T.R.

GEBZE TECHNICAL UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

POLYNOMIOGRAPHY

İSMAİL TAPAN

SUPERVISOR
DR. TÜLAY AYYILDIZ AKOĞLU

GEBZE
2023

T.R.
GEBZE TECHNICAL UNIVERSITY
FACULTY OF ENGINEERING
COMPUTER ENGINEERING DEPARTMENT

POLYNOMIOGRAPHY

İSMAİL TAPAN

SUPERVISOR
DR. TÜLAY AYYILDIZ AKOĞLU

2023
GEBZE

This study has been accepted as an Undergraduate Graduation Project in the Department of Computer Engineering on 19/01/2023 by the following jury.

JURY

Member
(Supervisor) : Dr. Tülay Ayyıldız Akoğlu

Member : Prof. Dr. Didem GÖZÜPEK

ABSTRACT

Polynomiography is a a visual art form that involves visualization of complex polynomials.

...

In this work we will explore new methods to generate artistic images/image sequences from polynomials. We will create a Python library to help people use and improve these methods for generating artistic images from polynomials. Our goal is to encourage more research and experimentation in this area by sharing these methods as open-source software.

Keywords: Polynomial, Polynomiography, Digital Art

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1. Polynomials

// TODO: extend this to polynomial rings

A polynomial is a mathematical expression of the form:

$$a_n z^n + a_{n-1} z^{n-1} + \cdots + a_1 z + a_0$$

where a_n, a_{n-1}, $\ldots$, a_0 are coefficients, n is the degree, a_n is nonzero, and z is a variable.

The degree of a polynomial is the highest power of the variable z in the expression. For example, the polynomial $3z^2 + 2z - 1$ has degree 2, while the polynomial $4z^3 + 7z^2 - 5z + 1$ has degree 3.

A polynomial equation is an equation of the form:

$$a_n z^n + a_{n-1} z^{n-1} + \cdots + a_1 z + a_0 = 0$$

A solution, or root, of a polynomial is any specific value of z that would satisfy the polynomial equation.

1.1.1. Polynomial Rings over Finite Fields

A polynomial ring over a finite field is a set of polynomials with coefficients taken from a finite field. Given a finite field F_q with q elements, the polynomial ring over F_q is denoted by $F_q[x]$ and the elements of $F_q[x]$ are polynomials of the form:

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

where $a_0, a_1, \ldots, a_n \in F_q$ and n is a non-negative integer.

(e.g.) All degree 3 polynomials of $F_q[x]$ where $F_q = 0, 1$:

- x^3
- $x^3 + x$
- $x^3 + x^2$
- $x^3 + x^2 + x$
- $x^3 + 1$
- $x^3 + x + 1$
- $x^3 + x^2 + 1$
- $x^3 + x^2 + x + 1$

1.2. Root finding algorithms

In mathematics and computing, a root algorithm is an algorithm for finding roots of continuous polynomial functions. Most of the root finding algorithms are iteration based, they require one or more initial values and try to converge to the a root of the function in each iteration. These methods don't find an exact solution but an approximation to a root.

1.2.1. Iterative methods

1.2.1.1. Householder's methods (Derivative Based Methods)

It's an iterative method that uses this generic formula to converge to a root:

$$x_{n+1} = x_n + d\frac{(1/f)^{(d-1)}(x_n)}{(1/f)^{(d)}(x_n)}$$

where $f^{(n)}$ denotes the (n)th derivative of f.

The methods that uses first two order of this formula have special names, Newton's Method and Halley's method respectively.

1.2.1.1.1 Newton's Method

If you plug 1 as d in the previous formula, you get the following formula:

$$x_{n+1} = x_n + \frac{(1/f)(x_n)}{(1/f)^{(1)}(x_n)}$$

$$x_{n+1} = x_n + \frac{\frac{1}{f(x_n)}}{-\frac{f'(x_n)}{f^2(x_n)}}$$

$$x_{n+1} = x_n + \frac{1}{f(x_n)}(-\frac{f^2(x_n)}{f'(x_n)})$$

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

So the algorithm will be as follow:

1 – Start with some x
2 – Iterate the following formula until the difference between x_n and x_{n+1} is less then some ϵ
$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$
3 – Return the approximated value [and iteration count]

1.2.1.1.2 Halley's Method

If you plug 2 as d in the previous formula, you get the following formula:

$$x_{n+1} = x_n + \frac{(1/f)^{(1)}(x_n)}{(1/f)^{(2)}(x_n)}$$

$$x_{n+1} = x_n + \frac{-\frac{f'(x_n)}{f^2(x_n)}}{\frac{f''(x_n)}{f^2(x_n)} - 2\frac{f'(x_n)f'(x_n)}{f^3(x_n)}}$$

$$x_{n+1} = x_n + (-\frac{f'(x_n)}{f^2(x_n)})(\frac{f''(x_n)}{f^2(x_n)} - 2\frac{f'(x_n)f'(x_n)}{f^3(x_n)})^{-1}$$

$$x_{n+1} = x_n + (-\frac{f'(x_n)}{f^2(x_n)})(\frac{f(x_n)f''(x_n) - 2f'(x_n)f'(x_n)}{f^3(x_n)})^{-1}$$

$$x_{n+1} = x_n + (-\frac{f'(x_n)}{f^2(x_n)})(\frac{f^3(x_n)}{f(x_n)f''(x_n) - 2f'(x_n)f'(x_n)})$$

$$x_{n+1} = x_n - \frac{f(x_n)f'(x_n)}{2f'(x_n)f'(x_n) - f(x_n)f''(x_n)}$$

So the algorithm will be as follow:

1 – Start with some x
2 – Iterate the following formula until the difference between x_n and x_{n+1} is less then some ϵ
$x_{n+1} = x_n - \frac{f(x_n)f'(x_n)}{2f'(x_n)f'(x_n) - f(x_n)f''(x_n)}$
3 – Return the approximated value [and iteration count]

1.2.1.1.3 Others

This method can be implemented for higher orders too. The only thing that will change is the formula that computes the next iteration in the algorithm.

A general algorithm can be written as:

1 – Start with some x
2 – Iterate the following formula until the difference between x_n and x_{n+1} is less then some ϵ
$x_{n+1} =$ //x_n plugged into derived formula
3 – Return the approximated value [and iteration count]

1.2.1.2. Steffensen's Method

// TODO: maybe explain how it is obtained or cite

In this method the next iteration is computed with the following formula:

$$x_{n+1} = x_n - \frac{f(x_n)}{\frac{f(x_n + f(x_n))}{f(x_n)} - 1}$$

So the algorithm will be as follow:

1 – Start with some x
2 – Iterate the following formula until the difference between x_n and x_{n+1} is less then some ϵ

$$x_{n+1} = x_n - \frac{f(x_n)}{\frac{f(x_n + f(x_n))}{f(x_n)} - 1}$$

3 – Return the approximated value [and iteration count]

1.2.1.3. Secant Method

// TODO: implement

1.2.1.4. Inverse Interpolation

// TODO: implement

1.2.2. Bracketing methods

// note sure if it's possible to implement bracketing methods for polynomiography since we will need one input that gives positive and one input that gives negative result to start

1.2.2.1. Bisection Method

// TODO: implement

1.2.2.2. False Position (Regula Falsi)

// TODO: implement

1.2.3. Combinations of methods

1.2.3.1. Brent's Method

Combination of bisection method, the secant method and inverse quadratic interpolation.

// note sure if it's possible to implement bracketing methods for polynomiography since we will need one input that gives positive and one input that gives negative result to start

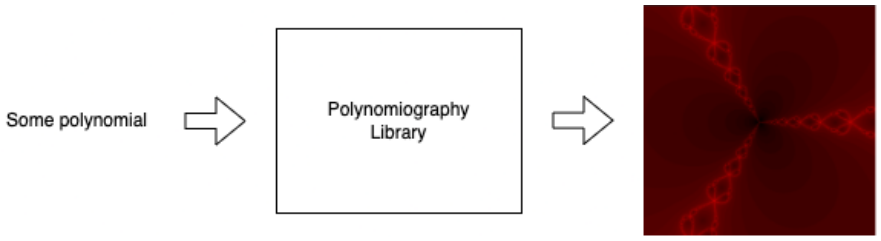
// TODO: implement

2. IMPLEMENTATION

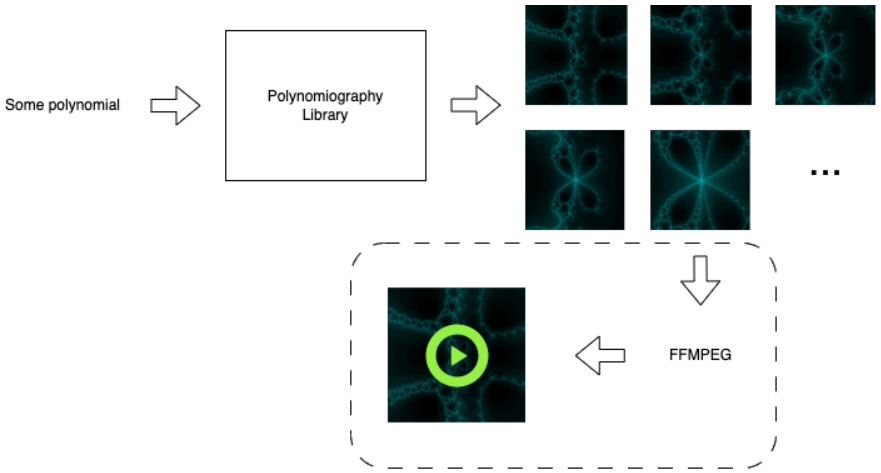
2.1. Project Design

// Update these pictures

Use Case 1



Use Case 2



Use Case 3

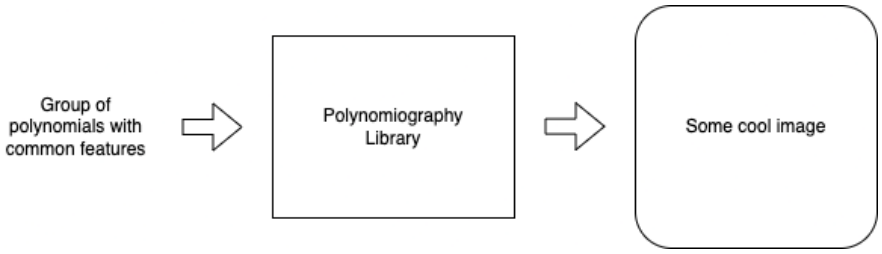


Figure 2.1: High Level Project Design

2.2. Implementation Details

2.2.1. Canvas $\iff$ Complex Plane

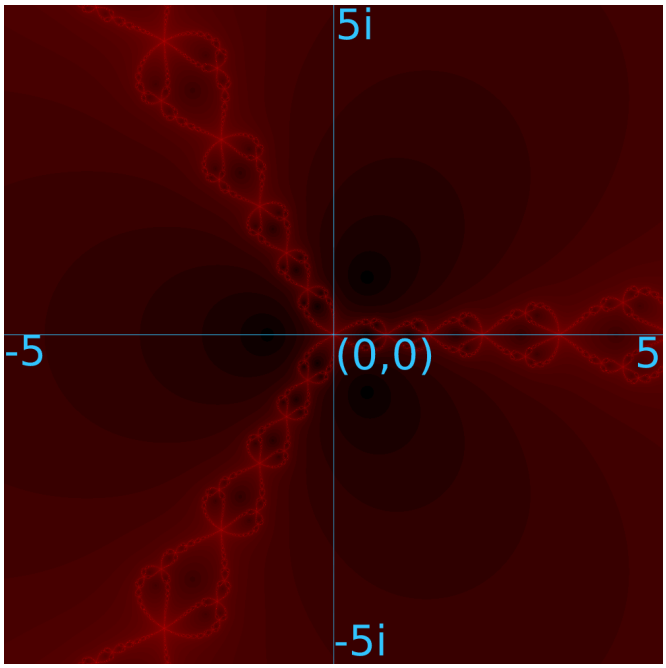


Figure 2.2: Complex plane on canvas

2.2.2. Visualization of Iterative Root Finding Methods

// Elaborate
- For each pixel find the corresponding complex number
- Start the iterative method with the corresponding number
- Get the total iteration count until it converges
- Assign a color to this pixel with respect to iteration count

2.2.3. Visualization of Roots of Polynomial Rings Over Finite Fields

// Elaborate
- Compute the roots for each polynomial in finite field
- Iterate over each root, find the corresponding pixel from the complex number
- Increase the color on that pixel

2.3. Usage

For detailed usage see API Documentation

3. RESULTS

4. CONCLUSIONS

APPENDICES

Appendix 1: API Documentation

// Documentation of the Polynomiogra**py** library