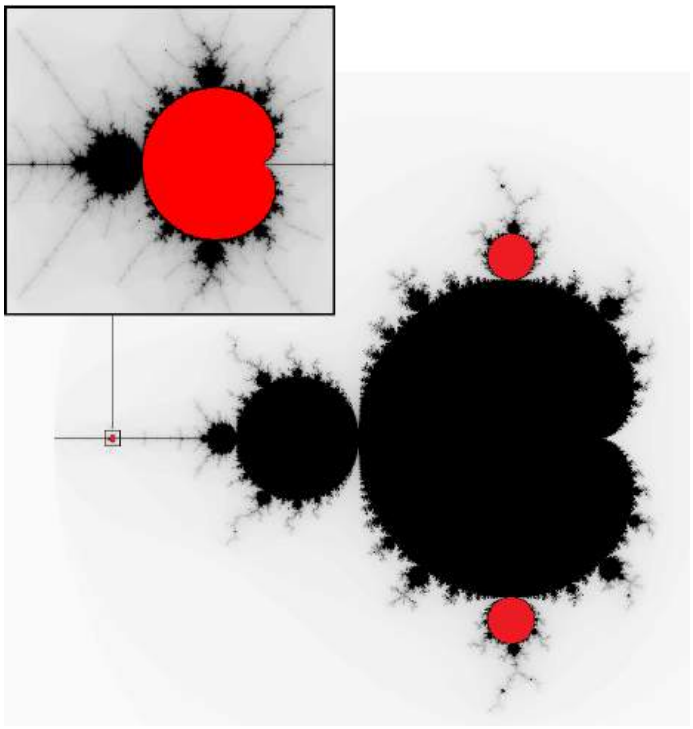
$$|z_1 \cdot z_2 \cdot z_3| \leq \frac{1}{8}.$$

Tentokrát vycházíme z předpokladu $z_{n+3} = z_n$, což nám dává následující polynomickou rovnici pro c stupně čtyři, která má po úpravách tvar

$$c^4 + c^3(4z^2 + 2) + c^2(6z^4 + 4z^2 + 1) + c(4z^6 + 2z^4 + 1) + (z^8 + 1) = 0, \qquad (3.12)$$

kde z je jedno z trojice z_1, z_2, z_3.

Věta 3.2.1 (Důsledek základní věty algebry)**.** *Každý polynom (každá algebraická rovnice) stupně n má v oboru komplexních čísel právě n kořenů (řešení). Přitom každý kořen počítáme i s jeho násobností.*



Obrázek 3.5: Červeně zvýrazněné hyperbolické komponenty pro $n = 3$.

Za pomoci následující věty z [3] víme, že budeme mít čtyři komponenty vymezující c, pro které kritická orbita konverguje k cyklu s periodou tři. Jak již ale víme z předcházející sekce, jedno řešení je komponenta periody jedna.

Věta 3.2.2. *Je-li c kořenem polynomu $P_n(x)$, lze tento polynom rozložit na tvar*

$$P_n(x) = (x - c)Q_{n-1}(x).$$

Tato věta nám zase zaručuje dělitelnost (3.12) již známým řešením (3.7) pro $n = 1$, čímž se výsledný polynom může zjednodušit. Podělením (3.12) členem $c - z + z^2$ a následnou úpravou dostáváme

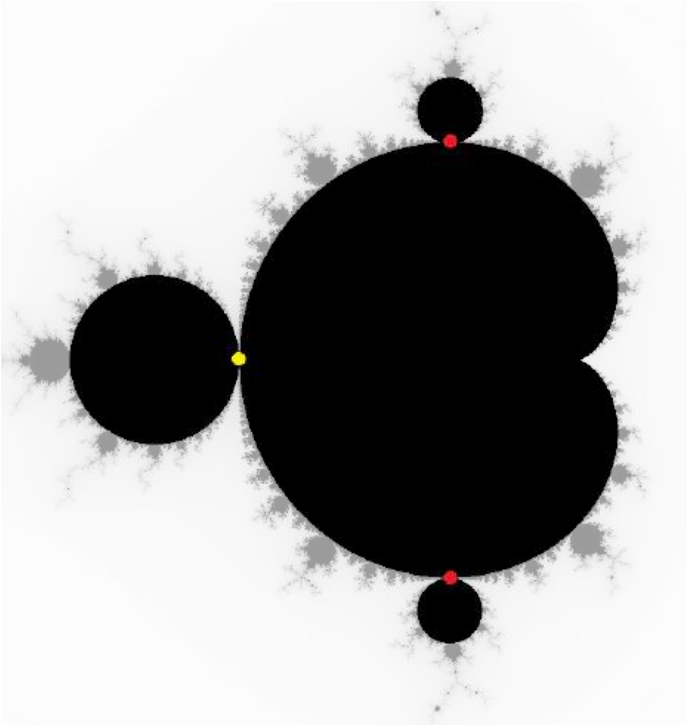
$$\begin{aligned} c^3 + c^2\left(3z^2 + z + 2\right) + c\left(3z^4 + 2z^3 + 3z^2 + 2z + 1\right) + \\ + \left(z^6 + z^5 + z^4 + z^3 + z^2 + z + 1\right) = 0 \end{aligned}$$

Podle stupně polynomu proměnné c opravdu existují tři komponenty s periodou tři. To navíc ze symetrie Mandelbrotovy množiny značí, že jedna z nich musí být na reálné ose a další dvě buď mimo reálnou osu a navzájem si symetrické, nebo obě také na reálné ose, viz obrázek 3.5.

Pokusme se opět spočítat průsečík. Jak již bylo zmíněno dříve, předpokládáme, že průsečík komponent $n = 1$ a $n = 3$ existovat určitě bude, naopak $n = 2$ a $n = 3$ ne, jelikož 2 není dělitelem 3.

Dosazením $c = z - z^2$ do (3.12) a úpravou nám vyjde

$$4z^2 + 2z + 1 = 0.$$



Obrázek 3.6: Body dotyku komponent s periodou $n = 1$ a $n = 2$ (žlutě), resp. $n = 1$ a $n = 3$ (červeně).

Po využití diskriminantu mají hodnoty $z1$ a z_2 tvar $\frac{-1 \pm i\sqrt{3}}{4}$. Dosazením zpět do c pak máme průsečíky v bodech

$$c = \frac{-1 \pm i3\sqrt{3}}{8}.$$

Protože jsme získali dva body dotyku symetrické podle reálné osy, viz obrázek 3.6, můžeme už nyní říct, že dvě komponenty leží mimo reálnou osu a jedna někde na reálné ose.

3.2.3 Středy hyperbolických komponent

Jak jsme viděli v předchozí podkapitole, nalezení hyperbolických komponent se stává komplikovanější již od periody tři. Bylo by tedy vhodné zkusit tento výpočet zefektivnit i za cenu získání méně informací o komponentě. Pomocí předchozí metody jsme totiž získali přesný tvar a polohu komponent pro $n = 1, 2$ a cenné informace o komponentě pro $n = 3$. Zkusme tedy nyní získat pouze to nejdůležitější, což by v tomto případě měl být počet komponent pro jednotlivá n a pozice v komplexní rovině. Na rozdíl od předchozí kapitoly nám však bude stačit pouze jeden bod náležící hyperbolické komponentě a to její *střed*.

Střed komponenty

Jako první je ale potřeba definovat, co se myslí pod pojmem střed nějaké kompaktní množiny v $\mathbb{C}$. Až na výjimky se sice může zdát, že hyperbolické komponenty jsou kruhy a nalezení středu by tak nebylo nic složitého. Tento fakt platí však bohužel pouze $n = 2$. Zadefinujeme tedy střed oblasti jako její geometrický střed nebo také jako těžiště za předpokladu, že uvažujeme konstantní hmotnost všech částí oblasti.

Vycházíme opět z nerovnosti (3.4), kde pro jednoduchost položíme $g(c) = |2^n \cdot \prod_{i=1}^{n} z_i|$. Budeme tedy psát, že

$$g(c) \leq 1.$$

Funkce g je funkcí parametru c. Zprvu se to může zdát poněkud matoucí, avšak připomeňme, že posloupnost bodů $\{z_i\}_{i=1}^{n}$ jsou body kritické orbity, což pro Mandelbrotovu množinu znamená $z_0 = 0$ a pak $z_1 = c$. Tedy chování orbity závisí pouze na parametru c.

Funkce g je nekonstantní, spojitá funkce. Kdybychom ji vykreslili pomocí trojrozměrného grafu, vznikl by útvar podobný díře. Víme totiž, že na hranici komponenty je velikost derivace rovna jedné, naopak v jednom z jejich vnitřních bodů, kde je velikost derivace rovna nule, se nachází minimum a toto minimum by mohla být rozumná volba středu.

Z definice 3.2 víme, že pro střed komponenty bude platit super-přitahující vlastnost bodu, tedy že $f'(0) = 0$. Navíc platí, že v super-přitahujícím cyklu je jeden z bodů cyklu kritický bod. To v našem případě znamená $z_0 = 0$. Například tedy střed pro $n = 1$ je počátek souřadnicového systému. Protože aby vše bylo splněno, už po první iteraci musí platit $f(0) = 0$. Pokud bychom chtěli najít střed pro komponentu s periodou dva, musí již druhá iterace být rovna nule (obsahovat kritický bod). I zde existuje pouze jedno řešení $c = -1$. Kontrolou je provedení prvních dvou iterací. Počáteční bod a zároveň kritický bod $z_0 = 0$. Pak následuje $z_1 = 0^2 - 1 = -1$ a nakonec druhá iterace $z_2 = (-1)^2 - 1 = 0$, což je opět kritický bod.

Obecně tedy platí, že pro nalezení středu komponenty s periodou n stačí vygenerovat n iterací kritické orbity pro $z_0 = 0$ a sledovat, jestli platí $f^n(0) = 0$. Pokud ano, pak zkoumaný parametr c je hledaný střed.

Počet hyperbolických komponent

Postupem z předcházející sekce získáme polynom $P_n(c) = f^n(0) = 0$, kde polynomy pro prvních pár n jsou tvaru:

$$\begin{aligned}
P_1(c) &= c, \\
P_2(c) &= c^2 + c, \\
P_3(c) &= c^4 + 2c^3 + c^2 + c, \\
P_4(c) &= c^8 + 4c^7 + 6c^6 + 6c^5 + 4c^4 + 2c^3 + c^2 + c, \\
&\vdots
\end{aligned}$$

Zřejmě stupeň polynomu P_n je 2^{n-1}. To také znamená, že P_n musí mít podle věty 3.2.1 právě 2^{n-1} kořenů. To by nás mohlo vést k myšlence, že v Mandelbrotově množině existuje 2^{n-1} hyperbolických komponent s periodou n. Avšak jak již víme z dřívějšího pozorování, počet pro $n = 1$ je 1, pro $n = 2$ je také 1 a pro $n = 3$ existují komponenty 3, což už nekoresponduje s počtem řešení polynomu $2^0, 2^1, 2^2, \ldots$. Tento problém už byl také nastíněn dříve. Zjistili jsme, že komponenta s periodou n je také komponentou s periodou $2n, 3n, 4n, \ldots$. To právě například pro $n = 3$ znamená, že do řešení polynomu P_n počítáme také jedno řešení pro $n = 1$. Obecně totiž počet řešení polynomu P_n odpovídá počtu komponent s periodou n, ale také počtu komponent s periodou k, kde $k|n$. Nechť počet komponent přesně periody n je U_n. Potom platí následující vztah:

$$U_n = 2^{n-1} - p,$$

kde

$$p = \sum_{k|n,\, k<n} U_k.$$

Zkusme tento vztah aplikovat na prvních pár n:

$$\begin{aligned} U_1 &= 2^{1-1} - 0 = 1, \\ U_2 &= 2^{2-1} - 1 = 1, \\ U_3 &= 2^{3-1} - 1 = 3, \\ U_4 &= 2^{4-1} - 2 = 6, \\ &\vdots \end{aligned}$$

Nyní se již výsledky shodují s tím, co jsme rozebírali v sekci 3.2.2. Přirozeně můžeme očekávat, že pokud $n \to \infty$, pak i $U_n \to \infty$. Pro zajímavost poznamenejme, že například už pro periodu 20 existuje 524 283 hyperbolických komponent. Zvýšíme-li periodu na 50, dostáváme řádově vyšší číslo 562 949 953 421 307. Jedná se o číslo přesahující pět set šedesát bilionů.

Vraťme se však k polynomům $P_n(c)$. Jak vidíme, z celkových 2^{n-1} řešení polynomu $P_n(c)$ nás ve výsledku bude zajímat pouze U_n. Podle věty 3.2.2 musí být $P_n(c)$ dělitelný všemi $P_k(c)$ takovými, kde $k|n$. Označme $Q_n(c)$ jako redukovaný polynom, jehož řešením jsou pouze kořeny pro komponenty periody n bez jejich dělitelů k. Tvar $Q_n(c)$ je pak následující:

$$Q_n(c) = \frac{P_N(c)}{\prod_{k|n,\, k<n} Q_k(c)} \quad \text{pro} \quad Q_1(c) = P_1(c).$$

To tedy znamená, že pro prvních pár period n budou polynomy Q_n vypadat následovně:

$$\begin{aligned} Q_1(c) &= c, \\ Q_2(c) &= c + 1, \\ Q_3(c) &= c^3 + 2c^2 + c + 1, \\ Q_4(c) &= c^6 + 3c^5 + 3c^4 + 4c^3 + 2c^2 + 1, \\ &\vdots \end{aligned}$$

Položíme-li jednotlivé polynomy rovny nule, tak vyřešením najdeme všechny středy, a tedy zároveň i všechny hyperbolické komponenty s danou periodou. Dokonce víme, že kořeny budou buď reálné, nebo vzájemně komplexně sdružené. Tento fakt vychází opět ze symetrie Mandelbrotovy množiny podle reálné osy.

Aplikace polynomu $Q_n(c)$ pro dané n

V této sekci nyní převedeme do praxe předchozí teoretické závěry týkající se výpočtu středu hyperbolické komponenty. Rozebereme postupně opět pouze prvních pár n. I zde bude jasně vidět, jak se i pro malá n velice rychle zvyšuje výpočetní náročnost.

Polynom $Q_1(c)$

Jak už bylo zmíněno výše, pouze položíme daný polynom rovno nule. Tedy

$$\begin{aligned} Q_1(c) &= 0, \\ c &= 0. \end{aligned}$$

Okamžitě vidíme, že střed je opravdu v bodě $c = 0$.

Polynom $Q_2(c)$

Opět si nejprve napišme rovnost pro periodu dva a následně výsledek pro parametr c:

$$\begin{aligned} Q_2(c) &= 0, \\ c + 1 &= 0, \\ c &= -1. \end{aligned}$$

To znamená, že střed komponenty pro periodu dva se nachází v bodě $c = -1$. Tento a i předchozí výsledek již byli rozebírány v úvodu této podkapitoly, jelikož jejich nalezení nebylo nikterak obtížné. Pojďme se však nyní podívat na výpočet středů periody tři.

Polynom $Q_3(c)$

$$\begin{aligned} Q_3(c) &= 0, \\ c^3 + 2c^2 + c + 1 &= 0. \end{aligned}$$

Výsledek této rovnice už není tak zřejmý a pro jeho nalezení budeme muset využít některou z numerických metod, např. Newtonovu metodu. Pomocí této iterační metody pak získáme jeden ze středů komponenty s přesností na pět desetinných míst $c_1 = -1{,}75488$. Jak už jsme naznačili, jedná se právě o jeden z bodů na reálné

ose. S využitím věty 3.2.2 pak můžeme polynom Q_3 podělením už známými kořeny zjednodušit na tvar

$$\frac{c^3 + 2c^2 + c + 1}{c + 1{,}75488} = c^2 + 0{,}02451c + 0{,}56984,$$

odkud pak řešením kvadratické rovnice s komplexními kořeny dostáváme

$$c_{2,3} = -0{,}12256 \pm 0{,}74486i.$$

Takto jsme dostali zbývající dva středy hyperbolických komponent s periodou tři, které jsou vzájemně komplexně sdružené a tedy i symetrické podle reálné osy.

Tímto výpočtem jsme navíc získali i pozici třetí komponenty pro $n = 3$. Jako zajímavost uveďme, že tato třetí komponenta po přiblížení vůbec nepřipomíná kruh, jako všechny ostatní této periody, ale naopak se tvarem významně podobná Mandelbrotově množině, viz obrázek 3.5 z předchozí sekce.

Polynom $Q_4(c)$

Nyní zkusíme pokročit ještě dál a spočítáme kořeny polynomu pro komponenty s periodou čtyři. Stejným postupem opět získáme

$$\begin{aligned} Q_4(c) &= 0, \\ c^6 + 3c^5 + 3c^4 + 3c^3 + 2c^2 + 1 &= 0, \end{aligned}$$

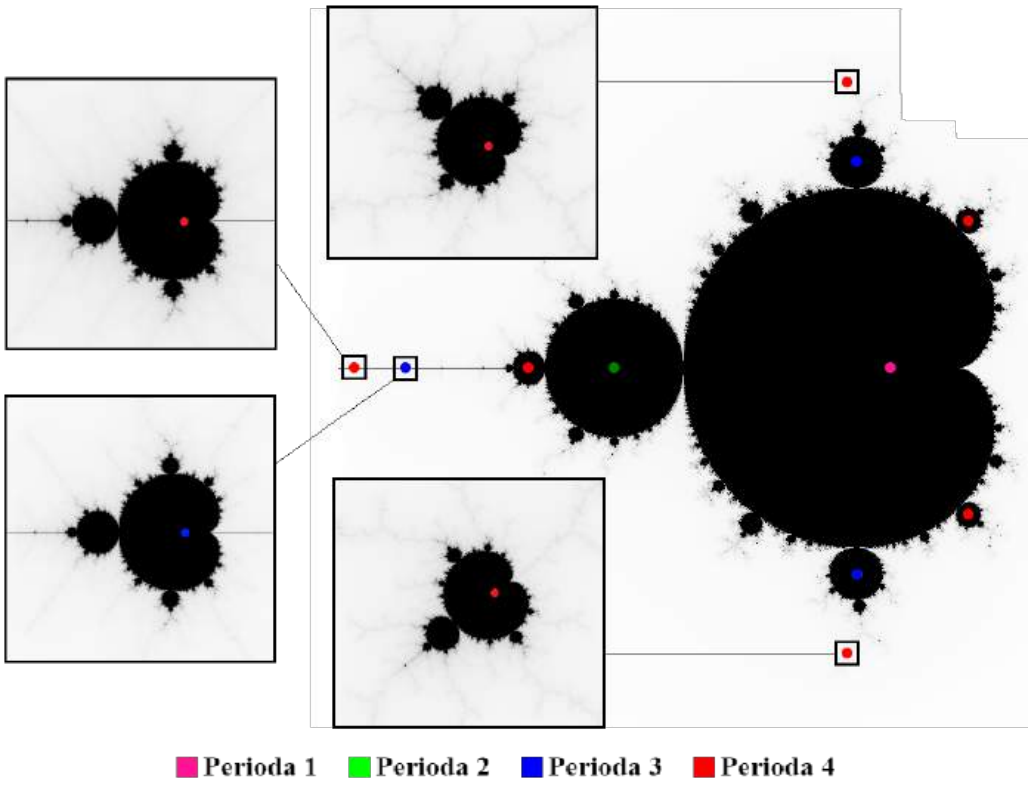
kde opět pomocí iterační metody získáme tyto dva reálné kořeny $c_1 = -1{,}94080$ a $c_2 = -1{,}31070$. Stejně jako minule, podělíme již známými kořeny, což nám dá výsledný polynom

$$c^4 - 0{,}25150c^3 + 1{,}27395c^2 - 0{,}50247c + 0{,}39310 = 0.$$

Protože se jedná o čtyři komplexní kořeny, víme, že se musí zákonitě jednat o dvojice komplexně sdružených řešení. Následně pomocí vybrané numerické metody obdržíme zbylé kořeny polynomu $Q_4(c)$ jako

$$\begin{aligned} c_{3,4} &= 0{,}28227 \pm 0{,}53005i, \\ c_{5,6} &= -0{,}15652 \pm 1{,}032247i. \end{aligned}$$

Jak je vidět, výpočet středů hyperbolických komponent pro danou periodu je relativně jednoduchý. Je potřeba pouze sestrojit polynom $Q_n(c)$, položit jej rovný nule a následně vhodně zvolenou numerickou metodou spočítat kořeny polynomu. Výsledek pro středy hyperbolických komponent s periodami $n = 1,\ 2,\ 3$ a 4 jsme opět ilustrovali graficky na obrázku 3.7. Tečky představující středy velikostně neodpovídají skutečnosti a jsou mnohonásobně zvětšeny, aby je bylo na obrázku možné bezproblémově najít. Můžeme si také všimnout, jak již bylo zmíněno i dříve, že hyperbolické komponenty nejsou tvarem podobné pouze kruhu, ale také dost často vypadají jako menší kopie celé Mandelbrotovy množiny lišící se pouze velikostí. Na obrázku jsou s různě velkým přiblížením vyobrazeny v rámečcích některé komponenty periody tři a čtyři.



Obrázek 3.7: Pozice středů hyperbolických komponent pro $n = 1, 2, 3$ a 4.

3.3 Newtonova metoda jedné komplexní proměnné

Newtonova metoda, také někdy Newton-Raphsonova metoda[2], je jednou z nejznámějších iteračních metod pro hledání kořenů funkce $f(z) = 0$, za předpokladu, že funkce $f(z)$ je holomorfní (komplexně diferencovatelná). Je hojně využívaná i kvůli řádu konvergence, který je dva (viz 3.1). To jinými slovy znamená, že konvergence metody je pro vhodně zvolenou počáteční hodnotu poměrně rychlá. Postup Newtonovy iterační metody je následující:

1. Zvolíme počáteční hodnotu iterace z_0 z definičního oboru funkce f.
2. Získáme hodnotu z_{n+1} pomocí vztahu:

$$z_{n+1} = N(z_n) = z_n - \frac{f(z_n)}{f'(z_n)},$$

   kde $f'(z_n)$ je derivace funkce $f(z)$. N nazveme Newtonova funkce.
3. Opakujeme, dokud není $|z_{n+1} - z_n| < \epsilon$, pro dostatečně malé $\epsilon > 0$.

Jak si lze všimnout, postup při výpočtu je totožný s Newtonovou metodou v $\mathbb{R}$ oboru. Jediný rozdíl je pouze ve volbě počátečního bodu iterace. Zvolíme-li z_0 jako

[2] V $\mathbb{R}$ známá také jako metoda tečen, jelikož přesnější aproximaci hledáme právě ve směru tečny funkce $f(x)$.

komplexní číslo, budeme schopni nalézt i komplexní kořeny polynomu. Je zde však zásadní, správně zvolit tento počáteční bod. Jelikož i v oboru $\mathbb{R}$ čísel už může nastat několik situací, kdy Newtonova metoda selže. Pokud například počáteční bod iterace je kritický bod $f(x)$, a tedy pak $f'(x) = 0$. Což ve smyslu směrnice tečny značí vodorovnou přímku, která nikdy neprotne osu x. Takže ve výsledku nebude existovat ani další bod iterace. Dále může nastat problém, pokud se aproximace řešení dostanou do cyklu a k nejednoznačnému řešení. V neposlední řadě nemusí mít polynom v $\mathbb{R}$ žádné řešení. Pro podrobnější informace viz [34].

Vraťme se k $\mathbb{C}$ případu. Jak již víme z věty 3.2.1, počet kořenů polynomu v $\mathbb{C}$ odpovídá jeho stupni. Otázkou tedy je, co se stane, pokud má polynom více kořenů, což v našem případě bude prakticky vždy až na $Q_1(c)$ a $Q_2(c)$. Odpovědí je, že volbou různých počátečních bodů z_0 najdeme různé kořeny.

Oblasti přitažení

Tímto se dostáváme k definici oblasti, kde všechny body z této oblasti zvolené jako počáteční body iterace konvergují ke stejnému kořenu polynomu.

Definice 3.3. Pokud je x_* kořen funkce f, oblast přitažení (*basin of attraction*) bodu x_* je množina všech bodů x_0, pro které Newtonova metoda s počátečním bodem iterace x_0 konverguje k x_*. Symbolicky:

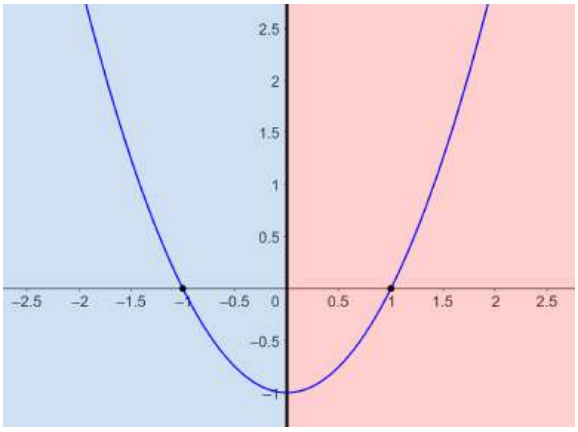
$$A(x_*) = \{x_0 \mid x_n = N^n(x_0) \text{ konverguje k } x_*\}.$$

Nejlepší varianta jak analyzovat chování oblastí přitažení v $\mathbb{C}$ oboru bude zakreslit tyto oblasti do obrázku. Přestavme si komplexní rovinu jako množinu počátečních bodů iterace Newtonovy funkce N. Následně je obarvíme podle toho, ke kterému kořenu bude metoda konvergovat.

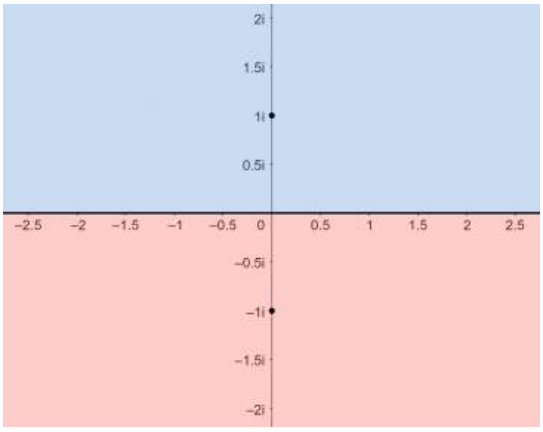
Jako první začneme funkcí $f(z) = z^2 - 1$. Není složité určit, že kořeny tohoto polynomu jsou $z = 1$ a $z = -1$. V tomto případě máme všechny kořeny reálné. Můžeme si tedy vykreslit funkci $y = x^2 - 1$ (viz obrázek 3.8), která představuje parabolu s vrcholem v bodě -1 na ose y. Navíc při pohledu na tvar paraboly lze odvodit oblasti přitažení spočítaných kořenů. Jelikož se jedná o reálné kořeny, můžeme se tedy i pro počáteční body iterace omezit pouze na $\mathbb{R}$ osu. To tedy znamená, že v tomto konkrétním případě vynecháme v zápisu čísla imaginární část (např. $z_0 = 0 + 0i = 0$). První, co z grafu vidíme, je, že bod $z = 0$ je kritický bod a zde Newtonova metoda selže. Následně vidíme, že všechny počáteční body $z_0 > 0$ budou konvergovat ke kořenu $z = 1$. Naopak pro počáteční body $z_0 < 0$ Newtonova metoda dokonverguje k $z = -1$. Budeme-li vycházet ze značení v definici 3.3, tak $A(1) = (0, \infty)$ a $A(-1) = (-\infty, 0)$. Tedy jinými slovy jsme rozdělili komplexní rovinu na dvě oblasti, které od sebe dělí imaginární osa.

Druhým příkladem je funkce $f(z) = z^2 + 1$. Opět poměrně jednoduše nalezneme kořeny $z = i$ a $z = -i$. Můžeme tedy očekávat, že komplexní rovina bude rozdělena stejně jako v předchozím případě na dvě části. Jak vidíme z obrázku 3.9, tentokrát je však jejich hranicí reálná osa.

Na závěr uveďme ještě asi nejznámější funkci spojenou s Newtonovou metodou a to $f(z) = z^3 + 1$. Kořeny tohoto kubického polynomu jsou $z = -1$ a $z = \frac{1 \pm i\sqrt{3}}{2}$. Na



Obrázek 3.8: Výřez grafu paraboly $f(x) = x^2 - 1$. V pozadí barevně odlišené oblasti přitažení pro $A(-1)$ (modře) a pro $A(1)$ (červeně). Zvýrazněná imaginární osa představuje hranici těchto dvou oblastí.



Obrázek 3.9: Výřez komplexní roviny barevně rozdělené oblastmi přitažení kořenů funkce $f(z) = z^2 + 1$ pro $A(-i)$ (modře) a pro $A(i)$ (červeně). Zvýrazněná reálná osa představuje hranici těchto dvou oblastí.

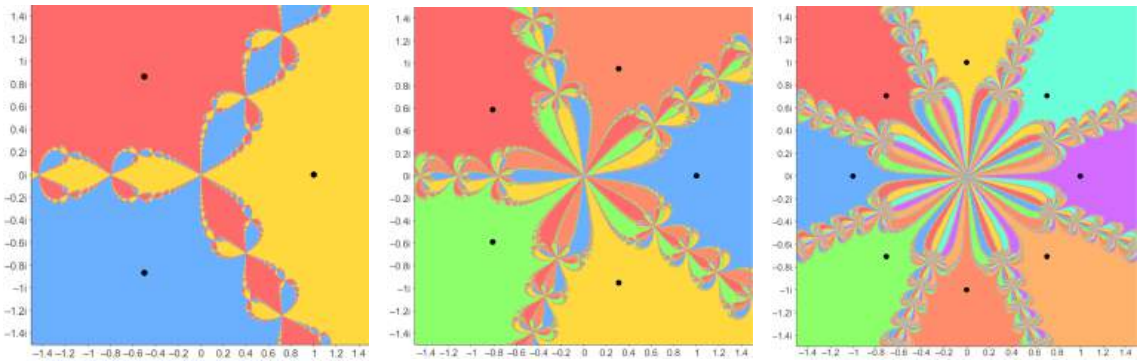
obrázku 3.10a vidíme dobře známý fraktální obrazec nazývaný také Newtonův fraktál. Je ještě důležité zmínit, že touto interpretací vlastně zobrazujeme Juliovu množinu (hranice mezi oblastmi) a Fatouovu množinu (barevné oblasti přitažení). Zvolíme-li tedy jako počáteční bod iterace jakýkoliv bod z Juliovy množiny, Newtonova metoda selže. Jen pro zajímavost jsou na obrázku 3.10 uvedeny ještě další oblasti přitažení. Jak si lze všimnout, s vyšším stupněm polynomu roste také citlivost na volbu počátečního bodu iterace.

Volba počátečního bodu pro výpočet středu hyperbolických komponent

Nyní přejděme ke konkrétnímu problému, který vznikl v sekci 3.2.3. Zde jsme se bavili o tom, že pro výpočet středů hyperbolických komponent bude zejména pro vyšší periody nutné využít Newtonovu metodu. Jak už víme, bude dost záležet na volbě počáteční hodnoty. K tomu nám právě dopomohou oblasti přitažení.

V úvodních příkladech jsme ale prvně znali kořeny a následně podle nich určovali oblasti přitažení. Nyní bychom to chtěli naopak. Chceme znát nejprve oblasti přitažení a následně volbou počátečních bodů pro iterační proces z jednotlivých oblastí dokonvergovat k řešení, tedy v našem případě ke středu hyperbolické komponenty. Připomeňme, že pro c ve středu hyperbolické komponenty s periodou n dostaneme po n iteracích pro $z_0 = 0$ opět $z_n = 0$, protože střed je super-přitahující pevný bod. Z toho plyne, že body blízko středu nebudou sice přesně splňovat vlastnost super-přitahujícího pevného bodu, ale jejich hodnoty z_n budou velmi malé. Obecně tedy zadefinujme body oblasti přitažení pro střed hyperbolické komponenty s periodou n jako body komplexní roviny, jejichž absolutní hodnota po n iteracích dosáhne nového minima (vyjma počátečního bodu $z_0 = 0$).

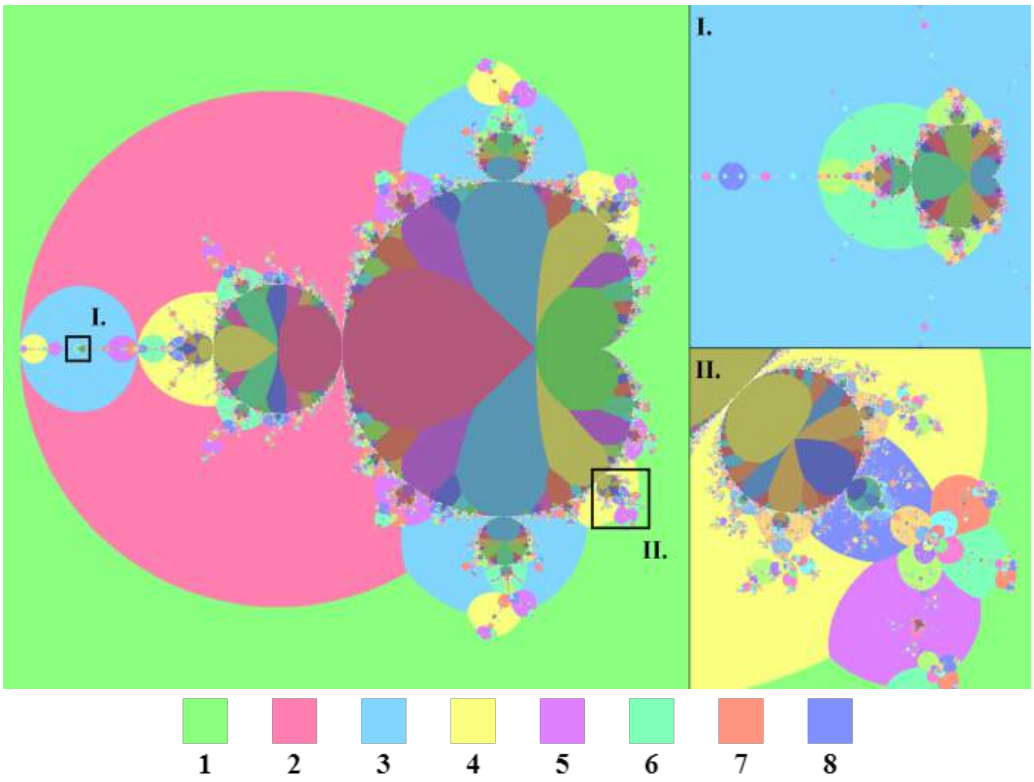
Otázkou je, proč k volbě počátečního bodu nevyužít již známé hyperbolické



(a) $n = 3$ **(b)** $n = 5$ **(c)** $n = 8$

Obrázek 3.10: Výřez barevně rozdělené komplexní roviny oblastmi přitažení kořenů funkce $f(z) = z^n + 1$. Kořeny jsou zvýrazněny černými body.

komponenty. Volbou právě z těchto komponent bychom také dokonvergovali k danému středu. Odpověď bude jasnější z obrázku 3.11, kde můžeme vidět barevně rozlišené oblasti přitažení pro jednotlivé středy hyperbolických komponent. Jak si lze na první pohled všimnout, oblasti přitažení jsou mnohonásobně větší, než pouhé hyperbolické komponenty. Zejména pro vyšší periody by byla volba počáteční iterace pouze z hyperbolických komponent opravdu složitá. Jsou zde také dále přiblíženy hyperbolické komponenty s periodou tři I. a čtyři II., abychom si dokázali lépe představit, jak záleží na volbě počátečního bodu iterace, pohybujeme-li se navíc poblíž hranice. Pro jednodušší představu jsme do obrázku zakomponovali i stín Mandelbrotovy množiny.



Obrázek 3.11: Barevně rozlišené oblasti přitažení pro jednotlivé středy hyperbolických komponent. V pravé části pak přiblížena jedna z komponent periody tři (I.) a periody čtyři (II.).

Kapitola 4

Juliovy množiny

V této kapitole rozebereme chování a vlastnosti Juliových množin. Hlavní zdroje potřebné k této kapitole jsou [4], [14] a [25]. Pro jednotlivé typy funkcí Juliových množin a jejich vlastností byla využita literatura [6], [18], [21], [28] a [29]. Dalšími pomocnými zdroji pro vytvoření této kapitoly byly také ještě [23] a [26]. Stejně jako u kapitoly 2, tak i zde platí, že většina důkazů je obsažena ve výše zmíněných zdrojích. V textu proto uvádíme pouze důležitější důkazy a ty zbylé lze dohledat v poskytnuté literatuře. Nejprve zavedeme pojem normální rodiny a následně připomeneme vlastnosti pro Juliovu množinu kvadratického zobrazení z^2+c. Poté už se podíváme na její zobecněné verze. Ať už se bude jednat o obecný exponent d, tedy $z^d + c$, nebo o transcendentní funkce $\lambda e^z,\ \lambda \sin z,\ \lambda \cos z$.

4.1 Normální rodiny

Abychom mohli dále pracovat se zobecněnými Juliovými množinami, je nutné zavést důležitý pojem komplexní analýzy – *normální rodiny*. Tento koncept však vyžaduje několik přípravných kroků. Nejprve musíme zajistit, aby rodina funkcí byla stejnoměrně spojitá, což je nezbytný předpoklad pro aplikaci další teorie.

Definice 4.1. Rodina funkcí $\mathcal{F} = \{f : K \to \mathbb{C}, f \in \mathcal{F}\}$ je *stejnoměrně spojitá* na K, pokud

$$\forall \varepsilon > 0, \exists \delta > 0, |z - z_0| < \delta \implies |f(z) - f(z_0)| < \varepsilon, \forall f \in \mathcal{F}.$$

Jakmile máme zajištěnou vlastnost stejnoměrné spojitosti, můžeme uvést první větu, konkrétně Arzelà-Ascoli větu. Ta nám umožňuje tvrdit, že rodina funkcí, která je stejnoměrně spojitá, obsahuje podposloupnosti, které konvergují stejnoměrně.

Věta 4.1.1 (Arzelà-Ascoli)**.** *Pokud je $\mathcal{F}$ stejnoměrně spojitá na K, pak každá posloupnost f_n v $\mathcal{F}$ má podposloupnost, která konverguje stejnoměrně na K.*

Nechť $\Omega \subset \mathbb{C}$ je otevřená množina. Rodina $\mathcal{F} = \{f^1 : \Omega \to \mathbb{C}, f \in \mathcal{F}\}$ se nazývá *normální*, pokud pro každou posloupnost $f_n \in \mathcal{F}$ existuje podposloupnost, která

[1] analytická funkce

konverguje normálně na Ω. Termín *normálně* zde znamená, že konvergence je stejnoměrná na každé kompaktní podmnožině Ω.

Následující věta říká, že pro posloupnost analytických funkcí stačí velmi málo k tomu, aby byla normální. Funkce musí být pouze stejnoměrně ohraničená na každé kompaktní množině $K \subset\subset \Omega$ stejnou konstantou, kde Ω je otevřená množina. Jen připomeňme, že termín *analytická* je nadřazený termínu *holomorfní*, o kterém bude řeč později. Spojení analytická funkce je používáno v širším smyslu k označení jakékoli funkce (reálné, komplexní nebo obecnějšího typu), na rozdíl od holomorfní funkce, která je používána pouze v kontextu komplexních funkcí. Nyní už tedy zavedeme známou Montelovu větu, která nám definuje pojem normální rodiny.

Věta 4.1.2 (Montel)**.** *Pokud $\mathcal{F}$ je rodina analytických funkcí definovaných na otevřené množině $\Omega \subset \mathbb{C}$, která je stejnoměrně ohraničená na každé kompaktní podmnožině Ω, pak $\mathcal{F}$ je normální.*

Pro názornost si uvedeme příklad. Mějme rodinu holomorfních funkcí $\mathcal{F} = \{f_n(z) = z^n\}$. Abychom zaručili, že tato rodina bude normální, vyjdeme přímo z Montelovy věty a budeme se snažit pro výše definovanou rodinu splnit všechny podmínky. Chceme tedy najít rodinu holomorfních funkcí, které jsou stejnoměrně ohraničené na kompaktní podmnožině nějaké otevřené množiny v $\mathbb{C}$. Tento předpoklad bude $\mathcal{F}$ určitě splňovat, pokud ji omezíme například na otevřený jednotkový kruh $\{z : |z| < 1\}$. Tady určitě bude platit, že $|f_n(z)| \leq 1$ pro všechna z v libovolné uzavřené podmnožině otevřeného kruhu. Z toho plyne, že $\mathcal{F}$ je stejnoměrně ohraničená, a tedy pak $\mathcal{F}$ je normální rodina funkcí. Pokud bychom ale už jen změnili omezení na $\{z : |z| < 2\}$, porušili bychom tím předpoklad stejnoměrné ohraničenosti, protože obecně pro $z > 1$ už platí, že $|f_n(z)| > 1$. To znamená, že bychom nebyli schopni v tomto případě funkci ohraničit nějakou stejnou konstantou pro $n \to \infty$. Rodina funkcí $\mathcal{F}$ pak normální nebude.

4.2 Juliovy množiny kvadratické funkce

V tomto momentě bude vhodné ve zkratce připomenout definici Juliovy množiny a její základní vlastnosti. Bude tak i jednodušší následné porovnání s vlastnostmi zobecněných Juliových množin. V práci se můžou vyskytovat zápisy Juliovy množiny jako $J_c(f)$, nebo také pouze J_c. Pokud bude z kontextu jasné, o čem je řeč. Stejně tak pro ostatní názvy množin.

Juliova množina je definována jako množina bodů komplexní roviny $\mathbb{C}$, jejíž dynamické vlastnosti jsou určeny iterací komplexní kvadratické funkce

$$f(z) = z^2 + c \quad z, c \in \mathbb{C}.$$

Nezapomeňme, že formálně je Juliova množina $J_c(f)$ hranicí mezi dvěma základními množinami:

- *Prisoner set*, tj. množina bodů $z \in \mathbb{C}$, jejichž orbita po n iteracích pro $n \to \infty$ zůstává omezená:

$$P_c(f) = \{z \in \mathbb{C} \mid \exists R > 0, \forall n \in \mathbb{N} : |f^n(z)| \leq R\}.$$

- *Escape set*, tj. množina bodů $z \in \mathbb{C}$, jejichž orbita po n iteracích pro $n \to \infty$ diverguje do nekonečna:

$$E_c(f) = \{z \in \mathbb{C} \mid \lim_{n\to\infty} |f^n(z)| = \infty\}.$$

Juliova množina je poté definována jako hranice množiny $K_c(f)$:

$$J_c(f) = \partial K_c(f),$$

kde $K_c(f)$ je tzv. *Filled Juliova množina* (definována v 2.2).

4.2.1 Základní vlastnosti

Mezi základní vlastnosti Juliových množin patří:

Základní dualita

Juliova množina J může mít dva základní typy struktur, a to navíc v závislosti na poloze parametru c vzhledem k Mandelbrotově množině:

1. Pokud c leží uvnitř Mandelbrotovy množiny, pak je J_c kompaktní a souvislá množina (tvoří jednu souvislou oblast).
2. Pokud c leží mimo Mandelbrotovu množinu, pak je J_c úplně nesouvislá (nekonečně mnoho oblastí).

Symetrie a soběpodobnost

Juliovy množiny J_c jsou obecně pro $c \in \mathbb{C}$ středově symetrické vůči počátku.

$$z \in J_c \quad \Longleftrightarrow \quad -z \in J_c.$$

Navíc, pokud $c \in \mathbb{R}$, pak je J_c symetrická i vůči reálné ose.

Pokud je $c \neq 0$, pak obecně J_c vykazuje fraktální strukturu. Připomeňme, že fraktální obrazce mají vlastnost tzv. soběpodobnosti (invariance vůči změně měřítka). Tedy, že nezávisle na zvětšení vykazují podobnost k části sebe sama.

Invariance vůči transformaci

Množina $J_c(f)$ je invariantní vůči transformaci funkce f, což znamená, že:

$$f(J_c(f)) = J_c(f) \quad \text{a také} \quad f^{-1}(J_c(f)) = J_c(f).$$

Říkáme, že Juliova množina je kompletně invariantní vůči f.

Juliova množina jako repelor

Věta 4.2.1. *Juliova množina je uzávěrem množiny všech odpuzujících periodických bodů funkce f.*

Juliova množina je repelor pro body mimo ni. Orbita s počáteční hodnotou v Juliově množině zůstává v množině uvězněna. Naopak počáteční hodnoty $z_0 \in E_c(f)$ divergují k nekonečnu a zároveň body $z_0 \in P_c(f)$ konvergují k přitahujícímu pevnému bodu, nebo k cyklu s určitou periodou. V každém případě se však nikdy nepřibližují k $J_c(f)$. Tedy Juliova množina působí odpudivě na orbity s počáteční hodnotou $z_0 \notin J_c(f)$.

Limitní poloměr divergence

Limitní poloměr divergence je klíčovým aspektem při grafické vizualizaci Juliových množin.

Věta 4.2.2. *Pro kvadratické zobrazení $f(z) = z^2 + c$ platí, jestliže*

$$|z| > max(|c|, 2),$$

pak orbita z diverguje k nekonečnu. Hodnota $r(c) = max(|c|, 2)$ se nazývá limitní poloměr divergence.

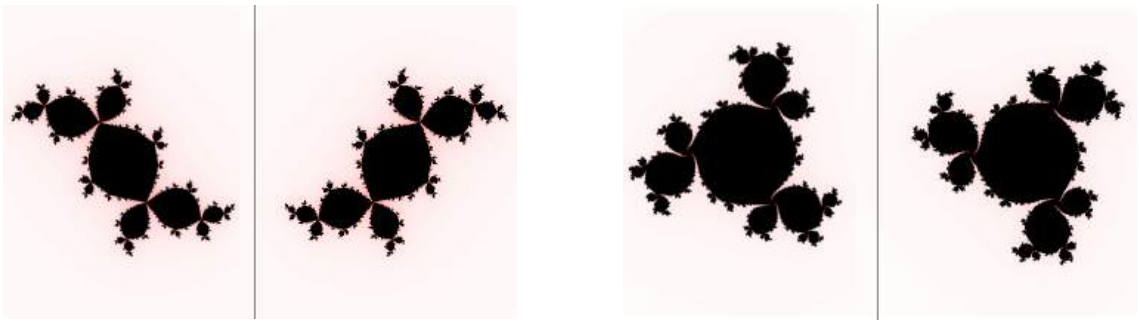
Touto rovností dokážeme ušetřit velké množství času při klasifikaci bodu, jestli náleží J_c, nebo ne.

4.3 Juliovy množiny funkce s $\mathbb{Q}$ exponentem

Po stručném shrnutí pro kvadratickou funkci se zaměříme na Juliovy množiny funkcí s obecným exponentem d, konkrétně $f(z) = z^d + c$ a rozebereme jejich vlastnosti.

Pro $d \in \mathbb{Z}$ a $c \in \mathbb{C}$ se rodina funkcí $\mathcal{F} = \{f^n(z) \mid f(z) = z^d + c\}_{n=0}^{\infty}$ chová na komplexní rovině dvěma různými způsoby. Buď konverguje k nějaké funkci na otevřené množině (*normální* chování), nebo má body, které vykazují chaotické chování. Tedy, že i body velmi blízko sobě v komplexní rovině mohou mít po několika iteracích značně odlišné chování. Jinými slovy chaotické chování znamená, že malé změny v počáteční hodnotě z_0 vedou k tomu, že chování orbity tohoto bodu se začne velmi rychle odlišovat od orbit okolních bodů, které jsou na začátku i velmi blízko sobě, což vede k nečekaným a těžko předvídatelným výsledkům.

$\mathcal{F}$ je normální rodinou na otevřené množině, která se nazývá *Fatouova množina* F_c ale zároveň $\mathcal{F}$ není normální na uzavřené množině, která je *Juliovou množinou* J_c, pro kterou platí $F_c = \mathbb{C} \setminus J_c$ (doplněk do F_c). Vidíme, že definice je totožná s definicí pro kvadratický případ (viz například [23]). Může nám to také naznačit, že i některé vlastnosti budou po zobecnění platit naprosto stejně jako pro konkrétní $d = 2$.



(a) J_c, kde $d = 2$ a $c = -0{,}125 + 0{,}716i$.

(b) J_c, kde $d = 3$ a $c = \frac{1}{2} + \frac{1}{2}i$.

Obrázek 4.1: Konjugovanost Juliových množin $z^d + c$ s volbou c (levá část obrázku) a $\overline{c}$ (pravá část obrázku).

4.3.1 Vlastnosti

V této sekci se podíváme na vlastnosti Juliových množin funkce $f(z) = z^d + c$. Místo formálních důkazů jednotlivých tvrzení však u většiny odůvodníme pravdivost vlastními slovy za podpory vizuálních ilustrací.

Symetrie

V bakalářské práci jsme se již o vlastnosti symetrie zmínili, ale pouze jako o určitém grafickém výsledku bez hlubšího zkoumání. V této části proto rozebereme symetrii podrobněji. Budeme uvažovat také neceločíselné exponenty.

Následující věta je také velmi známá pro Juliovu množinu kvadratické funkce. Nyní ji jen zobecníme a lehce přeformulujeme. Tato věta nám následně mimo jiné pomůže i při některých následujících větách a rozborech vlastností Juliových množin. Připomeňme, že konjugovanost (komplexní sdruženost) definujeme pro $z = x + iy$ jako $\overline{z} = x - iy$ a říkáme, že číslo $\overline{z}$ je číslo konjugované (komplexně sdružené) k z.

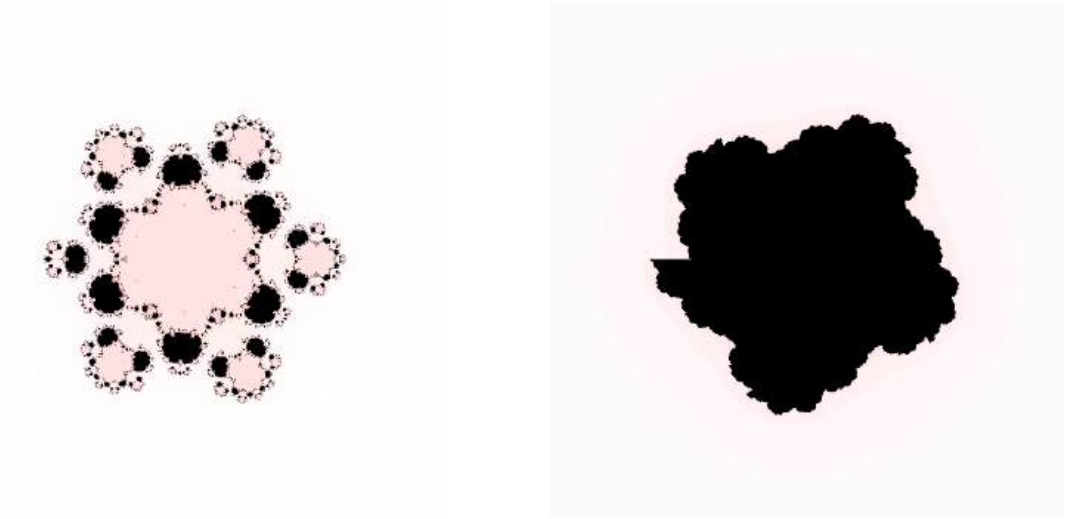
Věta 4.3.1. *Juliova množina $f(z) = z^d + c$ je konjugovaná k Juliově množině*

$$\tilde{f}(z) = z^d + \overline{c}.$$

Pro lepší představu opět využijeme grafické vizualizace, tentokrát na obrázku 4.1 máme Juliovu množinu, kde parametr c je vzhledem k jednotlivým množinám konjugovaný. Jak jde názorně z obrázku vidět, tak po vykreslení stejné Juliovy množiny jen pro $\overline{c}$, se výsledný obrázek „převrátí“ podél reálné osy.

Nyní přejděme k symetrii rotační. Tato vlastnost se bude projevovat zejména pro větší d a speciálně pak pro $d \in \mathbb{Q}$, kde nastanou určité komplikace. Začněme však s $d \in \mathbb{Z}$, pro které bude vše v pořádku.

Věta 4.3.2. *Juliova množina $f(z) = z^d + c$ pro $d \in \mathbb{Z}$ vykazuje $d-$násobnou rotační symetrii podle počátku.*



(a) J_c pro $z^{5,75} + 0{,}7$ **(b)** J_c pro $z^{4,5} + 0{,}5i$

Obrázek 4.2: Juliova množina funkce $z^d + c$ pro různou volbu d a c.

Z čehož dostáváme i následující důsledek, který říká, že pokud tedy rozdělíme komplexní rovinu na d oblastí, každá o úhlové velikosti $2\pi/d$, tak každá vytvořená oblast bude vypadat stejně.

Důsledek 4.3.1. *Juliova množina J_c, kde $d \in \mathbb{Z}^+$, je symetrická vůči rotaci o úhel $2\pi/d$ kolem počátku. Tedy obsahuje d identických oblastí, z nichž každá je obsažena v části s úhlem $2\pi/d$.*

Můžeme tedy tvrdit, že Juliova množina bude invariantní vůči rotaci o úhel $2\pi/d$ a navíc bude vykazovat požadovanou rotační symetrii.

Nyní se nám bude hodit pojem *základní dualita*, který jsme definovali pro $d = 2$ (viz 4.2.1). Tato vlastnost se dá bez problému rozšířit i obecně pro $d \in \mathbb{Z}$. Tedy buď je Juliova množina souvislá oblast, nebo je zcela nesouvislá. Tyto dva různé výsledky můžeme rozlišit na základě chování orbity kritického bodu z (kde $f'(z) = dz^{d-1} = 0$, což v našem případě znamená $z = 0$). Pokud zůstane orbita omezena, bude se jednat o souvislou Juliovu množinu, naopak pro orbitu divergující k ∞ bude platit, že příslušná Juliova množina bude zcela nesouvislá.

Nicméně, pokud se budeme bavit o $d \in \mathbb{Q}$, tak i když kritická orbita neunikne do nekonečna, Juliova množina nemusí být souvislá nebo dokonce křivkou. Tento fakt si ilustrujme na obrázku 4.2. Jak můžeme vidět, tak v levé části obrázku 4.2a Juliova množina nesplňuje základní dualitu, jelikož není ani souvislá, ani zcela nesouvislá, ale má konečné množství oblastí. Na pravé straně 4.2b si pak lze všimnout určitých skoků. Lze tedy říci, že J_c jako hranice dokonce není v tomto případě ani křivkou.

Skoky v Juliových množinách

Na obrázku 4.2b jsou viditelné skoky, které jsou způsobeny následujícím faktem. Když z přechází přes zápornou reálnou osu, hodnota $\arg(z)$ mění větev, takže z^d

může ležet na jedné z několika různých větví. U polárních souřadnicích je snadné sledovat, na které větvi se z^d nachází. Bohužel v našem případě, při přidání c, se musíme vrátit ke klasickým souřadnicím. To v reálu způsobí, že algoritmus pro vykreslení ztrácí přehled o větvích.

Skoky v Juliových množinách závisí na $|Im(c)|$. Uvažujme okolí N bodu z, kde $z^d+tc = 0$ s $0 \leq t \leq 1$. Po provedení iteračního procesu pro tento předpis se okolí N posunuje přes počátek. Může se tedy stát, že body v jedné části N překročí větvový řez (*branch cut*), například tak, že přejdou přes zápornou reálnou osu. Jenomže to pro body v jiné části N nemusí platit. Hranice mezi těmito dvěma částmi kopíruje přímku $z = tc$. Překročením řezu se změní jejich argument, což vede k tomu, že hodnoty iterací z pro body, které byly původně blízko, se začnou výrazně lišit. Výsledkem je, že Julia množina obsahuje skoky nebo viditelné nespojitosti.

Tyto skoky nastávají, pokud $Im(c) \neq 0$. Když $Im(c) = 0$, Juliova množina je symetrická podle reálné osy, protože $f(z)$ má reálné koeficienty. Proto se v Juliově množině pro tento konkrétní případ neobjevují žádné skoky.

Aplikace z^d+c může být brána jako násobení argumentu číslem d. Podívejme se na obraz záporné reálné osy po využití tohoto násobení. V celočíselném případě, konkrétně pro $z^2 + c$, tato operace odpovídá zdvojení úhlu. Stejně tak pro obecné d. Takže při každé iteraci roznásobíme $\arg(z)$ číslem d.

Symetrie pro racionální exponent d

Definujme filled Juliovu množinu a s ní spojené množiny pro obecné d.

Definice 4.2. „Filled“ Juliova množina, K_c, je definována jako uzávěr množiny bodů v $\mathbb{C}$, jejichž orbita funkce $f(z) = z^d + c$ nediverguje. Hranice této množiny je Juliova množina J_c. Fatouova množina F_c je doplněk Juliovy množiny. Doplněk K_c je otevřená množina bodů, pro které platí, že $f^n(z) \to \infty$ při $n \to \infty$, a označuje se $A(\infty)$.

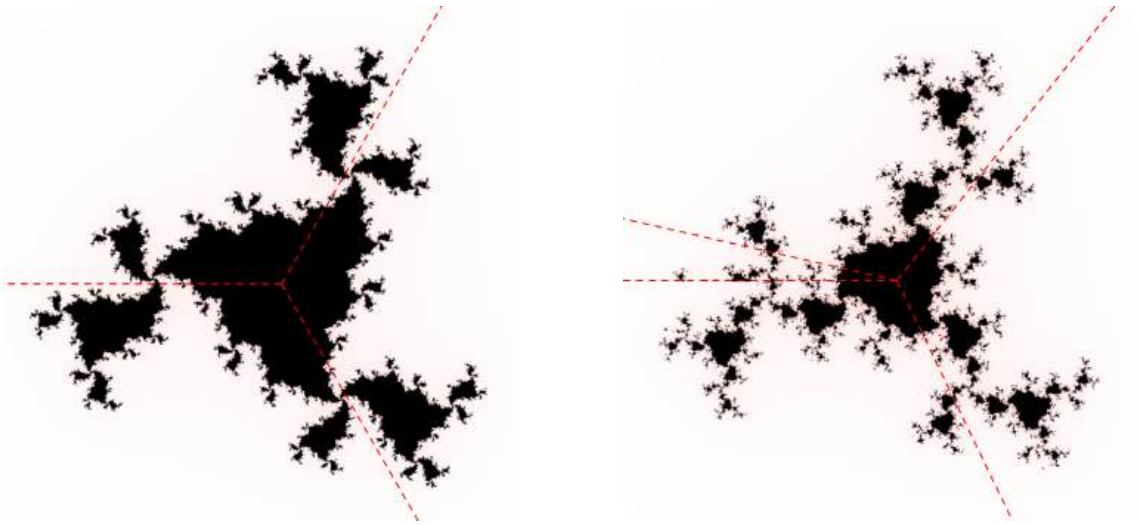
V případě polynomické funkce jsou tyto definice ekvivalentní s původními definicemi K_c a J_c. $A(\infty)$ je definováno jako oblast přitažení bodu ∞. Pro polynom platí, že $\partial A(\infty) = J_c$, kde ∂ označuje hranici množiny a $K_c = \mathbb{C} \setminus A(\infty)$. Juliova množina je obvykle definována jako doplněk Fatouovy množiny, což je množina bodů, u kterých $\{f^n(z)\}_{n=1}^{\infty}$ tvoří normální rodinu funkcí.

Věta 4.3.2 vysvětluje symetrii množin K_c a J_c pro exponent $d \in \mathbb{Z}$. Nyní představíme zobecněné tvrzení o symetrii množin K_c a J_c, kde d v tomto případě nabývá jak pozitivních (a negativních) celých čísel, tak i hodnot z oboru $\mathbb{Q}$ čísel.

Věta 4.3.3. *Juliova množina J_c, kde $d > 2$ je racionální nenulové číslo, je symetrická vůči rotaci o úhel $2\pi/d$ kolem počátku, ale je omezená větvením.*

Stejně jako v případě pro $d \in \mathbb{Z}$ můžeme tvrdit, že je Juliova množina pro $d \in \mathbb{Q}$ invariantní vůči rotaci, což lze zapsat korektně pomocí následující věty.

Věta 4.3.4. *Juliova množina funkce $f(z) = z^d + c$, kde $d = p/q$, je invariantní vůči rotaci o úhel $2\pi/d$, ale omezená větvením.*



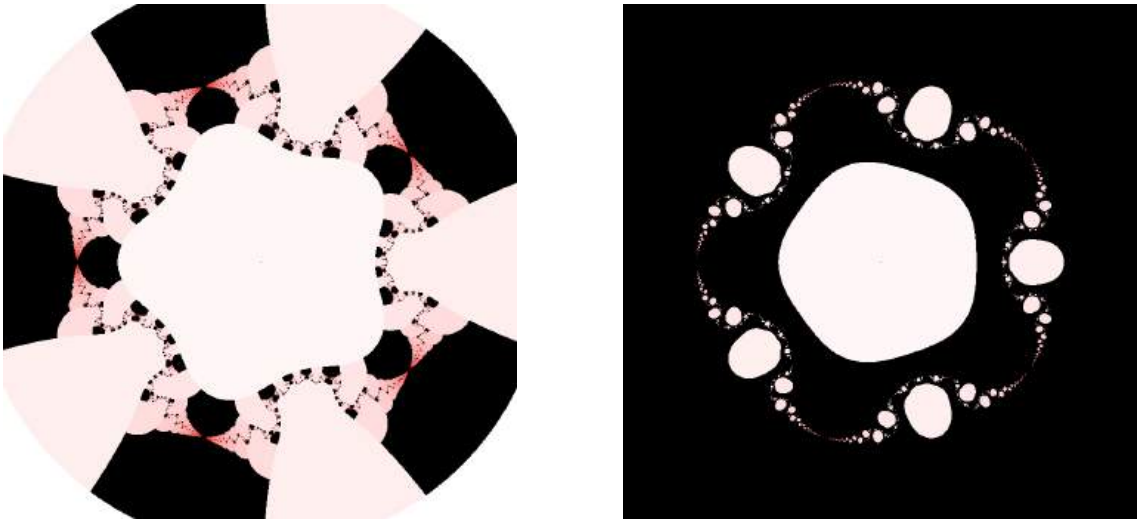
(a) J_c pro $z^3 + 0{,}596 + 0{,}5i$ **(b)** J_c pro $z^{3,11} + 0{,}596 + 0{,}596i$

Obrázek 4.3: Rotační symetrie Juliovy množiny funkce $z^d + c$ pro $d \in \mathbb{Z}$ (vlevo) a pro $d \in \mathbb{Q}$ (vpravo).

Takže pokud rozdělíme $\mathbb{C}$ na části o úhlu $\theta_d = 2\pi/d = 2\pi q/p$, pak každá taková oblast vykazuje stejné chování. Nicméně zde narážíme na problém. Když d patřilo do množiny celých čísel, vždy se komplexní rovina rozdělí na d stejně velkých částí. Bohužel pro d z racionálního oboru už nám celočíselný násobek těchto oblastí stačit nebude. Komplexní rovina se rozdělí na $\lfloor d \rfloor$ kompletních částí, jejichž dělení začíná zápornou částí reálné osy. Za poslední kompletní částí se bude nacházet neúplná část s úhlem menším než θ_d, která bude ale stále symetrická k příslušné části kompletních oblastí. Jelikož se nachází pod větvovým řezem, bude symetrická k „pravé“ části kompletní oblasti, protože „levá“ strana neúplné části se „ztratí“ v následujícím větvovém řezu. Takto bude Juliova množina invariantní vůči rotaci o úhel $2\pi/d$, omezená větvovým řezem a bude vykazovat požadovanou symetrii. Pro ilustraci viz 4.3, kde jsme uvedli Juliovu množinu jak pro d z $\mathbb{Z}$, tak i z $\mathbb{Q}$. Je zřetelně vidět, jak pro 4.3b je obrázek rozdělen na tři celočíselné části a zbylou, výrazně menší, nekompletní oblast.

Symetrie pro záporný exponent d

Nyní si uvedeme důsledky týkající se záporného exponentu pro z. Při výpočtu iterací bodu využíváme limitní poloměr divergence, tj. mimo jiné číslo, které má vlastnost, že pokud $|f^k(z)| > r$ pro nějaké k, pak $|f^n(z)| > r$ pro všechna $n > k$. Pro funkci $f(z) = z^{-d} + c = \left(\frac{1}{z^d}\right) + c$ je při malých $|z|$, $|z|^{-d}$ velké. Takže je poměrně obtížné definovat limitní poloměr divergence r_c. Proto je také vizualizace množin s tímto předpisem velmi závislá na určeném limitním poloměru, což můžeme vidět na obrázku 4.4, kde pro naprosto stejný předpis a počet iterací dostáváme velmi odlišné výsledky.



(a) J_c pro $r_c = 4$.

(b) J_c pro $r_c = 40$.

Obrázek 4.4: Juliova množina $z^5 - 1$ pro různý poloměr divergence r_c.

Nyní nechť funkce $f(z) = z^{-d} + c$ je zapsána ve tvaru

$$f(z) = \left(\frac{1}{z^d}\right) + c = \frac{cz^d + 1}{z^d},$$

kde $f(z)$ je racionální funkcí z. Ačkoliv se jedná o funkce, jejichž Juliovy množiny jsou dobře definovány, tyto funkce mohou mít ohraničené prstencovité oblasti Fatouovy množiny nazývané Hermanovy prstence (viz [1], [4]). To ztěžuje výpočet K_c.

Juliovy množiny pro $d \in \mathbb{Z}^-$ vykazují stejný typ symetrie jako pro $d \in \mathbb{Z}^+$, protože argument v 4.3.2 týkající se rotační symetrie stále platí. Stačí totiž nahradit exponent ve funkci $f(z)$ záporným číslem. Tedy poté platí i následující věta:

Věta 4.3.5. *Juliova množina funkce $f(z) = z^d + c$, kde $d \in \mathbb{Z}^-$, vykazuje d-násobnou rotační symetrii vůči počátku.*

Pokud se pokusíme rozšířit tuto definici na neceločíselné hodnoty, narazíme na stejný problém jako dříve. Nemůžeme definovat K_c jako část $\mathbb{C}$, která nediverguje kvůli prstencovitým oblastem, které se v f mohou vyskytnout. Pro celočíselné hodnoty definice platí z hlediska normálních rodin funkcí, ale to se bohužel v tomto případě nedá rozšířit na neceločíselné hodnoty. Tím pádem bohužel nemůžeme tvrdit to stejné jako ve větě 4.3.4 pro záporné exponenty d racionálních hodnot.

Limitní poloměr divergence

Nyní představíme lemma, které nám poskytne klíčovou podmínku pro počítačové generování Juliových množin. Jedná se o vztah pro limitní poloměr divergence r_c. Vzhledem k důležitosti samotného tvrzení si ho následně i dokážeme.

Lemma 4.3.1. *Pokud* $|z| > 2 + |c|$*, pak* $|f(z)| > |z|$.

Důkaz. Nechť $|z| > r_c = 2 + |c|$. Poté platí:

$$\begin{aligned} |f(z)| &= |z^d + c| \\ &= |z^d - (-c)| \\ &\geq |z^d| - |-c| \\ &\geq |z|^d - |c| \end{aligned}$$

Proto potom také:

$$|f(z)| \geq |z|\left(|z|^{d-1} - \frac{|c|}{|z|}\right).$$

Jelikož z předpokladu víme, že $|z| > 2 + |c|$, dostáváme pak:

$$|z|^{d-1} - \frac{|c|}{|z|} > (2 + |c|)^{d-1} - \frac{|c|}{2 + |c|}.$$

Dále, pro $d > 2$ platí $(2 + |c|)^{d-1} > 2$, což implikuje:

$$|f(z)| > |z|.$$

□

Všimněme si, že to znamená, že pokud $|f^n(z)| > r_c$ pro nějaké $n \in \mathbb{N}$, potom $|f^m(z)| > r_c$ pro všechna $m \geq n$. Díky tomu můžeme číslo $r_c = 2 + |c|$ nazvat limitním poloměrem divergence pro $f(z)$ a používat jej jako rozhodovací podmínku pro klasifikaci bodů v samotném vizualizačním algoritmu.

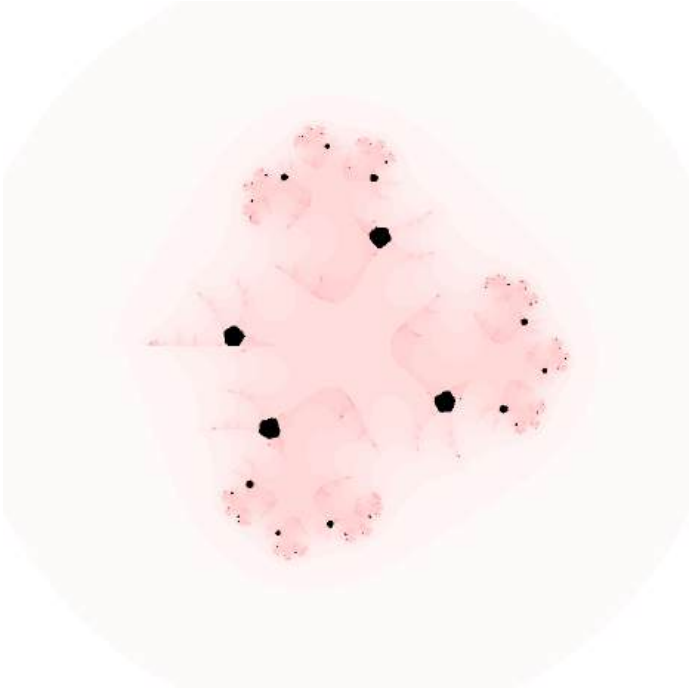
Souvislost

Věta 4.3.6. *Pro libovolný polynom* P *platí: pokud* $P^n(q) \to \infty$ *pro každý kritický bod* q*, potom je Juliova množina* J_c *úplně nesouvislá.*

Při pokusu přizpůsobit tuto větu rodině funkcí pro $d \in \mathbb{Q}$ narážíme na problém spojený se základní dualitou. Jak už bylo zmíněno výše, pro kritickou orbitu jdoucí k ∞ nemusí být Juliova množina úplně nesouvislá pro exponent z $\mathbb{Q}$. Příklad takové množiny uvedeme níže.

Příklad 2. Mějme funkci $f(z) = z^{3.5} - 0{,}55 + 0{,}21i$, u které jsme výpočtem ověřili, že kritická orbita po několika iteracích diverguje. Pokud se podíváme na pevné body pro tento předpis, vyjdou nám čtyři:

$$\begin{aligned} z_1 &= -0{,}63875 - 0{,}759457i, \\ z_2 &= -0{,}597901 + 0{,}0497261i, \\ z_3 &= -0{,}458959 + 0{,}649881i, \\ z_4 &= 1{,}17037 - 0{,}0502321i, \end{aligned}$$



Obrázek 4.5: Juliova množina funkce $z^{3,5} - 0{,}55 + 0{,}21i$.

Nyní zjistíme, který z těchto pevných bodů je přitahující. To uděláme pomocí derivace v daném bodě $f'(z) = 3{,}5z^{2,5}$. Vidíme, že $|f'(z_2)| = 0.97585 < 1$. Měla by tedy existovat malá oblast kolem z_2, která je přitahována k z_2 namísto „úniku“ do nekonečna při opakovaném aplikování f. Jak vypadá taková Juliova množina viz 4.5.

4.4 Juliovy množiny transcendentní funkce

V této sekci se podíváme blíže na Juliovy množiny transcendentních funkcí. Jako první zadefinujeme, co znamená být *transcendentní* funkce a v návaznosti na to pak definujeme Juliovu množinu těchto funkcí. V druhé části se zaměříme na tzv. Ishikawovu iteraci, pomocí které budou samotné fraktály vizualizovány. Společně s ní zavedeme také limitní poloměr divergence pro tyto funkce, což nám zejména u generování velmi ulehčí práci.

4.4.1 Transcendentnost

Nejprve definujme holomorfní funkci a z toho pak vyplývající transcendentnost.

Definice 4.3. Řekneme, že funkce f je *holomorfní* v bodě $z_0 \in \mathbb{C}$, jestliže existuje okolí bodu z_0, v jehož každém bodě je funkce f komplexně diferencovatelná. Řekneme, že f je holomorfní na množině $M \subseteq \mathbb{C}$, je-li holomorfní v každém bodě $z \in M$.

Definice 4.4. Funkce $f : \mathbb{C} \to \mathbb{C}$ se nazývá *celá*, jestliže je holomorfní na celém $\mathbb{C}$. Celé funkce, které nejsou polynomy, se nazývají *transcendentní*.

Teď už se můžeme přesunout přímo k samotné definici Juliovy množiny transcendentní funkce.

Definice 4.5. Nechť f je transcendentní celá funkce a nechť f^n označuje n-tou iteraci funkce f. Potom jsou Fatouova množina F_c a Juliova množina J_c funkce f definovány následovně:

$$F_c := \{z \in \mathbb{C} \mid \{f^n\}_{n=1}^{\infty} \text{ je normální rodina v okolí } z\},$$
$$J_c := \mathbb{C} \setminus F_c.$$

Jinými slovy lze říci, že Juliova množina pro takové funkce je uzávěr množiny bodů, jejichž orbity „utíkají“ k nekonečnu.

4.4.2 Ishikawova iterace

Definujme *Ishikawovu iteraci*, pomocí které budeme generovat Juliovy množiny pro $f(z) = \lambda e^z$. Jedná se o obecnější variantu Picardovy iterace a dovolí nám využití většího množství parametrů, čímž dosáhneme ještě většího množství možností výsledného fraktálu.

Definice 4.6. Nechť $\mathbb{C}$ je množina komplexních čísel a $f : \mathbb{C} \to \mathbb{C}$. Pro bod $z_0 \in \mathbb{C}$ sestrojme posloupnost $\{z_n\}$ následujícím způsobem:

$$z_{n+1} = \alpha f(v_n) + (1 - \alpha) z_n,$$

kde v_n představuje transformaci

$$v_n = \beta f(z_n) + (1 - \beta) z_n,$$

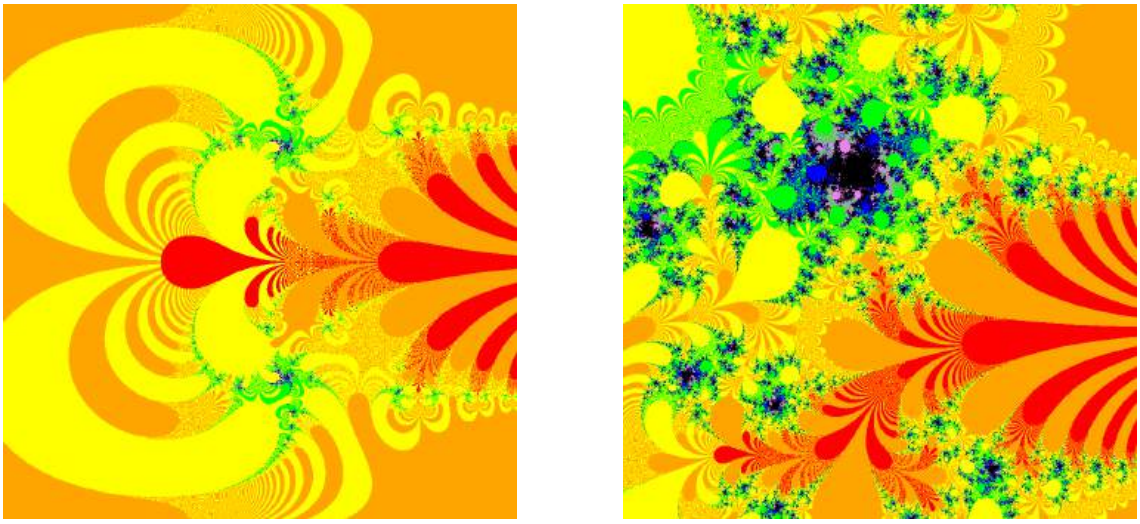
pro $n = 1, 2, 3, \ldots$, kde $0 < \alpha \leq 1$, $0 \leq \beta \leq 1$. Potom posloupnost $\{z_n\}$ sestrojená výše se nazývá Ishikawova iterace pro všechny body orbity bodu z_0. Speciálně volbou $\alpha = 1, \beta = 0$ se pak jedná o Picardovu iteraci viz [29].

Z pohledu počítačové grafiky bohužel nejsme schopni iterovat pro $n \to \infty$. Musíme proto pro obrazy Juliových množin celých funkcí, jako jsou

$$z \mapsto \lambda e^z, \quad z \mapsto \lambda \cos(z), \quad z \mapsto \lambda \sin(\lambda z),$$

hledat body, jejichž orbity se vzdalují od počátku. Čím dříve takové orbity najdeme a klasifikujeme, tím rychleji budeme schopni fraktální obrazce vykreslovat. Obecně se orbity vzdalují do nekonečna v konkrétních směrech. Pro $f(z) = \lambda e^z$ orbity směřují do nekonečna v kladném směru reálné osy. To znamená, že:

$$\lim_{n\to\infty} |f^n(z)| = \infty, \quad \text{pak také} \quad \lim_{n\to\infty} \mathrm{Re}(f^n(z)) = \infty,$$



(a) J_c pro $\lambda = 1 + 0i$. **(b)** J_c pro $\lambda = 1 + 1i$.

Obrázek 4.6: Symetrie Juliových množin funkce λe^z pro parametry $\alpha = \beta = 1$.

U $\lambda \sin(z)$ a $\lambda \sin(z)$ je situace poněkud odlišná. Tady naopak platí, že pokud

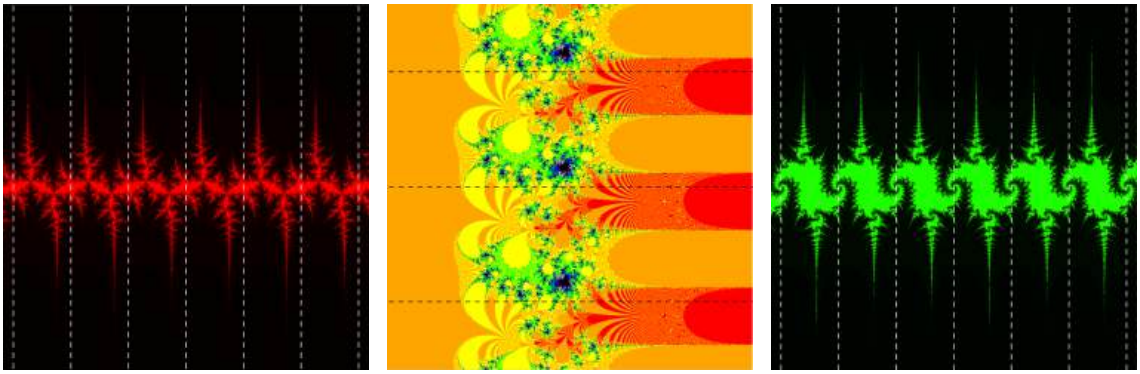
$$\lim_{n\to\infty} f^n(z) = \infty, \quad \text{pak také} \quad \lim_{n\to\infty} |\mathrm{Im}(f^n(z))| = \infty,$$

Takže zde naopak orbity směřují do nekonečna ve směru kladné či záporné části imaginární osy. Nyní využijeme podmínky, které uvádí autoři v [27]. Pro $f(z) = \lambda e^z$ pokud reálná část z $f(z)$ překročí hodnotu 50, pak můžeme s jistotou tvrdit, že orbita z diverguje. Podobně pro $f(z) = \lambda \sin(z)$ a $f(z) = \lambda \cos(z)$ pokud zjistíme, že velikost imaginární části překročí hodnotu 50. Tento fakt nám velmi pomůže v následné vizualizaci. Na obrázku 4.7 jsou uvedeny příklady Juliových množin pro jednotlivé transcendentní funkce.

4.4.3 Vlastnosti

Symetrie

Ukažme si vlastnost Juliových množin, která se bude konkrétně pro $\sin(z)$ a $\cos(z)$ v mnoha ohledech shodovat s vlastnostmi množin pro polynomické zobrazení. Řeč je o symetrii. Například tvrzení o středové symetrii zde platí úplně stejně a také speciálně pro $\lambda_y = 0$, jakožto imaginární složku λ, platí navíc i osová souměrnost podle reálné osy. U λe^z platí stejně tak osová symetrie podle reálné osy v případě nulové imaginární části λ_y. Avšak pokud $\lambda_y \neq 0$, středová symetrie zde neplatí (viz 4.6), kde je zobrazen stejný výřez komplexní roviny pro Juliovy množiny funkce $f(z) = \lambda e^z$. Můžeme si všimnout, že obrázek 4.6b se „deformuje“ směrem k dolnímu levému rohu, což je zapříčiněno právě nenulovou imaginární částí λ_y.



(a) J_c pro $(1+1i)\cos(z)$ s periodou π.

(b) J_c pro $(1+1i)e^z$ s periodou $2\pi i$ $(\alpha=\beta=1)$.

(c) J_c pro $(1{,}17+0{,}43i)\sin(z)$ s periodou π.

Obrázek 4.7: Periodicita Juliových množin transcendentních funkcí znázorněná pomocí přerušovaných čar na výřezu $[-10;10]$ na Re a $[-10i;10i]$ na Im ose.

Periodicita

V sekci 1.0.1 jsme se také zmínili, že exponenciální funkce je periodická s periodou $2\pi i$. Tato vlastnost ale například z obrázku 4.6 úplně zřejmá není. Problém je totiž ve velikosti výřezu komplexní roviny, pro který se Juliova množina λe^z vykresluje. Konkrétně v našem případě se jedná o výřez $[-3;3]$ na reálné ose a $[-3i;3i]$ na ose imaginární. Jenomže to znamená, že pokud bychom chtěli periodicitu pozorovat, bude potřeba zvětšit výřez minimálně na dvojnásobek, jelikož se množina začne opakovat až od hodnot $6{,}28i$, resp. $-6{,}28i$ na imaginární ose. Na obrázku 4.7 můžeme vidět výřezy Juliových množin odpovídajících rozsahu $[-10;10]$, resp. $[-10i;10i]$, čímž jsme zajistili, aby bylo periodické chování zaručeně vidět. Přidali jsme také obrázky funkcí $\lambda\cos(z)$ (viz 4.7a) a $\lambda\sin(z)$ (viz 4.7c), kde nás periodicita nepřekvapí. Na druhou stranu můžeme vidět, že se zde fraktální obrazec opakuje s periodou π, což je rozdíl od reálného případu, kdy se tyto funkce opakují s periodou 2π. Pro osy, které oddělují jednotlivé periodicky se opakující oblasti, platí, že první prochází počátkem buď ve svislém, nebo vodorovném směru a od ní se s určitou periodou odvíjí ostatní.

Kompaktnost

Pokud bychom měli testovat vlastnost kompaktnosti, která platila u Juliových množin polynomické funkce, tak už z obrázku 4.7 použitého k ilustraci předchozí vlastnosti se zdá, že tato vlastnost pro transcendentní funkce platit nebude. Potvrďme si tuto domněnku následující větou, která definuje kompaktní množinu takto:

Věta 4.4.1. *Množina $J \subseteq \mathbb{C}$ se nazývá kompaktní právě tehdy, když je uzavřená a omezená.*

Na první pohled vidíme, že podmínka omezenosti určitě splněna není. Jelikož se části periodicky opakují, nejsme ji schopni celou ohraničit libovolně velkým

kruhem se středem v počátku. Z toho plyne, že podle věty 4.4.1 opravdu nesplňujeme všechny podmínky pro kompaktní množinu a kompaktnost tedy můžeme vyloučit.

Kapitola 5

Webová aplikace

Tato kapitola popíše webovou aplikaci *Fractal Generator* (dostupná na https://www.math.muni.cz/~xmacharacekj/), která byla vytvořena jako praktická část diplomové práce. Hlavním cílem této aplikace je poskytnout intuitivní a vizuálně přívětivý nástroj, který propojuje teoretické poznatky s praktickými vizualizacemi komplexních systémů. Je kladen důraz na vysokou interaktivitu aplikace, ať už se bude jednat o volbu maximálního počtu iterací, počátečních parametrů nebo také například volbu barevného spektra výsledného fraktálu. Aplikace také nabízí možnost hlubšího zkoumání fraktálů za pomoci zoomu a pohybu ve vygenerovaném obrázku. Dále je kladen důraz na uživatelsky přátelské prostředí. Měl by tak být schopen ovládat jednotlivé nástroje nejen člověk znalý v oboru, ale také někdo, kdo by se s danou problematikou teprve rád seznámil.

5.1 Technologie použité při vývoji aplikace

Jelikož se jedná o rozšíření webové aplikace z bakalářské práce (viz [23]), není nutné znovu podrobně rozebírat volbu kombinací skriptovacích jazyků a jejich výhod, jelikož stejně jako v bakalářské práci je i webová aplikace této diplomové práce naprogramována pomocí HTML, CSS a JavaScriptu[1] s dvěma externími knihovnami MathJax a Math.js. Co se týče vývojářského prostředí, bylo zvoleno opět Visual Studio Code.

Novinkou je však využití Bootstrapu. Bootstrap je jedním z nejrozšířenějších front-endových frameworků. Mezi jeho klíčové vlastnosti patří především snadná tvorba responzivního designu díky jeho grid systému, který umožňuje přizpůsobení vzhledu aplikace jak mobilním zařízením, tak i velkým obrazovkám. Je zde předdefinováno spousta prvků. V této webové aplikaci bylo využito hlavně předdefinovaného vzhledu například u tlačítek nebo políček formuláře. Je tak zajištěna vyšší konzistentnost celé aplikace, což zlepšuje její estetický vzhled. Na druhou stranu je možnost všechny předdefinované prvky také upravovat a měnit. Ať už se jedná o stylizaci, či jejich funkčnost. Vzhled aplikace je proto i tak originální. Jinými slovy, Bootstrap žádným způsobem neomezí výsledný vzhled aplikace, ale

[1] V případě zájmu o více informací viz [23].

dopomůže dosáhnout požadovaného výsledku v rychlejším čase. Jelikož se stále primárně jedná o matematickou diplomovou práci, vývoj grafického rozhraní nebyl hlavním cílem, a použití Bootstrapu tak umožnilo soustředit se na klíčové části aplikace, aniž by se muselo věnovat nadměrné množství času stylizaci uživatelského prostředí. Záměr tohoto kroku je hlavně zajištění, aby vizuální stránka aplikace odpovídala kvalitě její funkčnosti a aby nedostatky v designu nesnižovaly celkový dojem z aplikace.

5.2 Strukura aplikace

Webová aplikace *Fractal Generator* je nástroj zaměřený na vizualizaci a zkoumání fraktálů a s nimi spojených matematických vlastností. Její hlavní účel spočívá v poskytnutí interaktivního prostředí pro generování, analýzu a experimentování s Juliovými a Mandelbrotovými množinami a jejich zobecněnými variantami. Jednou z významných částí aplikace je sekce *Hyperbolic Components*, která se zabývá problematikou hyperbolických komponent Mandelbrotovy množiny a jejich chováním. Dále aplikace nabízí v sekci *Coloring Methods* množství obarvovacích metod jak pro Juliovy množiny, tak pro množinu Mandelbrotovu. Generováním fraktálů se zabývá rovněž sekce *Rendering algorithms*, která představuje různé vizualizační algoritmy, přičemž se zaměřuje zejména na variabilitu cest, jak dospět k výslednému fraktálu, než na metodu jejich obarvování, jak tomu bylo v předchozí sekci. Součástí aplikace je také nástroj *Mandelbrot Set Explorer*, který uživatelům poskytuje možnost interaktivního zkoumání Mandelbrotovy množiny prostřednictvím zoomování, posouvání a podrobného prohlížení jednotlivých částí fraktálu. Mezi další sekce patří *Generalized Mandelbrot Set* a *Generalized Julia Set*, které zobecňují tradiční přístup generovat množiny pouze pomocí kvadratického zobrazení ($z \mapsto z^2 + c$) a poskytují možnost generovat fraktály funkcí s obecným exponentem a nebo na základě trigonometrických či exponenciálních funkcí. Poslední část s názvem *Buddhabrot* je jak už název napovídá věnována Buddhabrot množině, což je specifická vizualizace Mandelbrotovy množiny, při které je kladen důraz na trajektorie bodů při iteracích.

Celkově je aplikace určena nejen pro vizualizaci fraktálů, ale také pro pochopení principů, které je definují. Každá sekce totiž obsahuje mimo jiné i stručnou teorii pro danou vizualizaci. Aplikace tak může posloužit nejen jako výuková pomůcka pro lepší pochopení problematiky,ale díky logické struktuře a jednoduchosti ovládání si zde přijde na své i širší veřejnost.

5.2.1 Popis jednotlivých sekcí

Úvod

Tato první část aplikace má primárně za úkol uživateli představit aplikaci. Na co se aplikace zaměřuje a její základní popis. Následně se v této sekci také uživatel seznámí se základní teorií, která je nezbytná pro bezproblémové používání.

Hyperbolic Components

Sekce zabývající se hyperbolickými komponentami je stavěna odlišně od všech ostatních. Není zde možnost vizualizace pouze výsledného produktu, ale vizuální interpretace jsou zde možné již při teoretickém základu, z kterého hyperbolické komponenty vycházejí. Jelikož je těmto komponentám v diplomové práci věnována jedna celá kapitola, je vhodné jim i v aplikaci věnovat dostatek prostoru.

Jako první je zde znázorněna jedna z důležitých teoretických vlastností 2.2.2. Pomocí slideru *Period* si uživatel může zobrazovat jednotlivé komplexní odmocniny z jedné na jednotkové kružnici a sledovat jejich transformaci do kardioidu.

V druhém interaktivním poli je pak opět pomocí slideru *Period* možnost si volbou dané periody zobrazit bifurkační body kardioidu a jeho přilehlých hyperbolických komponent odpovídajících zvolené periodě.

Třetí pole se zabývá oblastmi přitažení. Konkrétně jak nám tyto oblasti mohou pomoci k nalezení středů hyperbolických komponent. Je zde možnost zoomování a také pohybu po vykreslených oblastech. Je tak možno hlubšího prozkoumání těchto oblastí. Jako vizuální důkaz toho, že nám oblasti přitažení pomohou najít středy hyperbolických komponent, slouží následující funkcionalita. Po kliknutí na vykreslený obrázek dokáže rozpoznat, ve které oblasti se kurzor nachází a spočítat tak periodu středu komponenty, kterému odpovídá zvolená oblast přitažení a na závěr tento střed vykreslí ve formě červeného bodu. Pro jednodušší představu, v jaké části Mandelbrotovy množiny se nacházíme, je zde umístěn slider *Transparency*, kterým si uživatel může nastavit průhlednost stínu Mandelbrotovy množiny, a tak jednoduše pozorovat pozice oblastí přitažení a samotné Mandelbrotovy množiny. Mimo jiné je zde také jako u většiny následujících sekcí pole s názvem *Rendering Time*, které nám říká, jak rychle se dané vykreslení či změny provádí. Další pole *Zoom* pak uživateli napovídá, kolikanásobně je obrázek přiblížen vzhledem k počátečnímu vykreslení.

Čtvrté a zároveň poslední pole této sekce je už samotné vykreslení komponent s periodou, kterou si uživatel sám zvolí pomocí slideru *Period*. Všechny komponenty se pak po zmáčknutí tlačítka *reload* zvýrazní červeně. Nechybí zde opět ani pole *Rendering Time*. Další políčko *Number of hyperbolic components* pak říká, kolik celkem existuje hyperbolických komponent dané periody. Totiž vzhledem k jejich velikostem, zejména u vyšších period, není pouhým okem možné všechny pozorovat.

Coloring Methods

Pro větší přehlednost je tato sekce rozdělena na dvě části. V první jsou obarvovací metody využity zvlášť pro Juliovy množiny a ve druhé pro Mandelbrotovu množinu. Není nutné zde popisovat parametry a ovládací prvky pro jednotlivé metody, jelikož jsou pro každou metodu lehce odlišné. Bude dostačující popsat pouze princip, jak tento blok funguje.

Sekce je ve formě jednoho univerzálního vizualizačního pole, kde jako první uživatel vybere metodu, kterou chce fraktál vykreslit. V závislosti na výběru se dynamicky upravuje panel parametrů jako je maximální počet iterací (*Max iterati-*

ons), limitní poloměr divergence (*Escape Radius*) nebo například v případě Juliových množin také konstanta c. U metody *Smooth Coloring* je možnost také pomocí tlačítka automaticky přepínat mezi „plynulými“ a „neplynulými“ přechody barev. Metoda *Multithreading* pak má speciální slider *Number of Workers*, což představuje počet „pracovníků“ (vláken), kteří výsledný obraz zpracovávají. Samozřejmě nesmí chybět ani pole *Rendering Time* pro rychlost vykreslení. Vše je nastaveno tak, že parametry z jedné metody se zachovávají i po přepnutí na další metodu. Je tak velmi jednoduché porovnat metody nejen z hlediska vizuálního, ale také co se týče rychlosti vykreslení například pro stejný počet iterací.

Rendering Algorithms

Stejně jako předchozí sekce je i tato rozdělena na část pro Juliovy množiny a pro Mandelbrotovu množinu. Co se týče principu fungování, je zcela stejný jako v předchozí sekci. Nachází se zde opět jedno univerzální vizualizační pole s měnícím se panelem parametrů v závislosti na algoritmu. Navíc se zde ve většině případů ale vyskytuje color picker, kde uživatel vybere *Low color* a *High color*, které představují počáteční a koncový odstín barvy, kterou se výsledný fraktál vykreslí. Navíc u *Rectangle checking* algoritmu je možnost skrytí čtverců, pomocí kterých se fraktál generuje a volba maximální velikosti čtverce.

Mandelbrot set Explorer

Jedná se asi o nejvíce interaktivní část celé aplikace. Opět je zde jedno vizualizační pole, v kterém aplikace nabízí hlubší prozkoumání Mandelbrotovy množiny. Pomocí zoomu a také pohybu může uživatel pozorovat fraktální struktury, které při počátečním vykreslení množiny nejsou pouhým okem viditelné. Zadáním hodnot do políček *Real part of the centre* a *Imaginary part of the centre* se automaticky upraví střed výřezu pole. Tento střed je znázorněn jako červený bod. A naopak posunem vykresleného výřezu množiny, a tedy zároveň změnou pozice středu vůči Mandelbrotově množině, se po ukončení pohybu automaticky změní hodnoty v políčkách. V bodě odpovídající středu aplikace určí hodnotu *Period*, tedy periodu komponenty Mandelbrotovy množiny. Nachází se zde ještě také pole *Rendering Time* a *Zoom*, jejichž funkce jsou popsány výše.

Generalized Mandelbrot set

Sekce zaměřená na vizualizaci zobecněné formy Mandelbrotovy množiny. Konkrétně Mandelbrotovy množiny s obecným exponentem (množina $\mathbb{Q}$ čísel) a její souvislost s logistickým zobrazením. Ovládání je opět velmi intuitivní. Po vybrání jedné ze dvou dostupných metod se znovu zpřístupní panel s volitelnými parametry a následně se pro předvolené hodnoty polí *Exponent*, *Low color*, *High color* a *Max iterations* vykreslí odpovídající fraktální obrazec. Změna přednastavených hodnot je opět čistě na uživateli, a tak si vzhled Mandelbrotovy množiny může jednoduše přizpůsobit.

Generalized Julia set

Velmi podobná část jako ta předchozí, jen má pro větší přehlednost svou vlastní pasáž. Uživatel má možnost vizualizovat tentokrát zobecněné formy Juliových množin, ať už se jedná o funkce s obecným exponentem (množina $\mathbb{Q}^+$ čísel) nebo také trigonometrické a exponenciální funkce. Princip již není potřeba znovu podrobněji popisovat, jedná se totiž stále o to samé. Uživatel zvolí parametry, podle kterých se vykreslí daný obrazec.

Buddhabrot

Poslední část webové aplikace je věnována Buddhabrot množině. Jedná se o jiné pojetí vizualizace Mandelbrotovy množiny, o kterém bude psáno podrobněji níže. Jako vždy se zvolí dané parametry a vykreslí se obrazec.

Kapitola 6

Implementace algoritmů a metod

V této kapitole rozebereme jednotlivé vizualizační metody a algoritmy. Jelikož se jedná o obsáhlou kapitolu s velkým množstvím různých metod a algoritmů, tak i množství zdrojů, které byly potřeba pro sepsání této kapitoly bude vyšší. Hlavní literatura pro obarvovací metody jsou texty [5],[10], [31] a [36]. Pro jednotlivé algoritmy byly využity tyto zdroje [7], [9], [12] a [24]. Dále také [20], [25] a [27], které kromě praktické stránky věci poskytnou i množství teorie nejen ohledně algoritmů. Pro úplnost byly ještě na nějaké drobnosti využity texty [17], [23], [26] a [33]. Nejprve se podíváme na oblast algoritmů, která stojí za interpretací teoretických vlastností popsaných výše v kapitole 3. Následně představíme metody a algoritmy nové či optimalizované z bakalářské práce. Hlavním cílem této kapitoly je poskytnout čtenáři pochopení principů fungování jednotlivých kódů, nikoli prezentovat jejich kompletní podobu nebo propojení s HTML dokumenty a dalšími částmi aplikace. Funkce jsou proto často mírně upraveny oproti originálu, aby obsahovaly pouze klíčové části důležité pro pochopení daného tématu. Tento přístup má za cíl zdůraznit hlavní myšlenky a logiku bez zbytečných detailů.

6.1 Hyperbolic Components

Začneme stejně jako v aplikaci částí zabývající se hyperbolickými komponentami. Složitost tohoto tématu vyplývá přímo z teorie. Při zvyšující se periodě se velmi rychle zvyšuje také výpočetní náročnost. Potvrzuje to také fakt, že obecně výpočty kořenů polynomů se používají k testování výkonu počítače. Je proto zřejmé, že pokud nám v aplikaci půjde zejména o uživatelskou přívětivost, není úplně možné vyvinout program pro vizualizaci hyperbolických komponent striktně přímo z teorie, ale bude potřeba některé části optimalizovat, aby se i pro vyšší periody výsledek vykreslil v přijatelném čase.

6.1.1 Bifurkační body

Jako první však rozebereme vizualizace teoretických vlastností z 2.2.2 a 3. Začněme transformací jednotkového kruhu na kardioid. Přesněji transformací n-tých

odmocnin z jedné. Jedná se o přípravnou teoretickou vlastnost týkající se bifurkačních bodů kardioidu Mandelbrotovy množiny a hyperbolických komponent. V tomto kódu není ještě zohledněna vlastnost 3.2.3, a tak se zobrazují $n-$té komplexní odmocniny z jedné, ale také $k-$té komplexní odmocniny z jedné, pro k, kde $k|n$ a $k < n$. Nejprve pro jednotlivé úhly na jednotkovém kruhu `angle` spočítáme odpovídající reálnou část komplexního čísla `x` a imaginární část `y`.

```
for (let angle = 0; angle <= 2 * Math.PI; angle += 0.01) {
      const x = 0.5 * Math.cos(angle) - 0.25 * Math.cos(2 * angle);
      const y = 0.5 * Math.sin(angle) - 0.25 * Math.sin(2 * angle);
    }
```

Následně pomocí funkce pro zjištění periody orbity počátečního bodu $x + yi$ vybereme jen ty body, pro které odpovídá zvolená hodnota *Period* uživatelem.

```
function periods(x, y) {
 var list = [];
 var maxIter = 2000;
 var cx = math.complex(x, y);
 var z = math.complex(0, 0);
 list[0] = z;
 for (var iterace = 1; iterace < maxIter; iterace++) {
     list[iterace] = math.add(cx, math.multiply(list[iterace-1], list[
        iterace-1]));
 }
 var hit = false;
 for (var i = 0; i < list.length - 1; i++) {
     for (var j = i + 1; j < list.length; j++) {
         if (i > 1000 && Math.abs((list[i].re - list[j].re)) < 0.0001
            && Math.abs((list[i].im - list[j].im)) < 0.0001) {
             hit = true;
             break;
         }
     }
     if (hit == true) {
         break;
     }
 }
 return j - i;
}
```

Druhá vizualizace navazuje na tu první. Ke transformaci jednotkového kruhu na kardioid přidáme navíc transformaci na kruh s poloměrem $0{,}25$ a středem v -1, což odpovídá komponentně s periodou dva. Tím jsme také rozšířili možnost zobrazení bifurkačních bodů i pro tuto část Mandelbrotovy množiny.

```
for (let angle = 0; angle <= 2 * Math.PI; angle += 0.01) {
 const period2X = -1 + (1 / 4) * Math.cos(angle);
 const period2Y = (1 / 4) * Math.sin(angle);
}
```

Je zde také v pozadí vykreslena Mandelbrotova množina, na které jsou modře zvýrazněny právě kardioid a hyperbolická komponenta. Použita je stejně jako v předchozím kódu funkce `periods`, ale navíc se zde již zohledňují dělitelé n, kteří

ve výsledném obrázku už nebudou zobrazeni, a tak se po zvolení hodnoty *Period* uživateli skutečně ukážou pouze bifurkační body komponent s periodou n.

6.1.2 Oblasti přitažení a Newtonova metoda

U této vizualizace vycházíme z faktů v 3.3.

```
function basinOfAttr() {
  for (let j = 0; j < canvasAtom.height; j++) {
    for (let i = 0; i < canvasAtom.width; i++) {
      //...
      let z = { re: 0, im: 0 };
      const c = { re: x, im: y };
      let dc = { re: 0, im: 0 }; //derivative of c
      let mz = Infinity;  //minimum distance
      let m = 0; // period of basin of attraction
      let de = -1; //eccentricity of the point (used for brightness
         calculation)
      for (let k = 1; k < n; k++) {
        dc = {
          re: 2 * (z.re * dc.re - z.im * dc.im),
          im: 2 * (z.re * dc.im + z.im * dc.re)
        };
        z = {
          re: z.re * z.re - z.im * z.im + c.re,
          im: 2 * z.re * z.im + c.im
        };
        const z2 = cnorm(z);
        if (z2 < mz) {
          mz = z2;
          m = k;
        }
        if (z2 > r2) {
          de = 2 * Math.sqrt(z.re * z.re + z.im * z.im) *
               Math.log(Math.sqrt(z.re * z.re + z.im * z.im)) /
              (Math.sqrt(dc.re * dc.re + dc.im * dc.im) * dc0);
          break;
        }
      }
      // Rendering Basins of Attraction
    }
  }
}
```

Jak již bylo zmíněno v 5.2.1 je do kódu ještě zakomponována možnost překrýt oblasti přitažení Mandelbrotovou množinou a nastavit její průhlednost. Dále je zde také možnost zoomu a posunu. Tato funkcionalita bude však rozebrána až později v 6.3. Kódy týkající se posledních dvou funkcionalit není potřeba uvádět, protože se nejedná o funkcionality přímo související s hlavním tématem práce, ale jde spíše o pomocné nástroje. Naopak podstatnou funkcí je `componentCenter`. Jak už název napovídá, jedná se o funkci založenou na Newtonově metodě, která vypočítá pro zvolenou počáteční podmínku `c0` a maximální počet iterací `nMax` odpovídající střed komponenty periody `period`. Počáteční podmínku zvolí uživatel

kliknutím myši na libovolné místo vykresleného obrázku a zároveň tím zadefinuje proměnnou `period`, kterou kód dokáže zjistit.

```
function componentCenter(c0, period, mMax) {
  let c = c0;
  for (let m = 0; m < mMax; m++) {
    let z = { re: 0, im: 0 };
    let dc = { re: 0, im: 0 };
    for (let i = 0; i < period; ++i) {
      dc = {
          re: 2 * (z.re * dc.re - z.im * dc.im) + 1,
          im: 2 * (z.re * dc.im + z.im * dc.re)
        };  // dc = 2*z*dc + 1
      z = {
          re: z.re * z.re - z.im * z.im + c.re,
          im: 2 * z.re * z.im + c.im
        };  // z = z*z + c
    }
    c = {
        re: c.re -  complexDivide(z, dc).re,
        im: c.im - complexDivide(z, dc).im
      };  // c = c - z/dc
  }
  return c;
}
```

6.1.3 Nalezení hyperbolických komponent

Nyní se už můžeme přesunout k hlavní vizualizaci této celé sekce a tou je samotný kód pro nalezení odpovídajících hyperbolických komponent po zvolení určité periody. Jak již bylo zmíněno dříve, naprogramovat tento kód vycházející přímo z teorie by bylo časově velmi neefektivní a kazilo by to výsledný uživatelský dojem. S ohledem na tento fakt jsme se rozhodli využít více brute-force přístup. Nijak jsme neporušili korektnost vykreslení, ale využili jsme jiných teoretických vlastností tak, abychom k požadovanému výsledku dospěli v co nejrychlejším čase.

Konkrétně jsme využili vlastnosti, kdy jsme schopni spočítat pomocí již zmiňované funkce `periods` periodu jednotlivých bodů množiny. Pokud toto provedeme pro všechny body a barevně odlišíme ty, jejichž periodu hledáme, získáme stejný výsledek, jako kdybychom vycházeli přímo z teorie 3.2. Tento kód využívá modifikované funkce `periods`, o které byla řeč již v předchozích kódech. Pro úsporu výpočetního času je v této funkci ale ukládán pouze určitý počet posledních hodnot `maxCheckLength` závislý na volbě periody `targetPeriod`. Platí totiž `maxCheckLength` = 2·`targetPeriod`. Ty se poté uloží do pole `lastFew` a začne kontrola periodicity.

```
for (var j = 0; div[j] <= targetPeriod; j++) {
  // max. 4 checks
  let numChecks = Math.min(4, maxCheckLength - div[j]);
  for (var i = maxCheckLength - div[j] - 1;
        i >= maxCheckLength - div[j] - numChecks; i--) {
    if (Math.abs(lastFew[i].re - lastFew[(i + div[j])].re) < 0.0001 &&
```

```
7           Math.abs(lastFew[i].im - lastFew[(i + div[j])].im) < 0.0001) {
8           hit = true;
9           period = div[j];
10          break;
11        }
12      }
13      if (hit) break;
14    }
```

Vnější `for` cyklus prochází všechny dělitele `div` zvolené periody `targetPeriod`. Je to rychlejší verze cyklu, kde bychom smyčkou procházeli od nuly a brali v úvahu i periody, které nejsou děliteli `tergetPeriod`. Vnitřní `for` cyklus kontroluje shodné hodnoty v poli `lastFew` s přesností 10^{-4}. Procházení vnitřní smyčky jsme opět optimalizovali, aby výpočetní náročnost byla co nejmenší. Nejlepší bude si princip demonstrovat na příkladu. Mějme `targetPeriod` = 6, `maxCheckLength` = 12, `div` = [1, 2, 3, 6] a velikost pole `lastFew` bude 12. Potom postup při výběru indexů shodných hodnot v poli podle naší podmínky ve `for` cyklu vidíme v tabulce 6.1. Jakmile se najdou shodné hodnoty, cyklus se ukončí.

div[j]	i	i + div[j]
1	10	11
1	9	10
1	8	9
1	7	8
2	9	11
2	8	10
2	7	9
2	6	8

div[j]	i	i + div[j]
3	8	11
3	7	10
3	6	9
3	5	8
6	5	11
6	4	10
6	3	9
6	2	8

Tabulka 6.1: Výběr indexů hodnot v `lastFew` podle podmínky ve `for` cyklu.

Následně se pomocí vykreslovacích funkcí vygeneruje Mandelbrotova množina v odstínech šedi s červeně zvýrazněnými odpovídajícími hyperbolickými komponentami. Jak už bylo vícekrát zmíněno, hlavně u komponent vyšších period, může být velmi obtížné je všechny najít kvůli jejich velikosti. Proto je v kódu funkce `numberOfHyperbolicComponents`, která je naprogramována pomocí rekurze na základě 3.2.3 a spočítá celkový počet komponent dané periody `period`.

```
1  function numberOfHyperbolicComponents(period) {
2    let all_num  = Math.pow(2, period - 1);
3      for (let i = 1; i < period; i++) {
4        if (period % i === 0) {
5          all_num -= numberOfHyperbolicComponents(i);
6        }
7      }
8      return all_num;
9  }
```

6.2 Buddhabrot

Jedná se o nestandardní možnost vizualizace Mandelbrotovy množiny, která se zaměřuje na orbity bodů divergujících do nekonečna v průběhu iteračního procesu. Dalo by se tedy říct, že stojí na úplně opačném principu než klasická Mandelbrotova množina, kterou zajímají body, jejichž orbita nediverguje. Lépe řečeno, jde o rozdělení pravděpodobnosti počátečních bodů orbit, které divergují na základě iteračního předpisu $z \mapsto z^2 + c$.

Samotný algoritmus pro vizualizaci pak funguje následovně. Vytvoříme dvourozměrné pole bodů `largeArray`, kde každý odpovídá pixelu výsledného fraktálu. Budou se do něj ukládat průchody jednotlivých bodů.

```
let largeArray = new Array(width).fill(null)
    .map(() => new Array(height).fill(0));
```

Následně pro každý bod výřezu komplexní roviny, který je určen hodnotami `realMax`, `realMin`, `imaginaryMax` a `imaginaryMin`, se provede iterační cyklus známý z běžné vizualizace Mandelbrotovy množiny. Průchody se opakují do doby, dokud cyklus nedosáhne maximálního počtu iterací `maxIterations`, nebo nepřekročí limitní poloměr divergence. Jen navíc se všechny hodnoty během iterací ukládají do dočasných proměnných `iterHistoryRe` pro reálnou část komplexního čísla a `iterHistoryIm`.

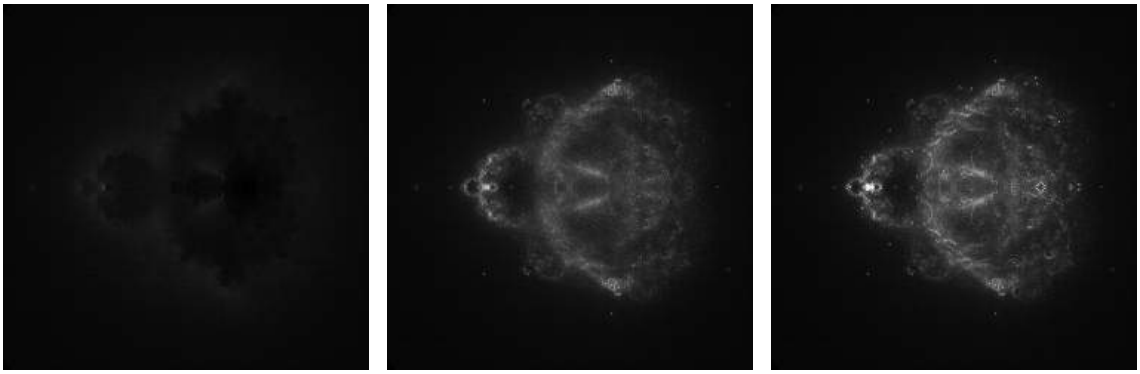
```
for (n = 0; n < maxIterations; n++) {
    let r2 = zr * zr;
    let i2 = zi * zi;
    let ri2 = zr * zi * 2;
    zr = r2 - i2 + cr;
    zi = ri2 + ci;

    iterHistoryRe.push(zr);
    iterHistoryIm.push(zi);

    if (zr * zr + zi * zi > escapeRadiusSquared) {
        escaped = true;
        break;
    }
}
```

Jestliže po provedení iteračního cyklu je daný bod vyhodnocen jako součást divergentní orbity, vezmou se všechny hodnoty z `iterHistoryRe` a `iterHistoryIm` a po přepočtu z komplexního čísla na pozici pixelu vůči vykreslenému výřezu se postupně vloží do `j` a `k`. Naopak hodnoty pro nedivergentní orbity se ignorují. Pro každou hodnotu se také zvýší čítač `largeArray` o jedna.

```
if (escaped) {
  for (let i = 0; i < n; i++) {
    let j = (iterHistoryRe[i] - realMin)*(width / (realMax - realMin));
    let k = (iterHistoryIm[i] - imaginaryMin) *
            (height / (imaginaryMax - imaginaryMin));
    if (j >= 0 && j < width && k >= 0 && k < height) {
        largeArray[Math.floor(j)][Math.floor(k)]++;
    }
```



(a) $n = 50$ **(b)** $n = 3500$ **(c)** $n = 10\,000$

Obrázek 6.1: Buddhabrot množina vykreslené pro počet iterací n.

```
9   }
```

Tento proces se provede pro všechny body a na závěr se vyhodnotí počet průchodů jednotlivými pixely, které mají každý unikátní pozici v `largeArray`. Platí, že čím větší počet průchodů, tím jasnější barva. Na obrázku 6.1 můžeme vidět Buddhabrot množiny pro odlišný maximální počet iterací. Všimněme si, jak se s rostoucím počtem iterací zvyšuje jas často navštívených bodů.

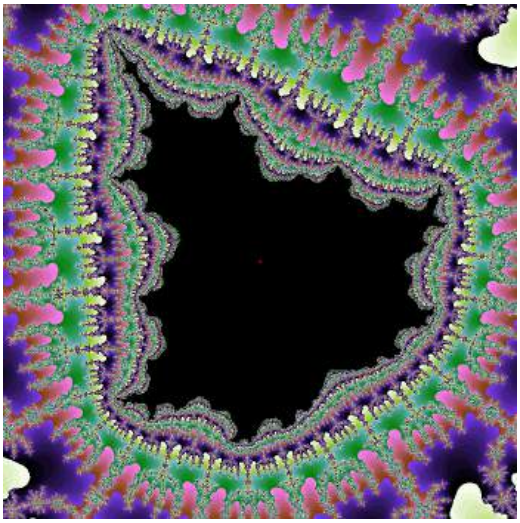
6.3 Mandebrot set Explorer

Klíčovým aspektem této části aplikace je rychlost. Nejedná se o nikterak složitou vizualizaci, nicméně pokud by byla pomalá, naprosto by to zkazilo výsledný dojem. Hlavní snaha je poskytnout uživateli co nejvíc interaktivní a zároveň funkčně efektivní program, jehož důležité pasáže si postupně nyní rozebereme. Vyskytuje se zde vícero pomocných funkcí:

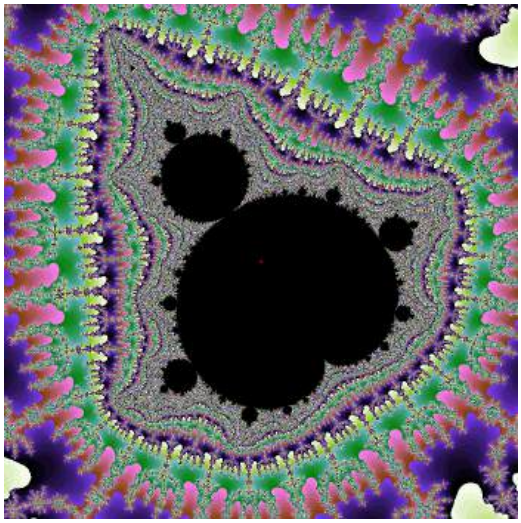
- `updateZoomInfo` - výpočet a výpis velikosti přiblížení.
- `updateInputs` - aktualizace hodnot středu v polích.
- `measureRendering` - výpočet a výpis rychlosti vygenerovaného obrazu.
- `detectPeriod` - výpočet a výpis periody bodu určeného středem.

Jejich konkrétní kód si však uvádět nebudeme, pro pochopení to není nezbytně nutné.

Původní verze kódu pracovala s fixním počtem maximálních iterací `maxIter`. Zde však nastal problém. Jak můžeme vidět z obrázku 6.2, při větším přiblížení se nevykreslí dostatek detailu, zejména ve složitějších fraktálních strukturách. Naopak jsme při ladění přišli na to, že při základním vyobrazení je potřeba výrazně menší množství iterací, abychom vykreslili Mandelbrotovu množinu v dostatečném detailu.



(a) Pevný maximální počet iterací. `maxIter` = 360.



(b) Dynamická změna maximálního počtu iterací. `maxIter` = 2260.

Obrázek 6.2: Detail Mandelbrotovy množiny s přiblížením 32 768x, se středem v bodě $-0{,}5597265625 - 0{,}639140625i$.

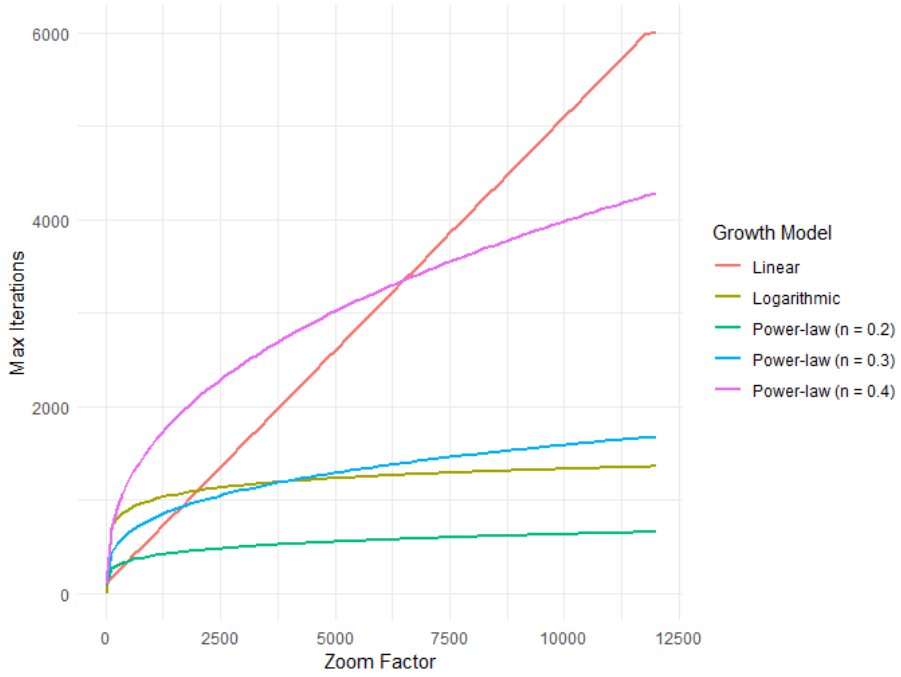
Vznikla proto funkce `getMaxIterations`, která dokáže dynamicky měnit maximální počet iterací `maxIter` v závislosti na hloubce přiblížení až do maximální hodnoty 6000 iterací. Dalším bodem bylo vymyšlení způsobu přičítání iterací. Jako první a zároveň nejjednodušším způsobem se jevil lineární nárůst iterací.

```
1 const maxIterations = Math.min(6000, Math.floor(baseIterations +
    zoomFactor));
```

Bohužel tento přístup není dostatečný. Buď jsme se dostali na velký počet iterací příliš brzy, nebo naopak při snaze přičítat jen určitou část `zoomFactor` (např. `zoomFactor`/2) nebyl počet iterací dostatečně vysoký u počátečního vykreslení. Další variantou proto bylo využití logaritmu. Myšlenka byla taková, že při využití logaritmické funkce zaručíme rychlejší nárůst iterací při nižším přiblížení, naopak u vyššího přiblížení bude nárůst pozvolnější.

```
1 const maxIterations = Math.min(6000, Math.floor(baseIterations * Math.
    log2(zoomFactor)));
```

I navzdory lepšímu výsledku, než při předchozím přístupu, má i tento své nedostatky. Počet iterací se totiž pro vyšší zoom přičítá až příliš pomalu. Výpočetní rychlost je tedy poměrně vysoká, na druhou stranu se brzy začnou projevovat nedostatky v detailním vykreslení. Následující variantou proto bylo vzít mocninnou funkci, ale s exponentem menším než jedna. Její nárůst je totiž pomalejší než u lineární funkce, ale rychlejší jak u logaritmu. Teď už šlo jen o to, správně zvolit exponent n. Po odladění jsme nakonec vybrali exponent $0{,}3$, který nejlépe vyhovuje našim požadavkům. Tedy potom kompletní funkce `getMaxIterations` vypadá následovně:



Obrázek 6.3: Graf závislosti maximálního počtu iterací na velikosti přiblížení.

```
function getMaxIterations() {
    const baseIterations = 100;
    const zoomFactor = 4 / scale;
    const maxIterations = Math.min(6000,
        Math.floor(baseIterations * Math.pow(zoomFactor, 0.3)));
    \\...
    return maxIterations;
}
```

Zaručíme tak menší výpočetní náročnost pro vykreslení s menším přiblížením. Na druhou stranu budeme mít dostatek iterací pro velké přiblížení a nebude tak problém vykreslit i opravdu drobné detaily.

Všechny vyzkoušené přístupy přírůstku počtu iterací můžeme porovnat na obrázku 6.3. Vidíme v něm také další dvě varianty pro exponent mocninné funkce. Pro $n = 0{,}2$ je nárůst velmi malý, dokonce menší jak u logaritmického růstu. Naopak pro $n = 0{,}4$ zase velmi velký, čímž by opět nastal podobný problém jako u lineárního růstu.

Nyní rozeberme samotný princip přiblížení. Jako pomoc uživateli zde slouží červený bod uprostřed obrázku. Lze tak jednoduše vidět, jakou část Mandelbrotovy množiny se chystá přiblížit. Může si proto například volbou souřadnic středu už před prvním přiblížením vybrat přesnou pozici, kterou chce prozkoumat. Následně už se stačí jen dostatečně přiblížit, aniž by musel hlídat, kam přesně se mu obraz přibližuje. Samotné přiblížení/oddálení se provádí pomocí tlačítek. Po jejich stisknutí se jednoduše změní hodnota měřítka `scale` a vygeneruje se odpovídající obrázek. Samostatné tlačítko pro resetování pak nastaví měřítku jeho původní hodnotu a na jejím základě se vykreslí původní obrázek. Proměnné `offsetY` a

`offsetX` představují posunutí počátku komplexní roviny vůči středu obrázku.

```
1 document.getElementById("zoomIn").onclick = () => { scale /= 2;
    renderCanvas(); };
2 document.getElementById("zoomOut").onclick = () => { scale *= 2;
    renderCanvas(); };
3 document.getElementById("resetView").onclick = () => { offsetX = -0.5;
    offsetY = 0; scale = 4; renderCanvas(); };
```

Pomocí funkce `renderCanvas` se spustí všechny ostatní funkce spojené s vykreslením. Její strukturu rozebereme později.

Co se týče posunu po různých částech obrázku, nebylo to tak snadné, jak s přibližováním. Opět připomeňme, že klíčovými faktory v této části aplikace jsou rychlost a plynulost. Posun funguje na principu „drag-and-pan“ (chycení a tažení). A i když vykreslení samotného obrázku není nijak zásadně pomalé, už i například 150 milisekund zapříčiní nedostatky v plynulosti při pohybu.

Tento problém řeší tzv. virtuální plátno. Vygenerujeme totiž obrázek s šířkou 1 600 pixelů, ale viditelných bude pouze 800 pixelů.

```
1 const overscan = 0.5;
2 const virtualWidth = canvas.width * (1 + 2 * overscan);
3 const virtualHeight = canvas.height * (1 + 2 * overscan);
```

Proměnná `canvas.width`, resp. `canvas.height` značí šířku, resp. výšku viditelného pole. Proměnná `overspan` je nastavena tak, aby šířka neviditelného pole byla dvojnásobná. Tedy ve výsledku je na každé straně ještě čtvrtina šířky pixelů schována.

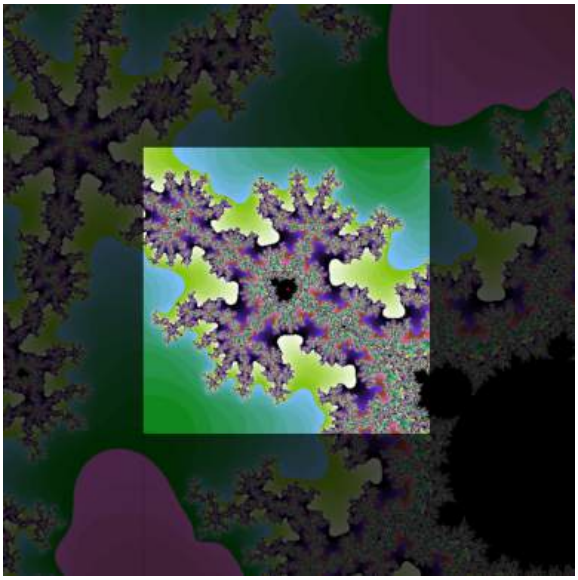
Pokud tedy uživatel bude chtít posunout s obrázkem, posune se do již vygenerované části, a tak není potřeba žádných nových výpočtů, které by zpomalily proces posunu. Vše tak bude probíhat naprosto plynule. Pro ilustraci této myšlenky viz obrázek 6.4. Navíc po každém dokončení posunu, jinými slovy po každém uvolnění tlačítka myši, se virtuální plátno vygeneruje znovu. Takže bude vždy na každé straně viditelného plátna nachystaná část pro posun.

Při dokončení jakékoliv operace (zoom, posun, změna souřadnic středu,...) se provede již zmíněná funkce `renderCanvas`.

```
1 function renderCanvas() {
2     measureRendering(() => {
3         draw();
4         drawVirtualCanvas();
5     });
6     updateZoomInfo();
7     updateInputs();
8     detectPeriod();
9 }
```

Můžeme v ní vidět všechny pomocné funkce společně s funkcemi `draw`, která vykreslí její viditelnou část a `drawVirtualCanvas`, která vygeneruje obraz na virtuální plátno.

V tomto momentě náš kód funguje už prakticky přesně tak, jak požadujeme. Při ladění se však vyskytl ještě jeden menší problém. Přibližování není konstantně rychlé pro všechny části Mandelbrotovy množiny. Vykreslit část, kde je spousta



Obrázek 6.4: Virtuální plátno (celý obrázek 1600x1600 px) vs. viditelné plátno (neztmavená část uprostřed 800x800 px)

bodů uvnitř množiny, je mnohem časově náročnější než vykreslení části hranice množiny. Způsobuje to fakt, že přiřazení barvy bodu, který je uvnitř množiny, zabere velké množství iterací. Naopak, pro bod na hranici, nebo vně množiny platí, že po pár iteracích „uteče“ do nekonečna a iterační cyklus tak skončí po nesplnění základní podmínky limitního poloměru divergence[1]. Tento problém nastával zejména v blízkosti hlavního kardioidu nebo komponenty s periodou dva. Přidali jsme proto ještě funkci `isInMandelbrot`.

```
1 function isInMandelbrot(x, y) {
2     const q = (x - 0.25) ** 2 + y ** 2;
3     return q * (q + (x - 0.25)) <= 0.25 * y ** 2
4            || (x + 1) ** 2 + y ** 2 <= 0.0625;
5 }
```

Touto funkcí projde každý bod před tím, než začne iterační smyčka. Není to nic jiného, než dosazení do analytického vyjádření kardioidu nebo komponenty s periodou dva. Navíc jsme předpis pro kontrolu bodů v kardioidu upravili pro vyšší efektivitu výpočtu. Tato úprava je ekvivalentní analytickému zápisu, i když je napsána jinak pro numerickou optimalizaci.

Funkce vrátí logickou hodnotu *true*/*false*. Pro *true* se bod rovnou obarví černou barvou, jakožto vnitřek množiny. Pro *false* přejdeme ke klasickému přístupu a začneme iterovat. Odpadne nám tak velká část výpočtů, zejména v případech, kde vykreslená část obsahuje velké množství bodů z „vnitřku“ množiny.

[1] V případě zájmu viz [23].

6.4 Coloring Methods

Obarvovací metody jsou část aplikace, která je rozdělena ještě navíc zvlášť pro Mandelbrotovu množinu a Juliovy množiny. Jelikož ale většina metod má stejný princip při generování jak Mandelbrotovy, tak Juliovy množiny, zamezíme zbytečným duplicitám a nebudeme uvádět stejnou metodu pro každou množinu jednotlivě. V případě, že by se v metodě nacházel významný rozdíl, bude na to v textu upozorněno.

6.4.1 Binary Decomposition

Metoda binární dekompozice je specifický způsob vizualizace, kdy cílem metody je na základě binární podmínky rozhodnout, jakou barvu přiřadíme jednotlivým bodům komplexní roviny. V případě binární dekompozice jsou výsledkem pouze dvě barvy, v našem případě černá a bílá.

V každém bodě komplexní roviny se iteruje, dokud není splněna úniková podmínka, nebo dokud nedosáhneme maximálního počtu iterací $n_{\max}$. Úniková podmínka je definována jako překročení limitního poloměru divergence r, což odpovídá tomu, že bod „unikl“ do nekonečna. Pro lepší výpočetní rychlost se však využívá následující vztah:

$$|z_k|^2 = \mathrm{Re}(z_k)^2 + \mathrm{Im}(z_k)^2 > r^2,$$

kde $\mathrm{Re}(z_k)$ a $\mathrm{Im}(z_k)$ jsou reálná a imaginární část čísla z po k-té iteraci. Hodnotu $|z_k|^2$ reprezentuje v kódu funkce `cnorm`.

```
1 function cnorm(z) {
2     return z.real * z.real + z.imag * z.imag;
3 }
```

Jak už bylo zmíněno, metoda binární dekompozice přiřazuje pixelům pouze dvě barvy na základě jednoduché podmínky, která je formulována následovně:

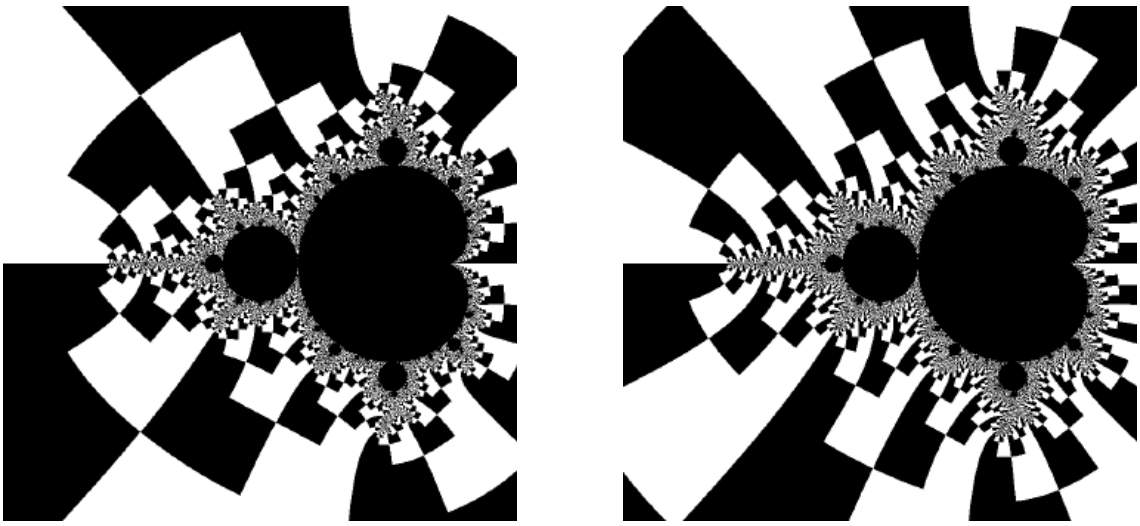
$$pixel_{barva} = \begin{cases} 1, & \text{pokud } k < n_{\max} \text{ a } \mathrm{Im}(z_k) < 0, \\ 0, & \text{jinak,} \end{cases}$$

Kde k je počet iterací, které byly provedeny před překročením limitního poloměru divergence nebo dosažením $n_{\max}$.

Jinými slovy nám tato podmínka říká, že pokud bod unikl před dosažením maximálního počtu iterací a zároveň imaginární část z je záporná, pixelu se přiřadí barva $pixel_{barva} = 1$ (bílá). Ve všech ostatních případech pixelu přiřadíme barvu $pixel_{barva} = 0$ (černá).

```
1 let color = (k < nMax && z.imag < 0) ? 255 : 0;
```

Pro ilustraci samotného kódu využijeme verzi pro Mandelbrotovu množinu s tím, že pro Juliovy množiny je kód prakticky stejný, jen je potřeba ještě zvolit konstantu c.



(a) `er2` = 100 **(b)** `er2` = 2000

Obrázek 6.5: Mandelbrotova množina metodou Binary Decomposition pro různé hodnoty limitního poloměru divergence.

```
function mandelbrotBinaryDecompostion() {
  for (let j = 0; j < canvas.height; j++) {
       let y = (canvas.height/2 - (j + 0.5))/(canvas.height/2)*r;
      for (let i = 0; i < canvas.width; i++) {
          let x = (i + 0.5 - canvas.width/2)/(canvas.height/2)*r - 1;
          let c = { real: x, imag: y };
          let z = { real: 0, imag: 0 };
          let k;

          for (k = 0; k < nMax; k++) {
              let zReal2 = z.real * z.real - z.imag * z.imag + c.real;
              let zImag2 = 2 * z.real * z.imag + c.imag;
              z.real = zReal2;
              z.imag = zImag2;

              if (cnorm(z) > er2) break;
          }
          //coloring...
      }
  }
}
```

Na obrázku 6.5 můžeme pozorovat výsledný obrázek. Je zde také hezky vidět rozdíl při změně hodnoty limitního poloměru divergence `er2`.

6.4.2 Color Decomposition

Pokročilejší verzí předchozí metody je barevná dekompozice. Jedná se o metodu vizualizace, která kombinuje ETA algoritmus (viz [23]) se sofistikovanější barevnou interpretací výsledků. Tato metoda přiřazuje barvu jednotlivým pixelům podle:

- **Počtu iterací** (n), po kterých bod překročí limitní poloměr divergence (r). Tento parametr slouží jako základní složka pro interpolaci barev.
- **Argumentu** (θ) poslední hodnoty z_n, který určuje úhel mezi touto hodnotou a kladnou částí reálné osy v komplexní rovině. Tento úhel je vyjádřen jako část celého kruhu (2π) a přidává hloubku vizualizace.

Kombinace těchto dvou veličin umožňuje vytvořit hladký gradient, který oživí výsledný vzhled fraktálu.

Stejně jako u binární dekompozice se pro každý bod v komplexní rovině provede iterační proces. V každém průchodu se kontroluje, zda-li hodnota bodu nepřekročila limitní poměr divergence r. Zároveň se využije Escape-time algoritmus, který zaznamenává počet iterací k, při kterých dojde k překročení r.

Výsledná barva pixelu je interpolována na základě tří proměnných definujících HSV (Hue, Saturation, Value) model barev:

- **Hue (odstín):** Odstín je dán normalizovanou hodnotou z escape-time algoritmu ($k/n_{\max}$). To znamená, že čím více iterací bod potřeboval k opuštění, tím se barva posouvá ve spektru.
- **Saturation (sytost):** V našem případě konstantní proměnná s hodnotou $0{,}8$.
- **Value (jas):** Jas je odvozen z hodnoty argumentu θ posledního bodu z_n před překročením limitního poloměru divergence, normalizovaného na $[0, 1]$.

Využili jsme vztah v definici z [13] pro argument θ, který je definován jako:

$$\theta = \arctan\left(\frac{\mathrm{Im}(z_k)}{\mathrm{Re}(z_k)}\right),$$

případně $\theta + \pi$ v případě, že $\theta < 0$. Normalizaci pak provedeme pomocí následujícího vztahu

$$\arg\,\mathrm{norm}(z_k) = \frac{\theta}{\pi}.$$

Funkce obstarávající v kódu výpočet argumentu θ se nazývá `turn` a vypadá následovně:

```
1 function turn(c) {
2     let arg = Math.atan2(c.imag, c.real);
3     if (arg < 0) arg += Math.PI;
4     return arg / Math.PI;
5 }
```

Pro naše účely a jednodušší manipulaci s barevnými složkami u jednotlivých pixelů při vykreslování je barevný model HSV (Hue, Saturation, Value) nakonec převeden do formátu RGB[2].

[2] V případě zájmu viz [31].

```
function hsv2rgb(h, s, v) {
    let r, g, b;
    if (s === 0) {
        r = g = b = v;
    } else {
        h = 6 * (h - Math.floor(h));
        const i = Math.floor(h);
        const f = h - i;
        const p = v * (1 - s);
        const q = v * (1 - s * f);
        const t = v * (1 - s * (1 - f));
        switch (i) {
            case 0: r = v; g = t; b = p; break;
            case 1: r = q; g = v; b = p; break;
            case 2: r = p; g = v; b = t; break;
            case 3: r = p; g = q; b = v; break;
            case 4: r = t; g = p; b = v; break;
            default: r = v; g = p; b = q; break;
        }
    }
    return {
        r: Math.round(255 * r),
        g: Math.round(255 * g),
        b: Math.round(255 * b),
    };
}
```

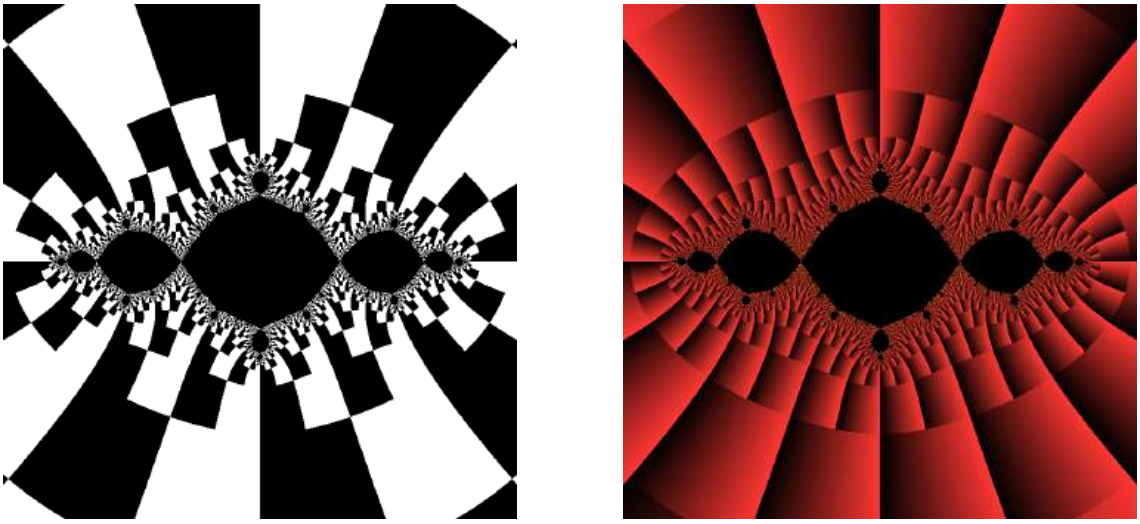
Jako ukázku kódu opět využijeme variantu pro Mandelbrotovu množinu s vědomím, že varianta pro Juliovu množinu je totožná, jen se opět musí navíc ještě zvolit konstanta c. Není nutné uvádět celou funkci, jelikož je založena na podobném principu jako binární dekompozice. Následující část proto ukáže, jakým způsobem je ošetřeno přiřazení jednotlivých barevných složek pixelu:

```
function mandelbrotColorDecomposition(w, h, nMax, r, er2) {
    //iterative process...
    let hue = 0, sat = 0, val = 0; // default color (black)

    if (k < nMax) {
        const et = k / nMax;
        const normalizedArgument = turn(z);
        hue = et;
        sat = 0.8;
        val = normalizedArgument;
        }

    const { r: red, g: grn, b: blu } = hsv2rgb(hue, sat, val);
    const index = 3 * (j * w + i);
    img[index] = red;
    img[index + 1] = grn;
    img[index + 2] = blu;
}
```

Pro porovnání výsledků první a druhé metody slouží obrázek 6.6. Pro úplnost je tentokrát vizualizována Juliova množina.



(a) Binary Decomposition **(b)** Color Decomposition

Obrázek 6.6: Porovnání metody Binary a Color Decomposition pro Juliovu množinu kde $c = -1$.

6.4.3 Field Lines

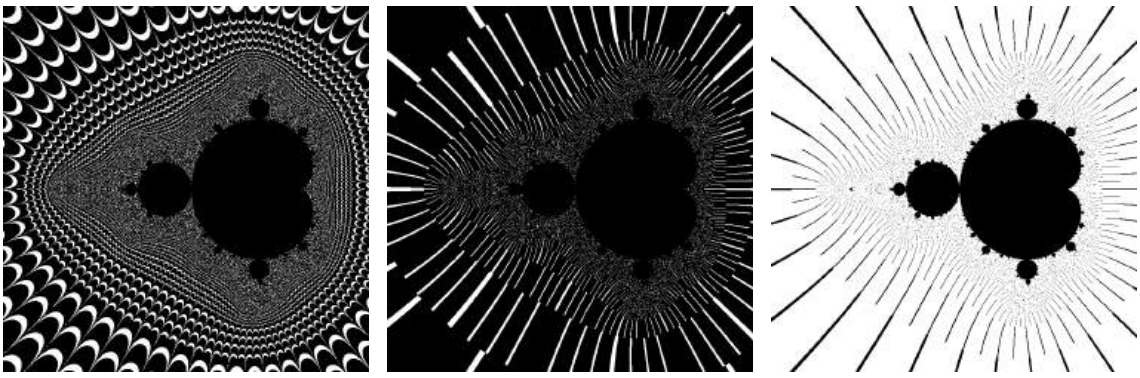
Další metodou je Field Lines, nebo také volně přeloženo do češtiny pole čar. Základní myšlenkou je rozdělit prostor na třídy na základě zvolené podmínky a vykreslit je různými způsoby. Slovo čára je zde použito, jelikož výsledná vizualizace připomíná čáry vedoucí z Mandelbrotovy množiny. Jelikož nejsme nijak omezeni tím, jak by podmínka pro zvýraznění bodu měla vypadat, naskýtá se zde velké množství variant. V této práci si však uvedeme pouze několik příkladů, přičemž v aplikaci je implementována jedna z nejznámějších.

- **Rychlost růstu trajektorie** - Spočítá se logaritmická rychlost růstu hodnoty bodu z_k. Čím vyšší hodnota, tím rychlejší růst. Funkce $\log(|z.real \cdot z.imag|)$ vyjadřuje rychlost změny hodnot komplexního čísla z. Následně vybereme body s podobnou rychlostí růstu. V našem případě jsme vybrali takové hodnoty z, které odpovídají rychlosti změny 4 s tolerancí 1. Volba hodnoty rychlosti i tolerance je na preferenci každého.

```
1 img[j * w + i] = (k < nMax && Math.log(Math.abs(z.real * z.imag))
      % 4 < 1) ? 255 : 0;
```

- **Úhlová orientace v komplexní rovině** - Nejprve se spočítá úhel mezi reálnou osou a vektorem, který spojuje počátek s hodnotou bodu z_k v komplexní rovině. Následně se kontroluje, zda je úhel bodu z_k blízko k nějakému násobku $\frac{\pi}{4}$ (45 stupňů). To znamená, že bod se zvýrazní, pokud jeho úhel je v rámci určité tolerance (menší než $0{,}1$ radiánu).

```
1 img[j * w + i] = (k < nMax && Math.abs(Math.atan2(z.imag, z.real))
      % (Math.PI / 6) < 0.1) ? 255 : 0;
```



(a) Rychlost růstu trajektorie **(b)** Úhlová orientace v komplexní rovině **(c)** Poměr reálné a imaginární složky

Obrázek 6.7: Mandelbrotova množina metodou Field Lines pro různé podmínky obarvení.

- **Poměr reálné a imaginární složky** - V tomto případě se porovnává rychlost změny velikosti hodnot reálné a imaginární složky z_k. Konkrétně v našem případě jsou zvýrazněny takové body, které mají značně vetší imaginární složky, než reálnou. Jinými slovy se dá říct, že se jedná o body, u kterých je růst jejich imaginární složky dominantní ve srovnání s reálnou složkou. Konstanta $0{,}1$ vyjadřuje, jak velký musí být rozdíl.

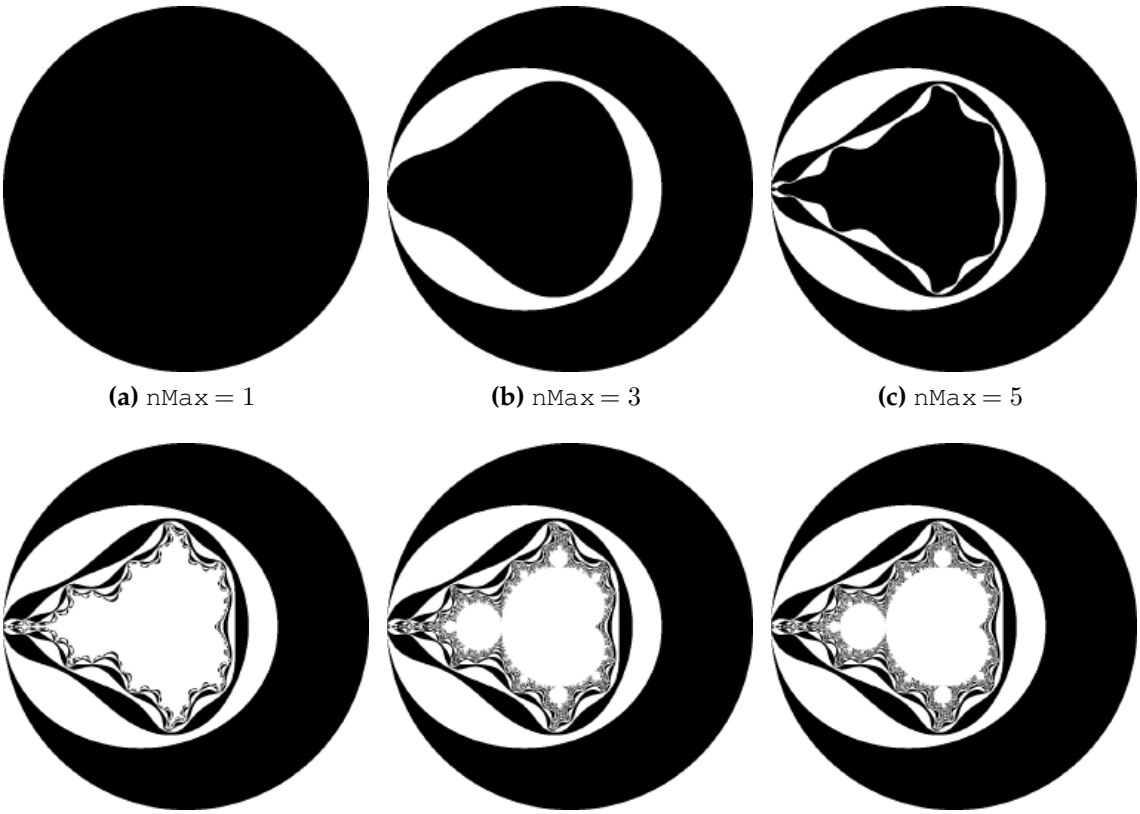
```
1 img[j * w + i] = (k < nMax && Math.abs(z.real) * 0.1 < Math.abs(z.
      imag)) ? 255 : 0;
```

Jak už bylo zmíněno, v aplikaci je použita pouze jedna varianta a to je *Poměr reálné a imaginární složky*. Je asi nejznámější a vizualizace také nejlépe odpovídá názvu. Nicméně pro představu, jak vypadají zbylé dvě, viz 6.7.

6.4.4 Level Sets

Uvedeme si ještě jednu metodu vycházející z podobného principu jako předchozí tři metody. Jedná se o metodu Level Sets, která rozlišuje oblasti komplexní roviny na základě počtu iterací, které jsou potřeba k rozhodnutí, zda bod patří do množiny nebo ne. Jelikož se jedná o určitou formu binární dekompozice komplexní roviny, stáčí zkoumat, zda-li se jedná o sudý, či lichý počet iterací. Každá iterace vytváří přirozené „úrovně“ (levely), kterými se při čím dál větším maximálním počtu iterací dostaneme k přesnější aproximaci výsledné množiny. Víc na toto téma naleznete v [23]. Přístup v této metodě je však poněkud rozdílný, i když vede ke stejnému výsledku.

K počtu iterací k přistupujeme jako k číslu v binární soustavě. To znamená, že například číslo 3 má v binární (dvojkové) soustavě zápis 11, číslo 4 pak 11 a například číslo 5 má zápis 101. Jak se čísla převádí, není tématem této práce, proto pro více informací viz [33]. Co je ale důležité, že díky binárnímu zápisu jsme schopni jednoduše zjistit, zda-li se jedná o sudé, či liché číslo. Pro sudé číslo platí,



(a) nMax = 1 **(b)** nMax = 3 **(c)** nMax = 5

(d) nMax = 10 **(e)** nMax = 50 **(f)** nMax = 150

Obrázek 6.8: Mandelbrotova množina metodou Field Lines pro různé podmínky obarvení.

že poslední číslice je vždy 0, naopak pro liché 1 a tohoto faktu využijeme v naší metodě.

Pomocí bitového operátoru & tuto vlastnost dokážeme jednoduše rozlišit a díky tomu pak obarvit jednotlivé body komplexní roviny.

```
1 img[j * w + i] = (k & 1) ? 0 : 255; // Binary color (0 or 255)
```

Tento řádek nám říká, že jestliže se na pozici posledního bitu nachází číslo 1, pak nám operace vrátí hodnotu *true* a obarvíme bod černě (hodnota 0), jelikož se jedná o liché číslo. Naopak pro sudá čísla, které končí nulou, (0&1) vrátí hodnotu *false* a přiřadí se druhá hodnota, tedy 255.

Tímto způsobem se podmínka provede pro všechny body výřezu komplexní roviny a vykreslí se obrázek. Pro různou volbu maximálního počtu iterací nMax můžeme vidět výsledek metody Level Set na obrázku 6.8. Všimněme si, že od hodnoty nMax větší jak 50 nejsou patrné příliš velké rozdíly, co se týče přesnosti aproximace.

6.4.5 Gradient Mapping

Metoda Gradient Mapping je způsob vizualizace, kde jak už název napovídá, bude hrát hlavní roli gradient. Konkrétně v našem případě se bude jednat o gradient ve smyslu velikosti změny počtu iterací dostačujících pro „únik“ bodu do nekonečna. Pro výpočet gradientu se používají hodnoty počtu iterací z okolních pixelů, což umožňuje zjistit, jak rychle se hodnota mění mezi sousedními pixely.

Gradient mezi dvěma sousedními pixely k_0 a k_1 spočítáme jako rozdíl mezi hodnotami počtu iterací:

$$\Delta\mu = \mu(k_1) - \mu(k_0).$$

Tento rozdíl určuje rychlost změny počtu iterací mezi sousedními pixely a používá se pro vizualizaci jemných detailů na okrajích množiny.

Pokud vezmeme rozdíly mezi sousedními pixely v horizontálním a vertikálním směru, gradient lze zapsat jako vektor:

$$\mathbf{v} = (\mu_x - \mu_0, \mu_y - \mu_0, s),$$

kde μ_x a μ_y jsou hodnoty počtu iterací sousedních pixelů, μ_0 je hodnota počtu iterací aktuálního pixelu a s je parametr určující „sílu“ gradientu, tedy jak silně gradient ovlivní výslednou barvu.

```
k0 = j * (w + 1) + i; //our position
kx = j * (w + 1) + (i + 1); //pixel on the right
ky = (j + 1) * (w + 1) + i; // pixel on the below
v1 = mu[kx] - mu[k0];
v2 = mu[ky] - mu[k0];
v3 = s*4; //gradient impact
```

V kódu jsou pozice pixelu převedeny do jednorozměrného pole pomocí vztahů výše. Například `kx`, které vyjadřuje pozici pixelu napravo od aktuálního pixelu je přepočteno následovně:

- $\mathbf{j} * (\mathbf{w} + \mathbf{1})$ - Posun na řádek j v mřížce, kde každý řádek obsahuje $w + 1$ hodnot (šířka a navíc jedna hodnota na okraji).
- $\mathbf{i} + \mathbf{1}$ - Posun na sloupec $i + 1$, což odpovídá pixelu vpravo od aktuálního.

Tímto způsobem se vypočítají všechny pixely mřížky:

```
for (let j = 0; j < h + 1; j++) {
    for (let i = 0; i < w + 1; i++) {
    //...
    }
}
```

Vektor v se následně normalizuje, z čehož získáme vektor g:

$$\mathbf{g} = \frac{\mathbf{v}}{|\mathbf{v}|}. \tag{6.1}$$

Implementace předchozího vztahu (6.1) do kódu vypadá následovně:

```
vNorm = Math.sqrt(v1 * v1 + v2 * v2 + v3 * v3); // |z|
g1 = v1 / vNorm;
g2 = v2 / vNorm;
g3 = v3 / vNorm;
```

Hodnota odstínu (hue) je poté určena úhlem gradientu v komplexní rovině. Tento úhel se vypočítá pomocí funkce `atan2`, která vrátí arkus tangens souřadnic g_1 a g_2 zadaných jako argumenty:

$$\text{hue} = \frac{\arctan(g_1, g_2)}{2\pi}. \tag{6.2}$$

Tento výsledný úhel je následně použit pro určení barvy v modelu HSV, kde v tomto případě je hue dáno předchozím vztahem, saturation `sat` je 0 pro body uvnitř Mandelbrotovy množiny a vyšší pro ostatní body (hodnota $0{,}5$) a value hodnota `val` je odvozena od normalizované hodnoty s popsané výše.

```
hue = Math.atan2(g1, g2) / TwoPi;
sat = mu[k0] > 0 ? 0.5 : 0;
val = g3;
const { r: red, g: grn, b: blu } = hsv2rgb(hue, sat, val);
```

Pro následnou vizualizaci je opět použita funkce `hsv2rgb` pro jednodušší manipulaci.

Pro jemnější zobrazení bodů, které mají vysoký počet iterací, ale jsou označeny jako divergentní ještě před dovršením maximálního počtu iterací, je použita logaritmická korekce ve tvaru:

$$\log_2(\log(|z|)).$$

Tato korekce zajišťuje plynulejší vizualizaci bodů například na hranici (přechodech) mezi oblastmi se stejným počtem iterací, nebo na hranici množiny.

```
mu[j * (w + 1) + i] = k === n ? 0 :
        k + 1 - Math.log2(Math.log(magnitude(zReal, zImag)));
```

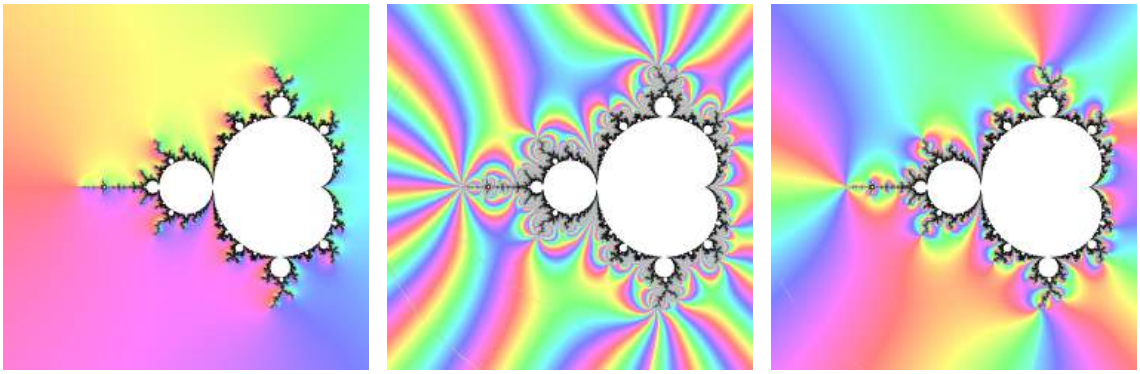
Zajímavé výsledky (viz 6.9) vycházejí při volbě různých hodnot dělitele pro `hue`, například:

```
hue = Math.atan2(g1, g2) / 2*TwoPi; // 4pi
hue = Math.atan2(g1, g2) / 8*TwoPi; // 16pi
```

Zajímavostí je, že kdybychom zvyšovali násobky π na hodnoty například 60 a výše, opět se vizualizace začne vracet ke stejnému rozložení jako pro původních 2π, jen s tím rozdílem, že se barvy budou jinak uspořádány. Nyní jsou odstíny z levého horního rohu žlutá, zelená, modrá, růžová. Při 80π je to naprosto stejné, jen žlutá, růžová, modrá a zelená.

6.4.6 Histogram Coloring

V překladu *Histogramová metoda barvení* je metoda, která rovnoměrně distribuuje barvy přes celou oblast vizualizace, aniž bychom potřebovali znát maximální počet iterací `nMax`.



(a) Původní hodnota 2π **(b)** Hodnota 4π **(c)** Hodnota 16π

Obrázek 6.9: Mandelbrotova množina metodou Gradient Mapping pro různé násobky π.

Jako první se spočítá počet iterací (funkce `mandelbrot`), které každý bod potřebuje, než překročí limitní poloměr divergence. V našem případě je to explicitně daná hodnota 4. Tento krok se provádí běžným způsobem za pomoci escape time algoritmu, kdy se výsledný počet iterací ukládá do proměnné `iterations`. Jelikož se dále s proměnnou `iterations` pracuje, je tato hodnota uložena do pole `data`.

Jakmile jsou počty iterací pro každý pixel spočítány, dalším krokem je vytvoření histogramu, který zaznamenává četnost výskytu každého počtu iterací. To znamená, že pro každý možný počet iterací je vytvořen index, který v histogramu (proměnná `histogram`) obsahuje počet pixelů, které dosáhly tohoto počtu iterací před únikem.

```
1  for (let y = 0; y < canvas.height; y++) {
2      for (let x = 0; x < canvas.width; x++) {
3      //...
4          const iterations = mandelbrot({x: cx, y: cy});
5          data[y * canvas.width + x] = iterations;
6          if (iterations < nMax) {
7              histogram[iterations]++;
8          }
9      }
10  }
```

Po sestavení histogramu je dalším krokem výpočet kumulativní distribuční funkce (`cdf[i]`), která spočítá celkový počet pixelů, které dosáhly počtu iterací v intervalu od 0 až do `i`. Tento krok slouží k tomu, aby byly barvy distribuovány v souladu s četností výskytu iterací.

```
1  let total = 0;
2  for (let i = 0; i <= nMax; i++) {
3      total += histogram[i];
4      cdf[i] = total;
5  }
```

Následně se `cdf` normalizuje, aby hodnoty byly v intervalu $[0, 1]$. Normalizace

probíhá dělením kumulativního součtu celkovým počtem pixelů `total`:

```
for (let i = 0; i <= nMax; i++) {
    cdf[i] /= total;
}
```

Poté už každému pixelu přiřadíme odpovídající barvu funkcí `colorMap`.

```
function colorMap(value) {
    const hue = 360 * value;
    return `hsl(${hue}, 90%, 50%)`;
}
```

Tato funkce převede hodnotu z normalizované kumulativní distribuční funkce na odstín `hue` v HSV modelu. Saturation a Value jsou v tomto případě neměnné a mají konstantní hodnoty.

```
for (let y = 0; y < canvas.height; y++) {
    for (let x = 0; x < canvas.width; x++) {
        const iterations = data[y * canvas.width + x];
        const colorValue = cdf[iterations];
        const color = iterations === nMax ? 'black' : colorMap(
            colorValue);
        //...
    }
}
```

Na obrázku 6.10 můžeme vidět výsledný obrázek. Všimněme si také, jak málo výsledný obrázek závisí na maximálním počtu iterací ve srovnání například s metodou *Smooth Coloring* o které bude řeč v následující sekci.

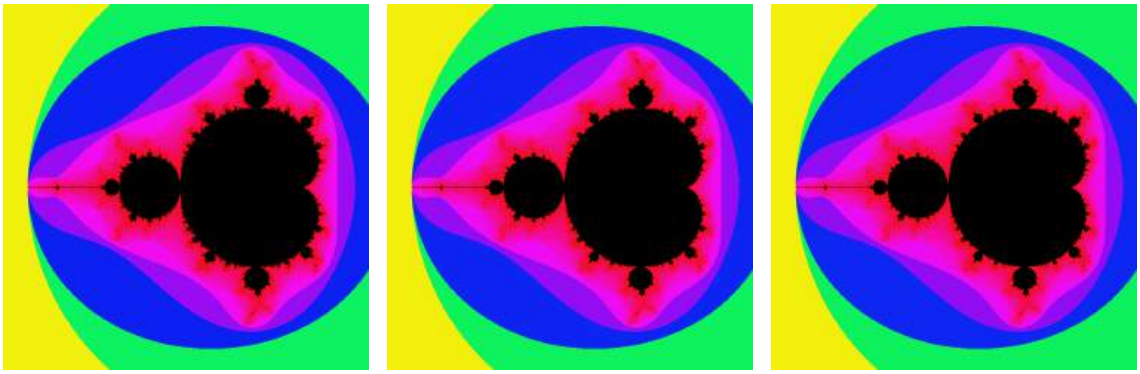
6.4.7 Smooth Coloring

Jak už víme, standardně přiřazuje barvy bodům na základě toho, jak rychle „unikají“ z množiny, tj. jak rychle jejich hodnota z_k přesáhne stanovený limitní poloměr divergence. Tento přístup je jednoduchý, rychlý a intuitivní. Na druhou stranu může vytvářet vizuální skoky mezi různými odstíny barev, jelikož přiřazujeme hodnotu, která je pro počet iterací velmi často celočíselná, pokud neprovedeme určitou transformaci. Smooth Coloring je metoda, která se pokouší tento problém eliminovat. Poskytuje jemnější přechody mezi barvami, což vede k plynulejšímu zobrazení.

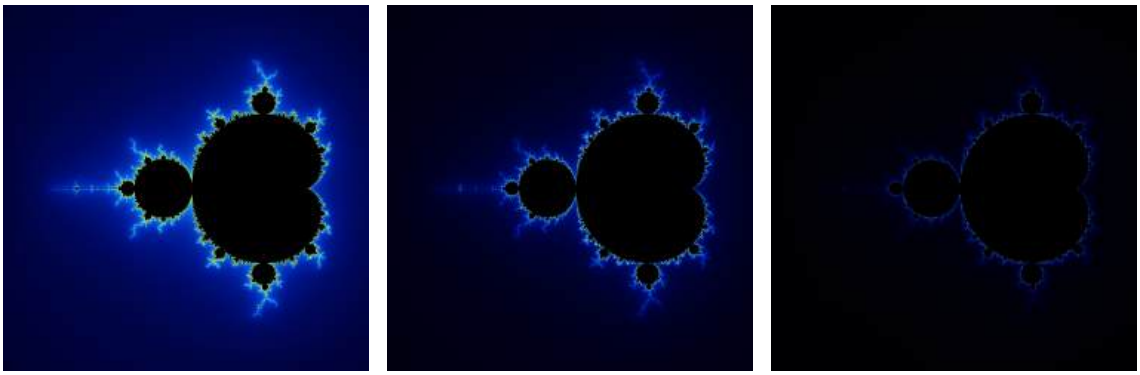
Místo používání celočíselného počtu iterací se pro každý bod počítá tzv. *normalizovaný počet iterací* $\nu(z)$. Tento přístup je založen na potenciální funkci (*potential function*) $\phi(z)$ (viz [10]), která popisuje rychlost růstu hodnoty z_n a určuje plynulost zbarvení. Pro výpočet $\nu(z)$ se používá následující vzorec:

$$\nu(z) = n - \log_P \left(\frac{\log |z_n|}{\log(r)} \right),$$

kde r je limitní poloměr divergence a P je exponent pro $z \mapsto z^P + c$. Také platí $\log(|z|) = \log\left(\sqrt{x^2 + y^2}\right) = \frac{1}{2}\log(x^2 + y^2)$ a za pomoci vlastnosti logaritmu

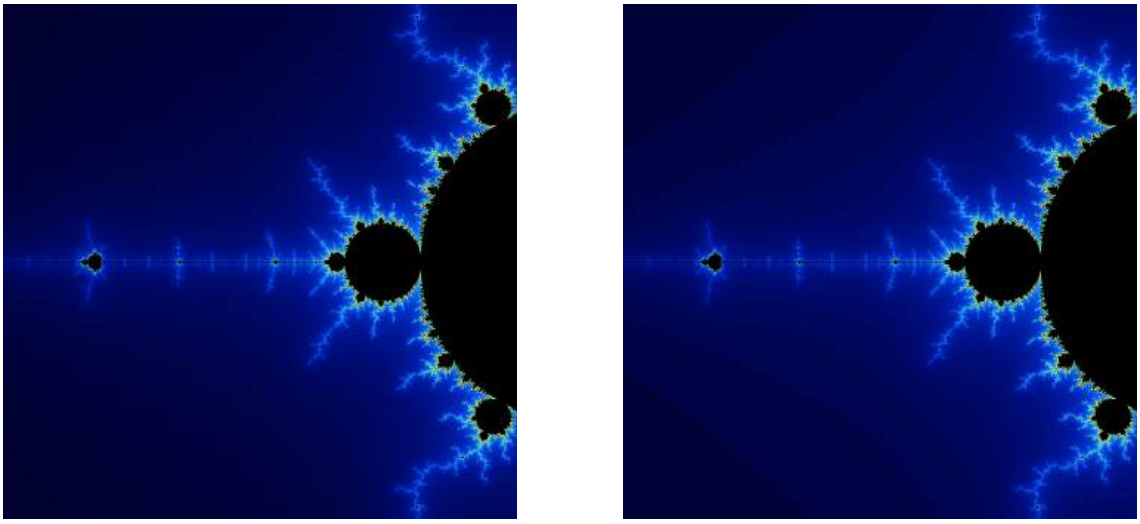


(a) nMax = 70 **(b)** nMax = 200 **(c)** nMax = 1000



(d) nMax = 70 **(e)** nMax = 200 **(f)** nMax = 1000

Obrázek 6.10: Mandelbrotova množina metodou Histogram Coloring (horní řádek) a Smooth Coloring (spodní řádek) pro různé hodnoty maximálního počtu iterací nMax.



(a) S vyhlazením. **(b)** Bez vyhlazení.

Obrázek 6.11: Část Mandelbrotovy množiny metodou Smooth Coloring.

$\log_P(x) = \frac{\log(x)}{\log(P)}$ pro $P = 2$ pak dostaneme tvar použitý v kódu, kde n je počet iterací, po kterých bod z_n překročí limitní poloměr divergence.

$$\nu(z) = n - \frac{\log\left(\frac{\log|z_n|}{\log(4)}\right)}{\log(2)}$$

```
if (n < maxIterations) {
    const logZn = Math.log(zx2 + zy2) / 2;
    const nu = n - Math.log(logZn / Math.log(4)) / Math.log(2);
    return nu; //
}
```

Tento vzorec zajišťuje, že $\nu(z)$ je reálné číslo v intervalu $[0, 1)$.

U této metody je možnost volby, zda-li chceme vykreslit množinu s vyhlazením, nebo bez. Dokážeme tak velmi snadno porovnat efekt předchozích vztahů na výsledek, který můžeme vidět na obrázku 6.11.

V kódu to potom vypadá tak, že se vyhodnotí, jestli proměnná `smooth` má hodnotu *true*, a nebo *false*. V závislosti na tom se následně volí ze dvou funkcí, buď `mandelbrot`, což je klasická funkce vracející celočíselný počet iterací, nebo `mandelbrotSmooth`, u které je implementována vyhlazovací logika.

```
const iteration = smooth ? mandelbrotSmooth(cx, cy) : mandelbrot(cx, cy
    );
const color = getColor(iteration);
// coloring ...
```

Nyní se ale přesuňme už k samostatnému obarvovacímu procesu, který je, jak jsme si mohli všimnout, obstarán funkcí `getColor` a tu si rozebereme. Máme možnost volby ze dvou palet. První je základní `default` a druhá je cyklická, pomocí sinusové funkce, s názvem `sinus`. Obě palety, pokud iterační cyklus dosáhne maximálního počtu iterací, pixelu přiřadí černou barvu:

```
if (iteration === maxIterations) {
    return 'black'; // Points inside the Mandelbrot set are black
}
```

Následně se ještě normalizuje proměnná `iteration`, čímž se zajistí rovnoměrné rozprostření barvy po celé škále iterací.

```
const t = iteration / maxIterations;
```

Teď už si každou paletu představíme zvlášť. Co se týče první zmiňované, tedy základní, tak každá složka (r, g, b) je definována jako kombinace polynomiálních výrazů mocnin t a $(1-t)$, což vytváří různé křivky pro hodnoty červené, zelené a modré složky.

```
const r = Math.floor(9 * (1 - t) * t * t * t * 255);
const g = Math.floor(15 * (1 - t) * (1 - t) * t * t * 255);
const b = Math.floor(9 * (1 - t) * (1 - t) * (1 - t) * t * 255);
```

Interpretujme si nyní tyto vztahy. Pro lepší pochopení nám s tím může pomoci i obrázek 6.12a.

- $\mathbf{r} = \lfloor \mathbf{9} \cdot (\mathbf{1} - \mathbf{t}) \cdot \mathbf{t^3} \rfloor$ - Výrazně ovlivněna třetí mocninou t, což značí, že červená složka bude dominantní hlavně pro vyšší hodnoty t.
- $\mathbf{g} = \lfloor \mathbf{9} \cdot (\mathbf{1} - \mathbf{t})^\mathbf{2} \cdot \mathbf{t^2} \rfloor$ - rovnoměrně ovlivněna jak druhou mocninou t, tak druhou mocninou $(1-t)$, což znamená dominanci pro hodnoty t okolo $0{,}5$, tedy pro hodnoty blízko středu intervalu $[0, 1]$.
- $\mathbf{r} = \lfloor \mathbf{9} \cdot (\mathbf{1} - \mathbf{t}) \cdot \mathbf{t^3} \rfloor$ - Výrazně ovlivněna třetí mocninou $(1-t)$, což značí, že modrá složka bude dominantní hlavně pro nižší hodnoty t.

Jak je ze vztahů patrné, tak pro hodnoty krajní hodnoty intervalu bude výsledná barva vždy černá. Tedy pro body vně množiny, nebo naopak velmi daleko od množiny.

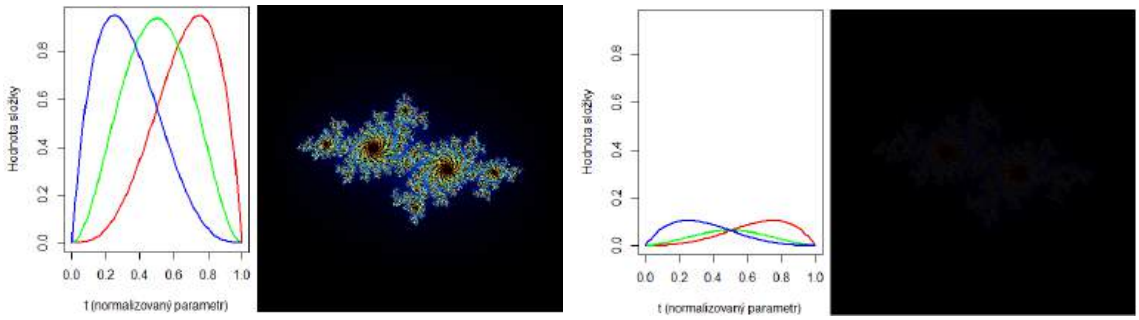
Svou roli zde hrají také konstanty na začátku každého vztahu. Z jejich pomocí se hodnoty složek ještě před roznásobením číslem 255 lépe rozdistribuují po celém intervalu $[0, 1]$. Po roznásobení tak hodnotami barevných složek pokryjeme celou barevnou škálu *RGB*. Co by se stalo, kdybychom tyto konstanty neuvažovali ukazuje obrázek 6.12b.

Maximální hodnota zde dosahuje po roznásobení velmi malých čísel, což zapříčiní extrémně tmavý výsledný obrázek.

Druhá paleta *sinus* opět využívá normalizovaný parametr t. V tomto případě se zadefinuje ještě velikost palety `paletteSize`, což je typicky hodnota 256. Následně se ještě spočítá hodnota proměnné `index`, která zde představuje konkrétní hodnotu odstínu barvy na paletě.

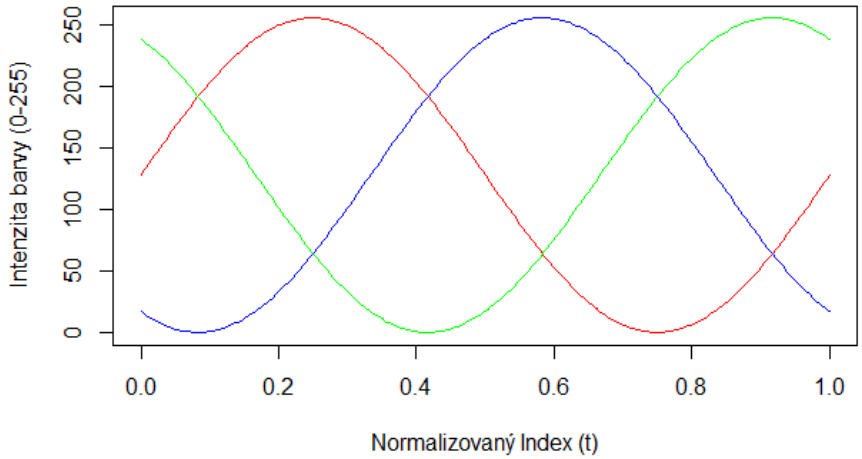
```
const paletteSize = 256;
const index = Math.floor(t * paletteSize);
```

Samotné přiřazení se pak provádí pomocí sinusové funkce, která je opět přizpůsobena k co nejlepšímu výslednému obrázku.



(a) S využitím upravujících konstant $9, 15$ a 9. **(b)** Bez využití upravujících konstant

Obrázek 6.12: Rozložení a intenzita barevných složek s odpovídající Juliovou množinou pro $c = -0.7 + 0.27015i$.



Obrázek 6.13: Rozložení barevných složek RGB u sinusové palety.

```
1 const r = Math.floor(128 + 128 * Math.sin(2 * Math.PI * index /
     paletteSize));
2 const g = Math.floor(128 + 128 * Math.sin(2 * Math.PI * index /
     paletteSize + 2 * Math.PI / 3));
3 const b = Math.floor(128 + 128 * Math.sin(2 * Math.PI * index /
     paletteSize + 4 * Math.PI / 3));
```

Jelikož funkce $\sin$ je symetrická podle reálné osy, jako první je potřeba posunout osu symetrie tak, aby všechny hodnoty funkce byly kladné, jelikož v barevném spektru nelze počítat se zápornými hodnotami. K tomu slouží právě konstanta 128 (polovina z celkové velikosti palety). Následně se zvýší amplituda na hodnotu 255, čímž pokryjeme celé barevné RGB spektrum. Barevné složky pak sinusově oscilují kolem středu 128 s navzájem posunutou fází o $\frac{2}{3}\pi$, tedy třetinu periody. Pro jednodušší představu viz obrázek 6.13.

6.4.8 Multithreading

Poslední obarvovací metoda využívá principů paralelních výpočtů, spočívajících v rozdělení úlohy na více menších částí, které jsou řešeny současně (tzv. *vlákna*). JavaScript je totiž jednovláknový jazyk, což znamená, že dokáže vykonávat pouze jednu operaci najednou (jeden řádek kódu). Pro multithreading lze ale využít funkce jako *Web Workers*, které umožňují spuštění paralelních procesů. V této metodě je hlavní myšlenkou rozdělení oblasti zobrazovaného komplexního prostoru na několik částí (*chunks*), přičemž každý worker zpracovává body pouze z přidělené oblasti.

Mezi hlavním vláknem a workery se komunikuje pomocí funkcí `postMessage` (odeslání dat workeru) a `onmessage` (zpracování odpovědi). Hlavní vlákno rozdělí oblast pro výsledný obrázek na menší úseky podle zadaného počtu workerů `numWorkers`. Každý worker obdrží specifické parametry, například rozmezí pixelů, šířku a výšku canvasu nebo maximální počet iterací. V každé vytvořené oblasti se klasickým způsobem provede obarvovací metoda. Jednotlivé části obrázku jsou pak zaslány zpět hlavnímu vláknu, které je vykreslí na plátno (canvas). Níže je na vysvětlenou schematicky znázorněn kód.

```
function draw_mandelbrot() {
  for (let i = 0; i < numWorkers; i++) {
      const worker = new Worker(URL.createObjectURL(new Blob([`
          // Event handler for messages sent to the worker
          self.onmessage = function(event) {
              const { /* data ... */ } = event.data;
              // coloring method ...
              // Send the calculated pixel data back to the main thread
              self.postMessage({ pixels });
          }
      `], { type: 'application/javascript' })));
      // Send data to the worker
      worker.postMessage({
        // data...
      });
      // Handle messages from the worker containing pixel data
      worker.onmessage = (e) => {
        //result...
      };
  }
}
```

Obarvovací je zde využita varianta Smooth Coloring bez vyhlazení, proto není potřeba rozebírat princip obarvení. Hlavním cílem u této metody bylo spíše představit, jak funguje komunikace mezi hlavním vláknem a workery.

6.5 Rendering Algorithms

V této sekci se zaměříme na principem různorodé algoritmy. Naším cílem je přiblížit, jakým způsobem jsou zmíněné Mandelbrotova množina a Juliovy množiny

generovány z matematického hlediska. Naopak o vizuální aspekty, jako jsou způsoby obarvování, se nebudeme tolik zajímat, jelikož toto téma je dostatečně rozebráno už v předešlé sekci. Na začátku ukážeme optimalizované algoritmy, jejichž základní verze byly představeny již v bakalářské práci. Dále navážeme novými algoritmy.

6.5.1 Escape Time Algorithm

Pro úplnost této práce v krátkosti připomeňme jeden ze základních algoritmů, a tím je v překladu algoritmus úniku. Na tomto způsobu jsou založeny všechny obarvovací algoritmy uvedené v této práci. Jedná se totiž o nejjednodušší, nejrychlejší a nejintuitivnější algoritmus.

Pro $z_0 = 0$ se pro všechna c postupně počítají hodnoty z_n a sleduje se pro $z_{n+1} = z_n^2 + c$, zda posloupnost těchto hodnot zůstane po celou dobu omezená, nebo unikne do nekonečna v závislosti na limitním poloměru divergence r (pokud $|z_n| > r$). Kde r se ve většině případů volí jako 2. Výsledek je vyjádřen buď jako počet iterací potřebných k úniku (tzv. rychlost úniku), nebo maximální povolený počet iterací, pokud bod neunikne. Body, které neuniknou, tvoří Mandelbrotovu množinu. Pro Juliovy množiny je pouze ten rozdíl v tom, že tento celý iterační cyklus počítáme pro všechna z_0 a fixní c. Pro více podrobností je tento algoritmus rozebrán v [23].

V této práci je kód pro Mandelbrotovu množinu optimalizován pro nižší výpočetní náročnost pomocí detekce bodů, které patří do hlavního kardioidu nebo komponenty s periodou dva. Tato optimalizace již byla vysvětlena v předcházející části zde 6.3 a proto ji není potřeba znovu popisovat. Detekci obstarává opět funkce `isInMandelbrot`. Body z těchto oblastí se vyhodnotí předem a ušetří se tak velké množství iterací. Zřejmě totiž právě pro tuto podmnožinu bodů by v normálním případě byl potřeba maximální počet iterací pro rozhodnutí. Na rozdíl od kódu z bakalářské práce je také přidána možnost volby barevného základu a volby maximálního počtu iterací.

6.5.2 Modified Inverse Iteration Method

Stejně jako předchozí algoritmus, tak i základní verze *Inverse Iteration Method* byla naprogramována již v bakalářské práci a v rámci diplomové práce byl hlavní cíl tento algoritmus optimalizovat. Shrňme vlastnosti, které jsou zásadní pro fungování algoritmu:

- **Invariance vůči iteraci:** Juliova množina $J(f)$ je kompletně invariantní vůči iteraci funkce f, neboli:

$$f(J(f)) = J(f), \text{ a zároveň } f^{-1}(J(f)) = J(f).$$

- **Uzávěr množiny všech odpuzujících periodických bodů:** Juliova množina $J(f)$ je uzávěrem množiny všech odpuzujících periodických bodů z, tedy:

$$J(f) = \overline{\{z \in \mathbb{C} \mid f^p(z) = z \text{ a } |f^{p\prime}(z)| > 1\}},$$

kde $f^p(z)$ označuje p-tou iteraci funkce f. Navíc jsou odpuzující periodické body a jejich vzory v $J_c(f)$ husté. Každý bod v Juliově množině je buď odpuzující periodický bod, nebo se nachází libovolně blízko takového bodu.

- **Repelory a atraktory:** Pokud je bod z repelorem pro funkci $f(z)$, tj. $|f'(z)| > 1$, pak je tento bod atraktorem pro inverzní funkci $f^{-1}(z)$, tj. $|(f^{-1})'(z)| < 1$, za předpokladu, že f^{-1} existuje a je dobře definována v okolí z.

Na základě těchto teoretických vlastností jsme schopní naprogramovat algoritmus, který bude generovat Juliovu množinu metodou Inverse Iteration, při které se využívá právě inverzní iterační funkce $f^{-1}(z) = \pm\sqrt{z - c}$ pro počáteční bod $z_0 = 1$.

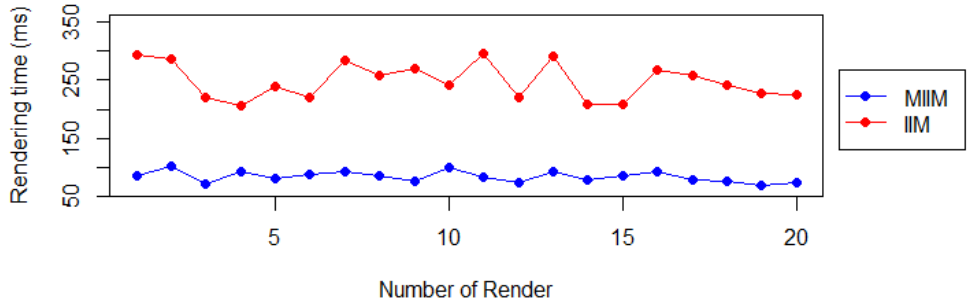
Jako první jsme kód přepsali tak, aby neobsahoval žádné funkce z externích knihoven. Zadefinovali jsme proto v kódu všechny funkce, které bylo potřeba načítat z externího zdroje, čímž jsme dosáhli zrychlení generování. Na základě optimalizace rychlosti jsme pak byli schopni využít větší počet iterací pro stejný vykreslovací čas, z půdních 17 na 18 iteračních průchodů cyklu. Na první pohled nepatrná změna, ale díky tomu můžeme vygenerovat 2^{18} bodů, tedy přesně o 131 072 bodů více. Tento fakt přispěje k lepší aproximaci výsledné množiny, zejména u složitějších struktur.

Další výraznou změnou od původní logiky algoritmu je využití náhodnosti. Konkrétně toto vylepšení spočívá v tom, že při každé iteraci vybereme náhodně pouze jednu větev, tedy buď $\sqrt{z - c}$ (funkce `inverseBranch1`), nebo $-\sqrt{z - c}$ (funkce `inverseBranch2`).

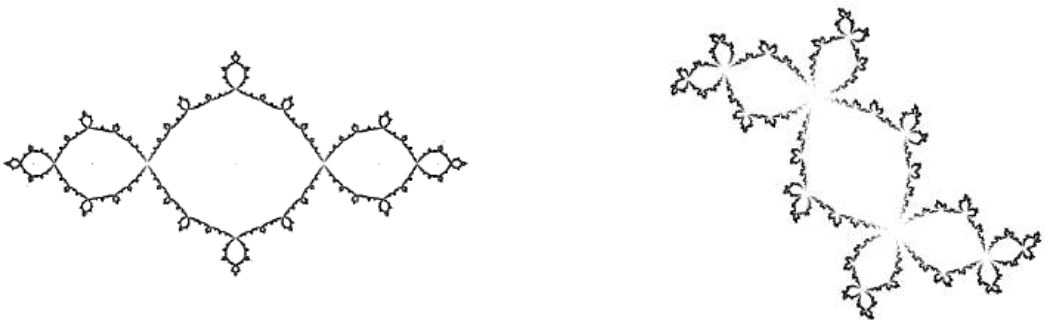
```
1 for (let i = 0; i < iterations; i++) {
2     // Randomly choose one branch
3     const branch = Math.random() < 0.5 ? inverseBranch1 :
          inverseBranch2;
4     z = branch(z, c);
5     points.push(z);
6     }
```

Docílíme tím výrazně většího počtu vykreslených bodů a zároveň zrychlení. Počet bodů se totiž už nebude zvětšovat exponenciálně, jako tomu bylo u základní verze. Provedli jsme 20 zkušebních pozorování, kdy jsme po sobě vygenerovali stejný obrázek se stejnými parametry. Tedy konkrétně Juliovu množinu pro $c = -1$. Jak jde vidět z grafu na obrázku 6.14. Můžeme si všimnout, že modifikovaná verze je v každém pozorování výrazně rychlejší. Její průměrný čas vykreslení z těchto dvaceti pozorování je 83,3 ms, což je výrazně méně, než u její základní verze, jejíž průměr je 247,8 ms.

Výslednou vizualizaci množiny pomocí Modified Inverse Iteration Method pak můžeme vidět na obrázku 6.15. Jak bylo zmíněno, došlo k výraznému zrychlení, co se týče doby vykreslení, na druhou stranu z vizuálního hlediska jsou zde pouze malé rozdíly a metoda má stále v tomto ohledu nedostatky.



Obrázek 6.14: Porovnání výpočetní rychlosti *MIIM* a *IIM*



(a) $c = -1$ **(b)** $c = -0.1256 + 0.7416i$

Obrázek 6.15: Juliovy množiny metodou Modified Inverse Iteration Method pro různé c.

6.5.3 Edge Detection

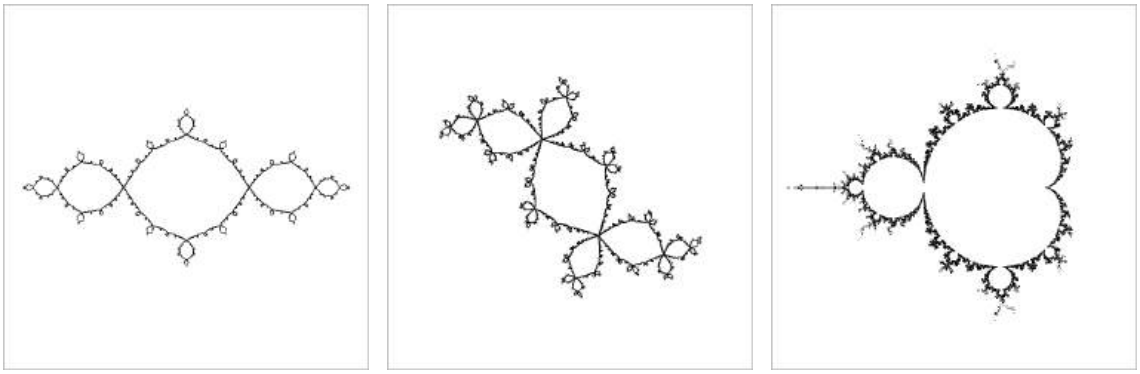
Zajímavým typem algoritmu je tzv. detekce hrany. Pracuje ve dvou fázích. Jako první se provede klasický výpočet založený na počtu iterací, po kterých daný bod diverguje. V závislosti na tomto počtu se následně se využije Sobelův filtr, což je známá metoda pro detekci hran. Tento filtr za pomoci dvou matic, které se aplikují na každý bod obrazu, dokáže zjistit změny intenzity v horizontálním (`sobelX`) a vertikálním směru (`sobelY`). Kombinováním těchto dvou směrů lze získat velikost gradientu, která určuje sílu hrany v daném bodě.

```
const sobelX = [
    [-1, 0, 1],
    [-2, 0, 2],
    [-1, 0, 1]
];

const sobelY = [
    [-1, -2, -1],
    [ 0,  0,  0],
    [ 1,  2,  1]
];
```

Výsledkem aplikace Sobelova filtru je dvoubarevný obraz, kde jsou zvýrazněny hrany (černě), které odpovídají hranici Mandelbrotovy nebo Juliovy množiny viz 6.16. Nyní se už podíváme na samostatný kód algoritmu. Detekci provádíme pomocí funkce `sobelEdgeDetection`, do které se vloží hodnoty počtu iterací pro každý bod komplexní roviny vypočítané dříve. Na základě těchto hodnot a aplikace Sobelových matic se pro každý bod vypočítá gradient. Jak vidíme z `for` cyklu pro Sobelův filtr, počítá se pomocí sousedních pixelů. Pozici aktuálního pixelu bereme jako $(0, 0)$, pixel vlevo od něj jako $(-1, 0)$, a tak dále. Proměnná `intensity` je spjata právě s počtem iterací potřebným k „úniku“ daného bodu a hraje klíčovou roli v následném přiřazení černé, resp. bílé barvy. Následně se vypočítá velikost gradientu v proměnné `gradient` a rozhodne se na základě velikosti této hodnoty, zda-li se jedná o bod hrany (černá barva - 0), nebo pozadí (bílá barva - 255). Jako hraniční hodnota pro rozhodování bylo zvoleno číslo 128, jakožto střed jasu barvy.

```
function sobelEdgeDetection(imageData) {
    //...
    for (let y = 1; y < height - 1; y++) {
        for (let x = 1; x < width - 1; x++) {
            let gX = 0, gY = 0;
            // Apply Sobel filter
            for (let ky = -1; ky <= 1; ky++) {
                for (let kx = -1; kx <= 1; kx++) {
                    const i = ((y + ky) * width + (x + kx)) * 4;
                    gX += sobelX[ky + 1][kx + 1] * intensity;
                    gY += sobelY[ky + 1][kx + 1] * intensity;
                }
            }
            // Magnitude of gradient
            const gradient = Math.sqrt(gX * gX + gY * gY);
```



(a) J_c pro $c = -1$ **(b)** J_c pro $c = -0.1256 + 0.7416i$ **(c)** Mandelbrotova množina

Obrázek 6.16: Metoda Edge Detection.

```
16              const index = (y * width + x) * 4;
17              const edgeValue = gradient > 128 ? 0 : 255;
18              //coloring ...
19          }
20      }
21      return new ImageData(output, width, height);
22  }
```

6.5.4 Rectangle Checking

Tato metoda je optimalizační technika, která zvyšuje efektivitu výpočtu tím, že snižuje počet bodů, které je třeba prakticky počítat pomocí iteračního procesu. Toho se dá zejména využít v oblastech s homogenním počtem iterací, tedy například hlavní kardioid, nebo oblasti daleko od množiny. Tento přístup je založen na dělení obrazu na čtverce a následné analýze těchto čtverců.

Konkrétně metoda Rectangle Checking zjišťuje, zda všechny body na hranicích čtverce mají stejnou hodnotu počtu iterací k, což by znamenalo, že všechny body uvnitř čtverce mají také stejnou hodnotu. Pokud tomu tak je, čtverec se vyplní jednou barvou odpovídající této hodnotě iterací. Pokud hranice čtverce nemají stejnou hodnotu, čtverec se rozdělí na menší části a kontrola se provádí rekurzivně. Vše se opakuje, dokud velikost čtverce nedosáhne předem definovaného minima `minimalRectSize`.

Tento proces funguje tak, že pro každý bod na hranici čtverce se vypočítá počet iterací (v kódu funkce `numberOfIter`). Pokud všechny hraniční body mají stejný počet iterací, pak nám funkce `checkBorder` vrátí pravdivostní hodnotu *true*.

```
1  function checkBorder(x, y, rectWidth, rectHeight) {
2      let iter = numberOfIter(x, y);  // Top-left corner
3      for (let i = 0; i < rectWidth; i++) {
4          if (numberOfIter(x + i, y) !== iter || numberOfIter(x + i, y +
               rectHeight - 1) !== iter) {
5              return false;
6          }
7      }
```

```
8       for (let j = 0; j < rectHeight; j++) {
9           if (numberOfIter(x, y + j) !== iter || numberOfIter(x +
                rectWidth - 1, y + j) !== iter) {
10              return false;
11          }
12      }
13      return true;
14  }
```

Následuje hlavní funkce `rectangleCheck`. Pokud funkce `checkBorder` vrátí pravdivou hodnotu, `rectangleCheck` obarví celý čtverec příslušnou barvou. Pokud ale vrátí hodnotu *false*, `rectangleCheck` rozdělí čtverec na čtyři menší čtverce a pomocí rekurze se proces opakuje až do situace, kdy velikost čtverce dosáhne minimální hodnoty `rectHeight`, resp. `rectWidth`.

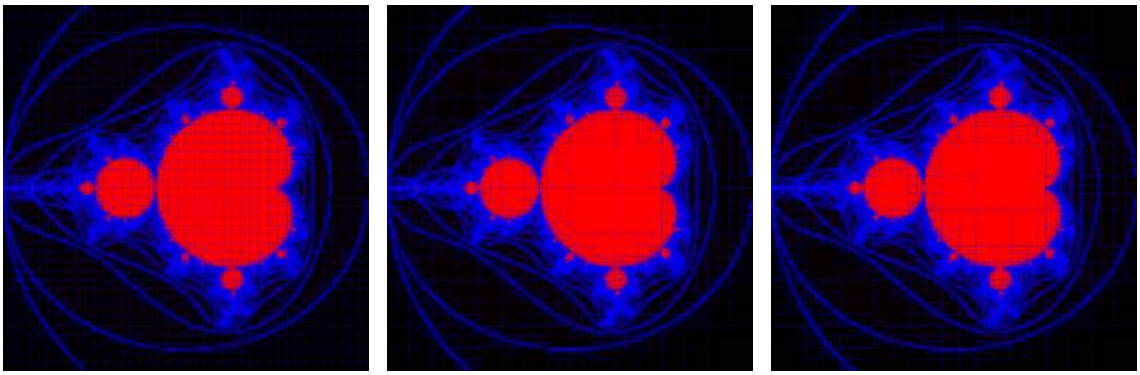
```
1   function advancedRectangleCheck(x, y, rectWidth, rectHeight) {
2       //...
3       if (checkBorder(x, y, rectWidth, rectHeight) || rectWidth <=
            minimalRectSize || rectHeight <= minimalRectSize) {
4           const iter = numberOfIter(x, y);
5           fillRectangle(x, y, rectWidth, rectHeight, iter);
6       } else {
7           const halfWidth = Math.ceil(rectWidth / 2);
8           const halfHeight = Math.ceil(rectHeight / 2);
9           advancedRectangleCheck(x, y, halfWidth, halfHeight);
10          advancedRectangleCheck(x + halfWidth, y, rectWidth - halfWidth,
                halfHeight);
11          advancedRectangleCheck(x, y + halfHeight, halfWidth, rectHeight
                - halfHeight);
12          advancedRectangleCheck(x + halfWidth, y + halfHeight, rectWidth
                - halfWidth, rectHeight - halfHeight);
13      }
14  }
```

Všechno je zabaleno do funkce `rectangleChecking`, která tento proces provede pro všechny čtverce obrázku. Bere také v potaz proměnnou `initialRectSize`, kterou si volí uživatel a představuje maximální velikost čtverce. Pokud si tedy pro vizualizace necháme vykreslit i samotné hranice čtverců, je rozdíl v počáteční podmínce velmi zřetelně viditelný (viz 6.17). Naopak při volbě minimální velikosti čtverce `minimalRectSize` doporučujeme vykreslení hranic zrušit a sledovat rozdíly v detailu vykresleného fraktálu (viz 6.17).

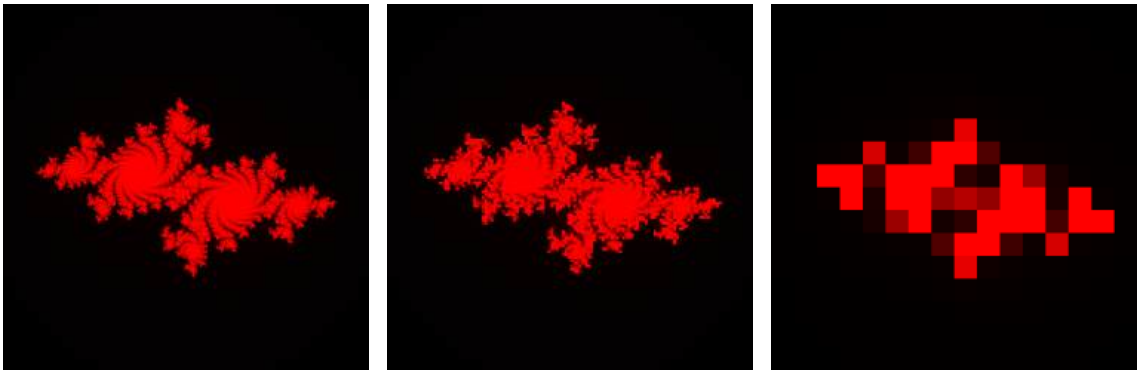
6.5.5 Remaping

Posledním zmíněným algoritmem této části je tzv. Remaping. Nelze jej aplikovat i pro Mandelbrotovu množinu, což vyplývá přímo ze samotného principu algoritmu rozebraného níže. V jednoduchosti se jedná o násobnou transformaci komplexní roviny. Jedná se o jeden z nejvíce názorných algoritmů, který dokáže velmi dobře vysvětlit nejen dynamiku na komplexní rovině, ale také samotný iterační proces.

Algoritmus využívá inverzního iteračního předpisu $\sqrt{z-c}$, který je opakovaně aplikován na všechny body. Velkým rozdílem od ostatních algoritmů je ale



(a) M pro $max = 20$ **(b)** M pro $max = 100$ **(c)** M pro $max = 300$



(d) J_c pro $min = 1$ **(e)** J_c pro $min = 10$ **(f)** J_c pro $min = 50$

Obrázek 6.17: Metoda Rectangle Checking pro různé hodnoty maximální (max) a minimální (min) velikosti čtverců v pixelech.

počáteční stav před první iterací. Běžně se berou postupně všechny body roviny, o kterých nemáme žádnou informaci kromě jejich pozice. To ale neplatí v tomto případě. Zde máme již předem zvýrazněné body náležící kruhu o poloměru 2. Tedy jinými slovy, body, které nepřekročily limitní poloměr divergence. Tyto body pak následně transformujeme. Takže vidíme „cestu" bodů, od kruhu k aproximaci Juliovy množiny. Při aplikování $\sqrt{z-c}$ (v kódu funkce `transform`) je potřeba si uvědomit, že při odmocnění komplexního čísla odmocňujeme jeho velikost (`magnitude`) a dělíme jeho úhel dvěma (`angle`). Pro výpočet je vhodnější uvažovat výsledné komplexní číslo v polárních souřadnicích, tedy $z = |z|(\cos\theta + i\sin\theta)$, takže po odmocnění bude platit $\sqrt{z} = \sqrt{|z|}(\cos\frac{\theta}{2} + i\sin\frac{\theta}{2})$.

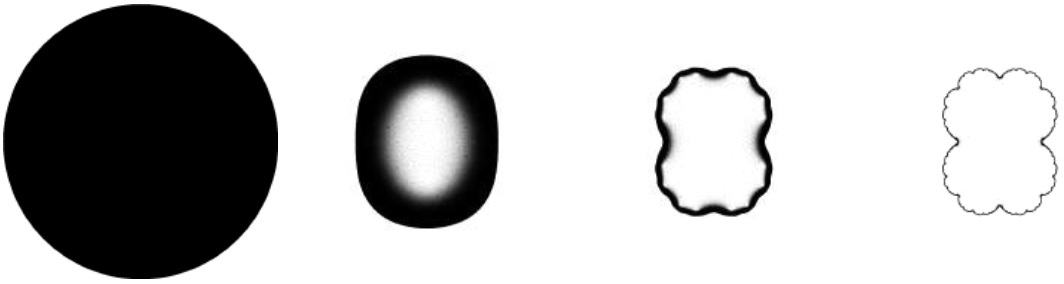
```
function transform(z, c) {
    const zMinusC = {re: z.re - c.re,
                     im: z.im - c.im};
    const magnitude = Math.sqrt(Math.sqrt(zMinusC.re * zMinusC.re +
        zMinusC.im * zMinusC.im));
    const angle = Math.atan2(zMinusC.im, zMinusC.re) / 2;
    return {
        re: magnitude * Math.cos(angle),
        im: magnitude * Math.sin(angle)
        };
}
```

Nyní už přejděme k samotné vizualizaci výsledku. Ta je provedena funkcí `remap`, z které si ukážeme její důležité pasáže. Při pokusu o převedení teorie do praxe vyplynulo na povrch pár problémů. Přitom hned ten první je naprosto zřejmý. Po odmocnění se všechny body transformují do kladné poloroviny ohraničené osou imaginární složky. Využili jsme proto vlastnosti Juliovy množiny, kdy víme, že množiny jsou středově symetrické. Proto je nutné ještě přidat do množiny všech transformovaných bodů `transformed` jejich středově symetrické protějšky `symmetricZ`.

```
points.forEach(point => {
    if (point.color === 'black') {
    const transformed = transform(point.z, c);
    const symmetricZ = { re: -transformed.re, im: -transformed.im };
    transformedPoints.push({ z: transformed, color: 'black' });
    transformedPoints.push({ z: symmetricZ, color: 'black' });
        }
    });
```

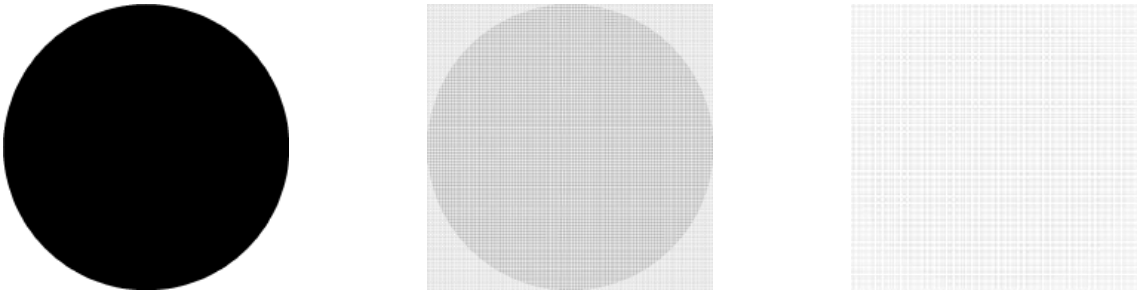
Tímto krokem však nastal druhý problém. Na začátku máme určitý počet bodů. Při každé iteraci a přidání symetrických protějšků bodů se tak počet zdvojnásobí, což zapříčiní exponenciální růst počtu bodů určených pro transformaci. Tento fakt je ale velmi problematický, zejména co se týče výpočetní náročnosti. Nebudeme proto brát pro každou transformaci všechny body, ale jen jejich určitou část.

```
let step = Math.pow(1.4,iterations);
for (let x = 0; x < canvas.width; x += step) {
    for (let y = 0; y < canvas.height; y+= step) {
        const z = toComplex(x, y);
        points.push({ z, color: (z.re * z.re + z.im * z.im <= 4) ? '
            black' : 'white' });
```



(a) $t = 1$ **(b)** $t = 2$ **(c)** $t = 5$ **(d)** $t = 10$

Obrázek 6.18: Juliova množina pro $c = 0{,}25$ metodu Remaping pro různé hodnoty počtu transformací t.



(a) $t = 1$ **(b)** $t = 3$ **(c)** $t = 5$

Obrázek 6.19: Množství bodů v kruhu o poloměru dva určených pro transformaci t v metodě Remaping.

```
6       }
7   }
```

Obecně je totiž kód naprogramován tak, že se zvolí počet iterací `iterations`, tedy počet opakování transformace. Poté se do pole `points` uloží všechna z taková, jejichž velikost je menší nebo rovno než dva jako počáteční body, pro které se začne provádět transformace. Na tomto místě bude dobré si připomenout jednu vlastnost Juliovy množiny, která byla představena v 6.5.2. Juliova množina je uzávěrem všech odpuzujících periodických bodů a navíc jsou v ní tyto body husté. Díky tomu vlastně víme, že body z kruhu se postupně budou stahovat k hranici množiny, tedy vlastně ke korektní Juliově množině, která je definována pouze jako hranice. Jako ilustraci této vlastnosti viz 6.18.

To taky znamená, že nebude potřeba takové množství bodů, abychom hranici spolehlivě vykreslili. K tomu nám slouží proměnná `step`, která v závislosti na počtu iterací určuje velikost kroku, kterým procházíme komplexní rovinu a „sbíráme“ počáteční body kruhu pro transformaci. Lépe se to dá představit díky obrázku 6.19.

Jde hezky vidět, jak hustota bodů rychle klesá. Zvolili jsme exponenciální závislost $1{,}4^{iterations}$, která nejlépe vyhovovala podmínkám. Díky tomu začínáme pro každý počet iterací s jiným počtem bodů, ale máme zaručeno, že pro zvolený

rozsah počtu iterací ve webové aplikace $(0 - 14)$ jich bude pokaždé ve finálním zobrazení dostatek. Na obrázku 6.18 můžeme vidět výsledky pro různý počet transformací.

6.6 Generalized Mandelbrot set

Sekce interpretující teoretické vlastnosti z 2. Nyní si představíme, jakým způsobem jsme tyto vlastnosti implementovali do kódu.

6.6.1 General Exponent

Při vývoji zobecněného algoritmu pro Mandelbrotovu množinu, která se řídí zobrazením $z \mapsto z^d + c$ jsme narazili na zásadní problém. Jakým způsobem vypočítat obecnou mocninu d komplexního čísla z. Pro $d = 2$ bylo řešení snadné. V kódu jsme explicitně zadefinovali vztahy iteračního cyklu komplexního čísla $z = z_x + iz_y$ pro reálnou část z jako $z_x = z_x^2 - z_y^2 + c_x$ a imaginární části z jako $z_y = 2z_x z_y + c_y$. Tento krok pro obecné d ale přestává být triviální, jelikož zřejmě ty stejné vztahy určitě neplatí pro jiné hodnoty d. Popišme si proto nyní určitý vývoj řešení tohoto problému od počátečních pokusů až po finální implementované řešení.

Jak už bylo řečeno, pro každé d budou vztahy pro iterační proces odlišné. Budou obsahovat různý počet členů, jiné konstanty, a tak podobně. Pojďme si proto nyní vypsat pár prvních a prozkoumat jejich strukturu. Pro lepší orientaci ve vztazích jsou jednotlivé části příslušející reálné, resp. imaginární části uzávorkovány.

- Pro $d = 2$:

$$z = z_x^2 + i2z_x z_y - z_z^2 + c = (z_x^2 - z_y^2) + i(2z_x z_y) + c.$$

- Pro $d = 3$:

$$z = z_x^3 + i3z_x^2 z_y - 3z_x z_y^2 - iz_y^3 + c = (z_x^3 - 3z_x z_y^2) + i(3z_x^2 z_y - z_y^3) + c.$$

- Pro $d = 4$:

$$z_{n+1} = z_x^4 + i4z_x^3 z_y - 6z_x^2 z_y^2 - i4z_x z_y^3 + z_y^4 + c = (z_x^4 - 6z_x^2 z_y^2 + z_y^4) + i(4z_x^3 z_y - 4z_x z_y^3) + c.$$

Všimněme si, že konstanty u jednotlivých členů polynomu můžeme postupně psát ve tvaru $\binom{d}{k}$ pro $k = 0, 1, \ldots, d$. Navíc jak můžeme vidět, tak i proměnné se chovají „rozumně“. Dají se opět postupně psát jako $z_x^{d-k} z_y^k$ pro $k = 0, 1, \ldots, d$ a nakonec se navíc vždy přičte c, což ale funguje nezávisle na d. Z toho vyplývá, že obecně tedy můžeme iterační vztah pro $d = n$ zapsat pomocí binomického rozvoje:

$$z = \sum_{k=0}^{d} \binom{d}{k} z_x^{d-k} (iz_y)^k + c.$$

Našli jsme sice určitou pravidelnost, ale nyní je potřeba zjistit, jak tento výsledný vztah rozdělit na reálnou a imaginární část.

Připomeňme, že imaginární jednotka i má známé vlastnosti, že:

$$i^1 = i \quad i^2 = -1, \quad i^3 = -i, \quad i^4 = 1, \quad \text{a obecně} \quad i^k = i^{k \mod 4}.$$

Z toho plyne, že každý člen rozvoje pro k sudé bude patřit do reálné části a naopak každý člen, pro k liché, pak do imaginární části. V obecnosti tedy můžeme psát, že pro znaménka u jednotlivých členů platí:

- Sudé k: $i^k = (-1)^{k/2}$.
- Liché k: $i^k = (-1)^{(k-1)/2}$.

Díky tomu lze rozvoj $z^d + c$ rozdělit na:

$$z^d + c = \mathrm{Re}(z^d + c) + i\ \mathrm{Im}(z^d + c).$$

Nyní si je jen potřeba uvědomit, že rozdělením na dvě části dělíme i maximální hodnotu sumy z d na $\frac{d}{2}$, protože v každé části bereme pouze každé druhé d. Abychom se vyhnuli problémům spojeným právě s dělením číslem dva v případě, že d bude liché, aplikujeme na tuto operaci ještě dolní celou část, tedy $\lfloor \frac{d}{2} \rfloor$. Obecný vztah pro reálnou a imaginární část rozvoje pro $z^d + c$ je po menších úpravách potom tedy:

$$\mathrm{Re}(z^d + c) = \sum_{k=0}^{\lfloor \frac{d}{2} \rfloor} \binom{d}{2k} (-1)^k x^{d-2k} y^{2k} + c_x,$$

$$\mathrm{Im}(z^d + c) = \sum_{k=0}^{\lfloor \frac{d-1}{2} \rfloor} \binom{d}{2k+1} (-1)^k x^{d-(2k+1)} y^{2k+1} + c_y.$$

Ačkoliv se jedná o poměrně sofistikovaný přístup, pro vyšší hodnoty d začíná být výpočet algoritmu velmi časově náročný, a proto tento přístup nakonec implementován do výsledného kódu není.

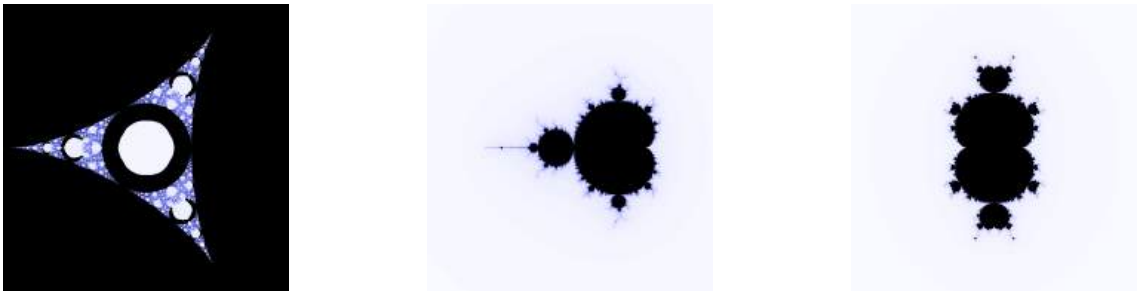
Bylo proto potřeba najít způsob, který nebude tak výrazně závislý na d. Tuto podmínku skvěle splňuje využití polárních souřadnic pro vyjádření z. V tomto tvaru je obecná mocnina z^d snadno definovatelná a navíc rychlost výpočtu prakticky vůbec nezávisí na d. Využijeme proto známého vztahu, který pro obecnou mocninu d je tvaru:

$$z^d = |z|^d e^{id\theta} = |z|^d(\cos(d\theta) + i\sin(d\theta)).$$

Navíc je zřejmě vidět, jak rozdělit vztah na reálnou a imaginární část:

$$\mathrm{Re}(z^d) = |z|^d \cos(d\theta),$$

$$\mathrm{Im}(z^d) = |z|^d \sin(d\theta).$$



(a) $z^{-2} + c$ **(b)** $z^2 + c$ **(c)** $z^3 + c$

Obrázek 6.20: Zobecněná Mandelbrotova množina pro různé exponenty d v předpisu $z^d + c$.

Jelikož při samotné implementaci může uživatel volit libovolné $d \in \mathbb{Q}$, je nutné ošetřit, aby nevznikalo dělení nulou. Uvědomme si ,že při vykreslení Mandelbrotovy množiny je vždy $z_0 = 0$. Potom například $|0|^{-2} = \frac{1}{|0|^2}$, což už není definováno. Proto je potřeba tento případ v kódu podchytit. Výsledná verze vypadá následovně:

```
1 const modulus = Math.sqrt(zRe * zRe + zIm * zIm);
2 const argument = Math.atan2(zIm, zRe);
3 const zReTemp = modulus === 0 ? reC : Math.pow(modulus, d) * Math.cos(d
      * argument) + reC;
4 const zImTemp = modulus === 0 ? imC : Math.pow(modulus, d) * Math.sin(d
      * argument) + imC;
```

Použitím tohoto přístupu výrazně zjednodušíme výpočty, zejména pro vyšší d, včetně desetinných nebo záporných hodnot. Příklady výsledné vizualizace viz 6.20.

6.7 Generalized Julia set

6.7.1 Generalized Exponent

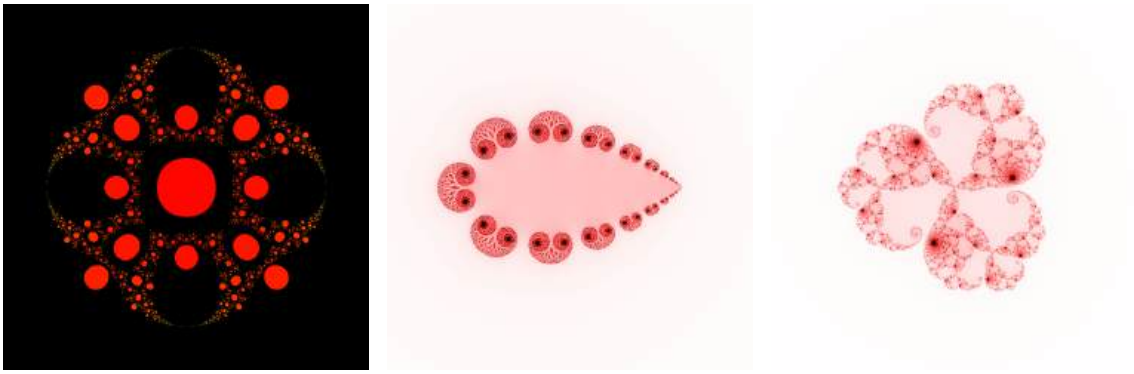
Vycházíme ze stejného principu jako u Generalized Mandelbrot set 6.6.1, proto nebudeme vše, co zde bylo již zmíněno, opakovat. Jen připomeňme, že u Juliovy množiny volíme navíc konstantu c, pro kterou generujeme výsledný fraktál. Pojďme si však na obrázku 6.21 ukázat pár výsledků tohoto kódu pro různé exponenty d a konstanty c.

6.7.2 Sinus

Podívejme se nyní na Juliovy množiny pro nepolynomiální předpis. V tomto případě jde o funkci

$$f(z) = c\sin(z). \tag{6.3}$$

Postup je obdobný jako pro kvadratické zobrazení $z^2 + c$. Zvolíme konstantu c a pro každý bod z_0 iterujeme funkci 6.3. Chování posloupnosti bodů z jednotlivých iteračních kroků určí, zda bod z_0 patří do Juliovy množiny.



(a) $z^{-4} - 0{,}791$ **(b)** $z^{1,5} - 0{,}1948$ **(c)** $z^3 + 0{,}4 + 0{,}002275i$

Obrázek 6.21: Zobecněná Juliova množina pro různé exponenty d a konstanty c v předpisu $z^d + c$.

Mírný rozdíl však existuje při kontrole, zda-li bod můžeme označit jako divergentní ještě před dosažením maximálního počtu iterací. Běžně se ptáme, jestli velikost hodnoty čísla z po k iteracích $|z_k|$ překročí určitý limitní poloměr divergence r_c (`r_c`). U sinu stačí kontrolovat pouze imaginární část z (v kódu `zIm`). Limitní poloměr divergence se běžně volí jako 50. Komplexní sinus lze také vyjádřit takto:

$$c\sin(z) = (c_x + ic_y)(\sin(z_x)\cosh(z_y) + i\cos(z_x)\sinh(z_y)),$$

kde funkce $\cosh$, resp. $\sinh$ představují hyperbolický kosinus, resp. sinus. Tento vztah jde rozdělit na reálnou a imaginární část, což bude velmi výhodné při implementaci:

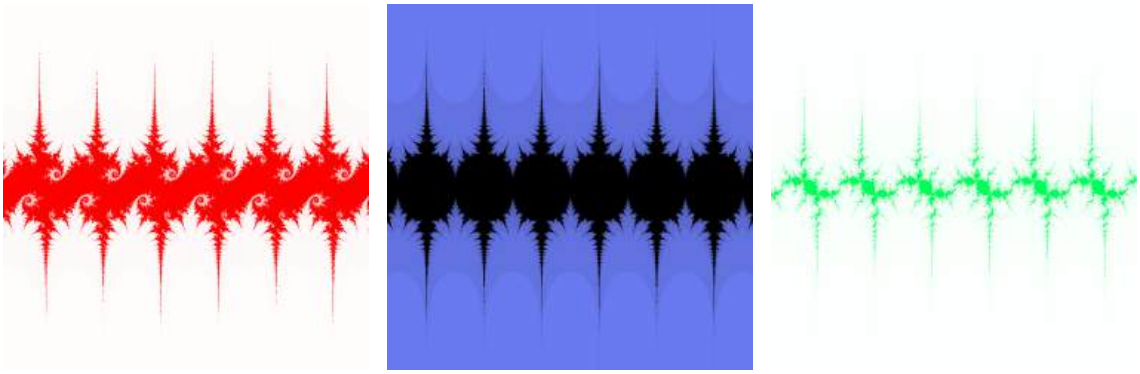
$$\mathrm{Re}(c\sin(z)) = c_x \sin(z_x)\cosh(z_y) - c_y \cos(z_x)\sinh(z_y)$$

$$\mathrm{Im}(c\sin(z)) = c_x \cos(z_x)\sinh(z_y) + c_y \sin(z_x)\cosh(z_y)$$

V kódu potom iterační smyčka vypadá následovně:

```
1 while (Math.abs(zIm) < r_c && iter < maxIter) {
2     const sinRe = Math.sin(zRe) * Math.cosh(zIm);
3     const sinIm = Math.cos(zRe) * Math.sinh(zIm);
4     const newZRe = cRe * sinRe - cIm * sinIm;
5     const newZIm = cRe * sinIm + cIm * sinRe;
6     //...
7 }
```

Následně pak už samotná vizualizace, která funguje stejným způsobem jako u běžné Juliovy množiny. Na obrázku 6.22 můžeme vidět opět pár příkladů. Všimněme si periodicity vykresleného fraktálu, kterým je funkce $\sin$ typická. V části 6.22c jde pak hezky vidět i tvar klasické Juliovy množiny, periodicky se opakující ve směru reálné osy.



(a) $c = 0{,}984808 + 0{,}173648i$ **(b)** $c = 1$ **(c)** $c = 1{,}87939 + 0{,}68404i$

Obrázek 6.22: Zobecněná Juliova množina funkce $c\sin(z)$ pro různé konstanty c.

6.7.3 Cosinus

Juliova množina pro komplexní kosinus je založena na prakticky stejném způsobu jako sinus. Její předpis je tvaru:

$$c\cos(z) = (c_x + ic_y)\left(\cos(z_x)\cosh(z_y) - i\sin(z_x)\sinh(z_y)\right),$$

kde

$$\begin{aligned}\mathrm{Re}(c\cos(z)) &= c_x\cos(z_x)\cosh(z_y) + c_y\sin(z_x)\sinh(z_y),\\ \mathrm{Im}(c\cos(z)) &= -c_x\sin(z_x)\sinh(z_y) + c_y\cos(z_x)\cosh(z_y).\end{aligned}$$

Obdobně využíváme také kontroly pouze imaginární části z, pro určení divergence bodu. Kód je potom jen s malými změnami následující:

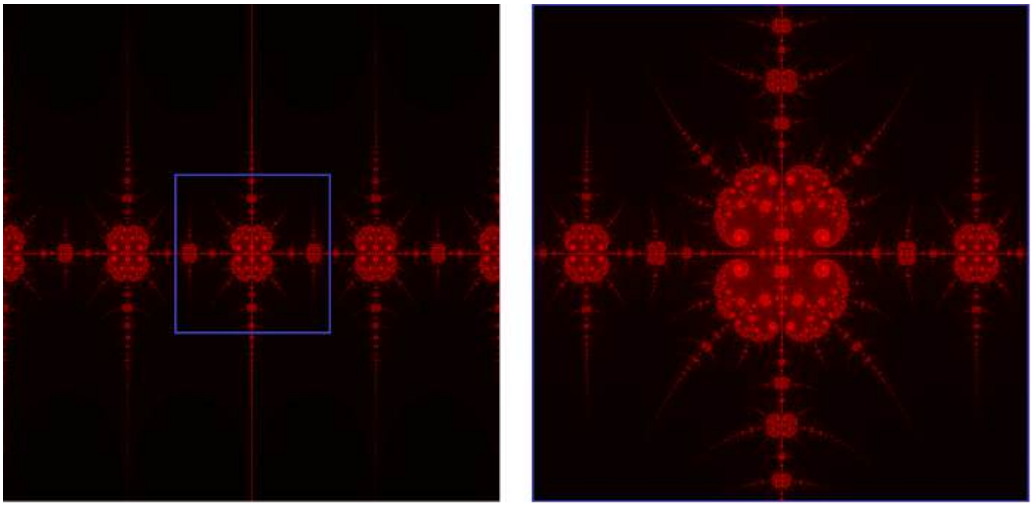
```
while (Math.abs(zIm) < r_c && iter < maxIter) {
    const cosRe = Math.cos(zRe) * Math.cosh(zIm);
    const cosIm = - Math.sin(zRe) * Math.sinh(zIm);
    const newZRe = cRe * cosRe - cIm * cosIm;
    const newZIm = cRe * cosIm + cIm * cosRe;
    //...
}
```

Obrázek Juliovy množiny pro komplexní kosinus máme zde 6.23. Je také možnost vidět přiblížení oblasti okolo středu, kde znovu můžeme zřetelně rozpoznat jak tvar Juliovy množiny kvadratického zobrazení, tak fraktální strukturu celého obrazce, kdy se tvar uprostřed nekonečněkrát opakuje i všude okolo.

6.7.4 Exponential

Poslední variantou zobecnění pro Juliovy množiny je exponenciální funkce, kterou lze vyjádřit jako

$$f(z) = \lambda e^z,$$



Obrázek 6.23: Zobecněná Juliova množina funkce $c\cos(z)$ pro $c = 2{,}95$ vlevo. Vpravo přiblížený modrý výřez.

kde $\lambda = \lambda_x + i\lambda_y$ (v kódu `lambdax`, `lambday`). Pro danou funkci platí podle [29] následující rozložení na reálnou a imaginární část:

$$\begin{aligned}\mathrm{Re}(\lambda e^z) &= \alpha e^{u_x}(\lambda_x \cos(u_y) - \lambda_y \sin(u_y)) + (1-\alpha)\,\mathrm{Re}(z_0),\\ \mathrm{Im}(\lambda e^z) &= \alpha e^{u_x}(\lambda_x \sin(u_y) + \lambda_y \cos(u_y)) + (1-\alpha)\,\mathrm{Im}(z_0),\end{aligned}$$

kde $\alpha \in [0,1]$ je váhový parametr (`alpha`) a z_0 je počáteční hodnota (`zReInitial` a `zImInitial`).

```
1 zRe = alpha * (Math.exp(uRe) * (lambdax * Math.cos(uIm) - lambday *
     Math.sin(uIm))) + (1 - alpha) * zReInitial;
2 zIm = alpha * (Math.exp(uRe) * (lambdax * Math.sin(uIm) + lambday *
     Math.cos(uIm))) + (1 - alpha) * zImInitial;
```

Proměnné u_x, u_y (`uRe` a `uIm`) jsou transformace obsahující parametry λ_x, λ_y a β.

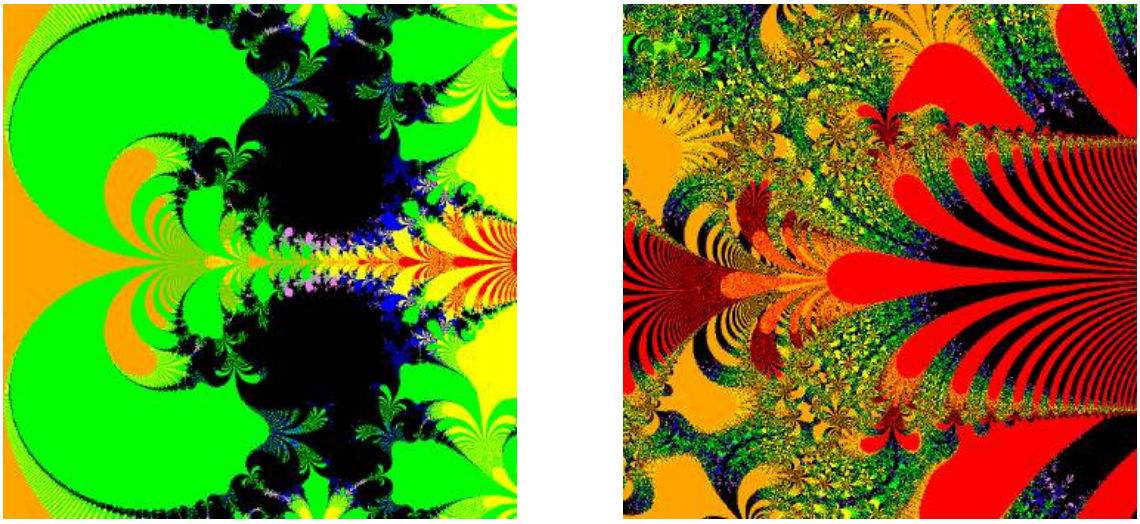
$$\begin{aligned}u_x &= \beta \exp(z_x)(\lambda_x \cos(z_y) - \lambda_y \sin(z_y)) + (1-\beta)z_x,\\ u_y &= \beta \exp(z_x)(\lambda_x \sin(z_y) + \lambda_y \cos(z_y)) + (1-\beta)z_y.\end{aligned}$$

V kódu tyto vztahy pak vypadají následovně:

```
1 let uRe = beta * (Math.exp(zRe) * (lambdax * Math.cos(zIm) - lambday *
     Math.sin(zIm))) + (1 - beta) * zRe;
2 let uIm = beta * (Math.exp(zRe) * (lambdax * Math.sin(zIm) + lambday *
     Math.cos(zIm))) + (1 - beta) * zIm;
```

Iterační smyčku v kódu provádíme pak běžným způsobem. Buď bod dosáhne maximálního počtu iterací `maxIter`, nebo velikost jeho reálné části z překročí limitní poloměr divergence `r_c`.

Následně už proběhne samotná vizualizace, v které jsme využili speciální ba-



(a) $\lambda = 0{,}4678;\ \alpha = 1;\ \beta = 0{,}3.$ **(b)** $\lambda = 2\pi + i;\ \alpha = 0{,}6;\ \beta = 0{,}8.$

Obrázek 6.24: Juliova množina funkce $\lambda \exp z$ pro různou volbu parametrů λ, α a β.

revnou paletu pro zvýraznění rychlosti divergence jednotlivých bodů takto:

$$\text{Barva} = \begin{cases} \text{Červená} & \text{pokud } \texttt{iter} < 0.2 \cdot \texttt{maxIter}, \\ \text{Oranžová} & \text{pokud } 0.2 \cdot \texttt{maxIter} \leq \texttt{iter} < 0.4 \cdot \texttt{maxIter}, \\ \text{Žlutá} & \text{pokud } 0.4 \cdot \texttt{maxIter} \leq \texttt{iter} < 0.6 \cdot \texttt{maxIter}, \\ \text{Zelená} & \text{pokud } 0.6 \cdot \texttt{maxIter} \leq \texttt{iter} < 0.8 \cdot \texttt{maxIter}, \\ \text{Modrá} & \text{pokud } 0.8 \cdot \texttt{maxIter} \leq \texttt{iter} < 0.9 \cdot \texttt{maxIter}, \\ \text{Fialová} & \text{pokud } 0.9 \cdot \texttt{maxIter} \leq \texttt{iter} < \texttt{maxIter}, \\ \text{Černá} & \text{pokud } \texttt{iter} == \texttt{maxIter}. \end{cases}$$

Příklady Juliových množin exponenciální funkce můžeme vidět na obrázku 6.24.

6.8 Shrnutí

Webová aplikace Fractal Generator neslouží jen jako pomůcka pro zkoumání fraktálních obrazců z hlediska matematiky, ale také jako nástroj pro možnou analýzu a srovnání z pohledu informatiky. V určitých případech je tedy záměr, že je vykreslení pomalejší nebo není dostatečně vizuálně propracované. Cílem je mimo jiné zdůraznit rozdíly v různých přístupech při využití stejných počátečních podmínek. Uživatel si tak dokáže uvědomit silné a slabé stránky jednotlivých metod a algoritmů.

Závěr

Hlavním cílem této práce bylo pokračovat ve studiu a generalizaci Juliovy a Mandelbrotovy množiny. Zkoumat jejich vlastnosti a na základě nich pak tyto fraktály vizualizovat. Dále bylo úkolem podrobně rozebrat pojem periodické komponenty. Praktická část spočívala v navržení a naprogramování algoritmů a metod pro vizualizaci Juliových a Mandelbrotových množin, na jejichž základě vznikla webová aplikace dostupná na https://www.math.muni.cz/~xmacharacekj/. Veškeré zdrojové kódy jsou k nalezení zde https://is.muni.cz/auth/th/b3ot8/.

Pomocí první kapitoly jsme poskytli vědomostní základ sloužící k jednodušššímu pochopení rozebíraného tématu. Ve druhé kapitole jsme zobecnili Mandelbrotovu množinu pro funkce s racionálním exponentem. Podrobně zkoumali její vlastnosti a nastalé problémy pro neceločíselné hodnoty d, způsobené větvením. Následně jsme ukázali transformaci komplexních odmocnin z jedné na bifurkační body Mandelbrotovy množiny a připravili si tak základ pro nadcházející kapitolu.

Ve třetí kapitole jsme se zabývali pojmem periodické komponenty. Formálně jsme je zadefinovali, a pak zkoumali jejich pozici vůči Mandelbrotově množině. Následně jsme představili způsob, jak spočítat jejich středy a také ukázali vztah pro výpočet celkového počtu periodických komponent. Na závěr jsme zavedli Newtonovu metodu, která se ukázala jako prospěšná při samotných výpočtech a uvedli možné zjednodušení při volbě počáteční hodnoty metody.

V kapitole čtyři jsme zadefinovali Juliovy množiny transcendentních funkcí a polynomických funkcí pro racionální exponent. Pro všechny jsme také představili limitní poloměr divergence, což je klíčový pojem při vizualizacích. Dále jsme zkoumali jejich vlastnosti a také se zaměřili na problematiku větvového řezu, což nastává pro neceločíselný exponent polynomických funkcí.

V praktické části jsme vyvinuli webovou aplikaci *Fractal Generator*. Vznikl tak vysoce interaktivní nástroj pro analýzu Juliových a Mandelbrotových množin s možností porovnání všech obarvovacích metod a vizualizačních algoritmů zmíněných v práci. Poté, co jsme si představili strukturu aplikace a popsali její jednotlivé části, jsme se následně seznámili s metodami a algoritmy. Dokázali jsme výrazně vylepšit výkon některých algoritmů z bakalářské práce a také implementovali různé varianty optimalizace i na nové algoritmy. Z odstínů šedi jsme aplikaci rozšířili o různé barevné palety. Povedla se nám implementovat funkcionalita přiblížení a posunu. V neposlední řadě jsme velkou část parametrů, které byly v bakalářské práci pouze jako konstantní hodnoty v kódů, dovolili měnit ze strany uživatele, čímž může být analýza ještě podrobnější a zajímavější.

Seznam použité literatury

[1] BEARDON, Alan F. *Iteration of rational functions: complex analytic dynamical systems*. Graduate Texts in Mathematics. New York: Springer, 2000. ISBN 0-387-95151-2.

[2] BROWN, James Ward a CHURCHILL, uel V. *COMPLEX VARIABLES AND APPLICATIONS*. 8th ed. McGraw Hill, 2009. ISBN 978-0-07-305194-9.

[3] BUDÍKOVÁ, Irena. *Polynomy: Text určený studentům učitelství matematiky*. 2014.

[4] CARLESON, Lennart a GAMELIN, Theodore W. *Complex Dynamics*. Universitext: Tracts in Mathem. New York: Springer, 1993. ISBN 0-387-97942-5.

[5] CUNNINGHAM, Adam. *Displaying the Internal Structure of the Mandelbrot Set*. State University of New York at Buffalo, 2013.

[6] DEVANEY, Robert L. Julia sets and bifurcation diagrams for exponencial maps. *American Mathematical Society*. 1984, roč. 11, č. 1, s. 167-171.

[7] DRAKOPOULOS, V. *COMPARING RENDERING METHODS FOR JULIA SETS*. Department of Informatics and Telecommunications, Theoretical Informatics, 2002.

[8] EBERLEIN, Dominik; MUKHERJEE, Sabyasachi a SCHLEICHER, Dierk. Rational Parameter Rays of the Multibrot Sets. In: HAGEN, Thomas a RUPP, Florian. *Dynamical Systems, Number Theory and Applications*. World Scientific, 2016, s. 49-84. ISBN 978-981-4699-86-4.

[9] FINOL, Gerard; PARÍS, Gerard; L-PEZ, Pedro García a SÁNCHEZ-ARTIGAS, Marc. *Exploiting Inherent Elasticity of Serverless in Irregular Algorithms*. Universitat Rovira i Virgili, Tarragona, Spain, 2022.

[10] GARCIA, Francisco; FERNANDEZ, Angel; BARRALLO, Javier a MARTIN, Luis. *COLORING DYNAMICAL SYSTEMS IN THE COMPLEX PLANE*. The University of the Basque Country, 2008.

[11] GARRETT, Paul. *Roots of unity*. 2005. Dostupné také z: https://www.academia.edu/6830854/Roots_of_unity.

[12] GONZALEZ, Rafael C. a WOODS, Richard E. *Digital image processing*. 3rd ed. Upper Saddle River, NJ: Prentice-Hall, 2008. ISBN 9780131687288.

[13] GUBNER, John A. *Magnitude and Phase of Complex Numbers*. Department of Electrical and Computer Engineering, University of Wisconsin—Madison, 2021.

[14] GUJAR, Uday G.; BHAVSAR, Virendra C. a VANGALA, Nagarjuna. Fractal from $z \leftarrow z^a + c$ In the Complex z-Plane. 1990.

[15] GUJAR, Uday G.; BHAVSAR, Virendra C. Fractal from $z \leftarrow z^a + c$ In the Complex c-Plane. 1988.

[16] KALAS, Josef. *Analýza v komplexním oboru*. Brno: Masarykova univerzita v Brně, 2006, iv, 202 s. ISBN 80-210-4045-9.

[17] KARLTORP, Johan Djärv a SKOGLUND, Eric. *Performance of Multi-threaded Web Applications using Web Workers in Client-side JavaScript*. Bakalářské práce. Karlskrona, Sweden: Faculty of Computing, Blekinge Institute of Technology, 2020.

[18] KISAKA, Masashi. On the connectivity of Julia sets of transcendental entire functions. *Cambridge University Press*. 1998, č. 18, s. 189-205.

[19] KNILL, Oliver. *Dynamical Systems* [online]. Cambridge (Massachuttes): Harvard University, 2005, 179 s. Dostupné z: https://people.math.harvard.edu/~knill/teaching/math118/118_dynamicalsystems.pdf

[20] KUJUR, Priyanka. Performance Evaluation of Edge Detection Techniques for Fractal Images. *International Journal of Research in Advent Technology*. 2020, roč. 8, č. 6, s. 34-39.

[21] LAU, E. a SCHLEICHER, D. Symmetries of fractals revisited. *The Mathematical Intelligencer*. 1996, č. 18, s. 45-51.

[22] LUTZKY, M. Counting hyperbolic components of the Mandelbrot set. *Physics Letters A*. 1993, č. 177, s. 338-340.

[23] MACHARÁČEK, Jan. *Juliovy množiny*. Bakalářské práce. Přírodovědecká fakulta, Masarykova univerzita Ústav matematiky a statistiky: Masarykova univerzita, 2022.

[24] MCCLURE, Mark. *Inverse Iteration Algorithms for Julia Sets*. 2016.

[25] PEITGEN, Heinz-Otto a RICHTER, H. Peter. *Beauty of fractals: images of complex dynamical systems*. Berlin: Springer-Verlag, 1986. ISBN isbn3-540-15851-0.

[26] PEITGEN, Heinz-Otto; JÜRGENS, Hartmut a SAUPE, Dietmar. *Chaos and Fractals: New Frontiers of Science*. 2nd ed. New York: Springer-Verlag, 1992. ISBN 0-387-20229-3.

[27] PEITGEN, Heiz-Otto a SAUPE, Dietmar. *Science of fractal images*. New York: Springer-Verlag, 1988. ISBN 0-387-96608-0.

[28] PÉREZ, Lluís Pastor. *Local connectivity of Julia set*. Diplomová práce. Barcelona: Departament de matemàtiques i informàtica, 2021.

[29] PRASAD, Bhagwati a KATIYAR, Kuldip. Fractals via Ishikawa Iteration. In: *Communications in Computer and Information Science*. Springer, 2011, s. 197-203. ISBN 978-3-642-19262-3.

[30] PŘIBYLOVÁ, Lenka. Bifurcations of One-Dimensional One-Parametric Maps Revisited. *Chaotic Modeling and Simulation International Conference*. 2019, č. 11, s. 215-227.

[31] STRATMANN, Lukas. *Conceptualization and Prototypical Implementation in WebGL of an Exercise in Color Theory*. Paderborn, 2017, viii, 66 s. Dostupné také z: https://color.lukas-stratmann.com/downloads/thesis.pdf. Bakalářská práce. Paderborn university, Department of Electrical Engineering, Computer Science, and Mathematics. Vedoucí práce Sabrina Heppner.

[32] SUTHERLAND, Scoot. *Finding Roots of Complex Polynomials with Newtons Method*. 1989.

[33] WEISS, Stewart. *Binary Number System*. 2006.

[34] WIERSMA, A. G. *The Complex Dynamics of Newton's Method*. Bakalářská práce, vedoucí Dr. A.E. Sterk. University of Groningen: University of Groningen, Faculty of mathematics and natural sciences, 2016.

[35] XINGYUAN, Wang; YUANYUAN, Sun a YIJIE, He. Accurate Computation of Periodic Regions- Centers in the General M-Set with Integer Index Number. *Discrete Dynamics in Nature and Society*. 2010, roč. 2010, s. 1-12.

[36] *How to map colors in the Mandelbrot set?* Online. FRACTALFORUMS.COM. 2007. Dostupné z: https://web.archive.org/web/20220909215450/http://www.fractalforums.com/programming/newbie-how-to-map-colors-in-the-mandelbrot-set/msg3465/#msg3465. [cit. 2024-12-29].