Міністерство освіти і науки України
НТУУ «КПІ ім. Ігоря Сікорського»
Навчально-науковий інститут атомної та теплової енергетики
Кафедра цифрових технологій в енергетиці

Лабораторна робота №1

з дисципліни «Геометричне моделювання та комп'ютерна графіка. Частина 2. Побудова реалістичних зображень»

Варіант №17

Студента 4-го курсу НН ІАТЕ гр. ТР-12
Ковальова Олександра
Перевірила: доц. Сидоренко Ю. В.

КИЇВ 2025

Завдання. Побудувати функцію методом МНК першого та другого порядку за такими даними:

$$x^3 + x$$

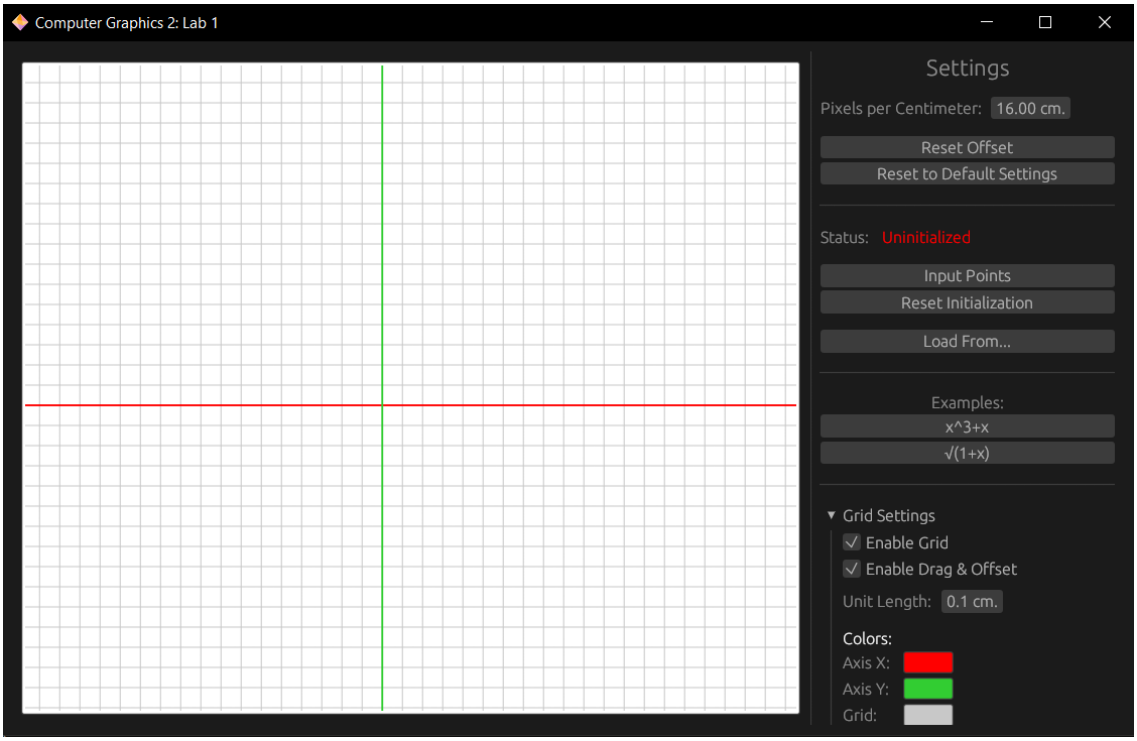
x	-0.900	-0.745	-0.591	-0.436	-0.282	-0.127	0.027	0.182	0.336	0.491	0.645	0.800
y	-1.629	-1.160	-0.797	-0.519	-0.304	-0.129	0.027	0.188	0.374	0.609	0.914	1.312

$$\sqrt{1 + x}$$

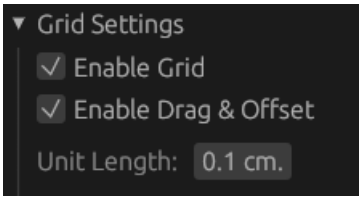
x	-0.900	-0.745	-0.591	-0.436	-0.282	-0.127	0.027	0.182	0.336	0.491	0.645	0.800
y	0.316	0.505	0.640	0.751	0.847	0.934	1.014	1.087	1.156	1.221	1.283	1.342

Олександр – 9 букв, A = -0.9
Ковальов – 8 букв, B = 0.8
Олексійович – 11 букв, C = 11

Хід роботи.
Вигляд інтерфейсу:

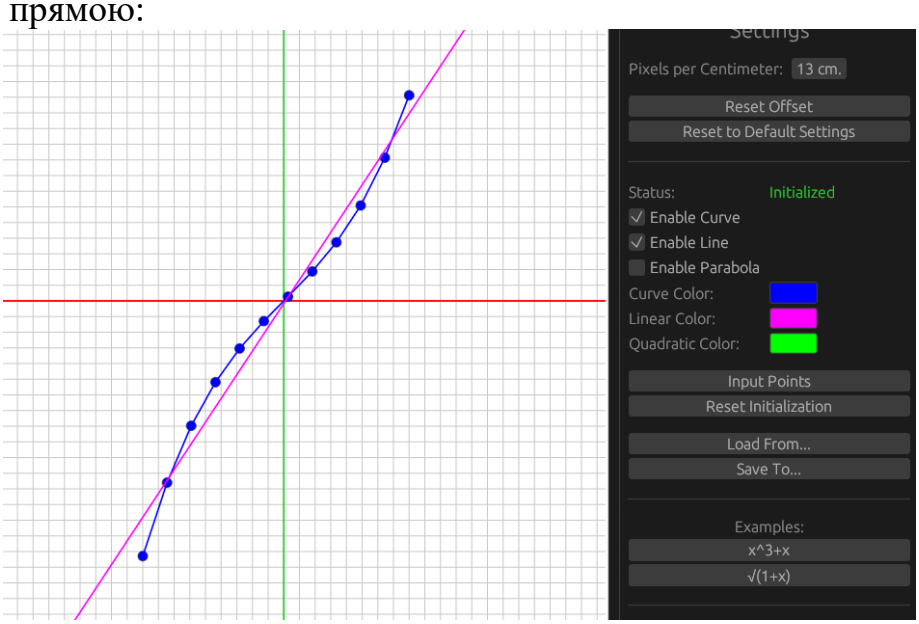


Одиничний відрізок (одна клітинка) налаштовується в меню «Grid» Settings:

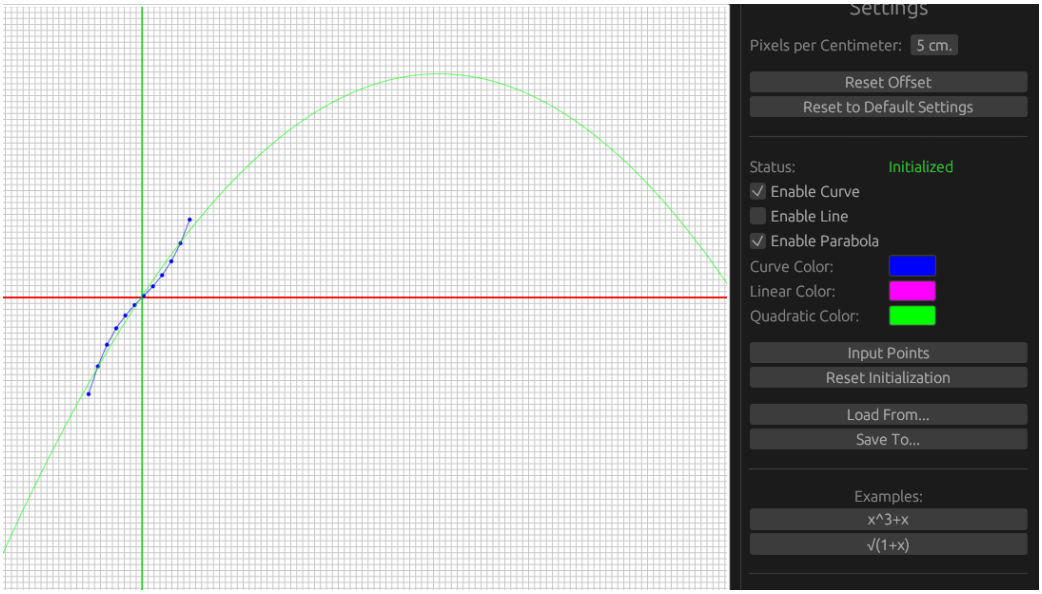


Результати:

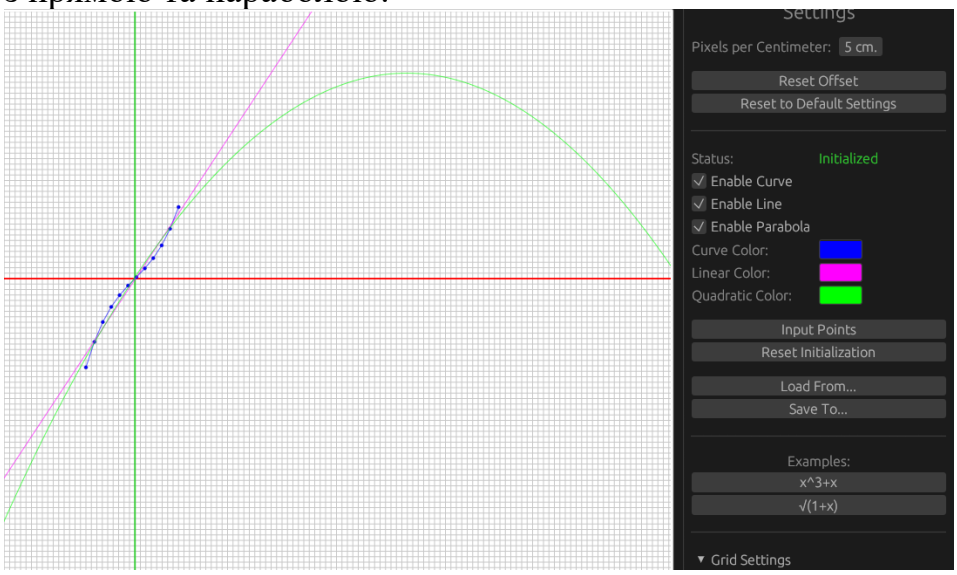
Функція 1 з прямою:



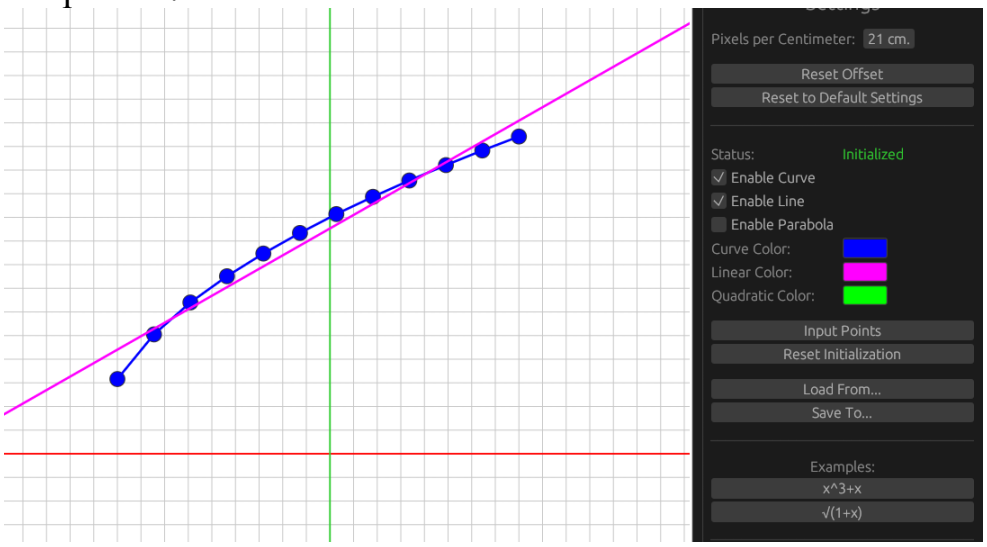
Функція 1 з параболою:



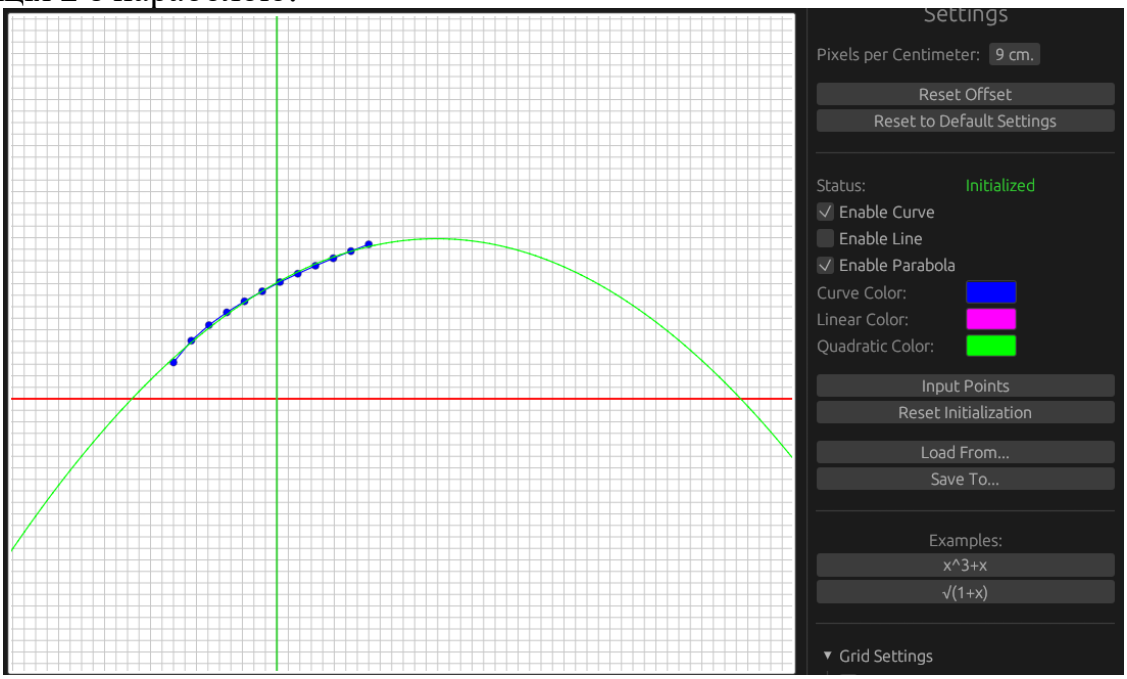
Функція 1 з прямою та параболою:



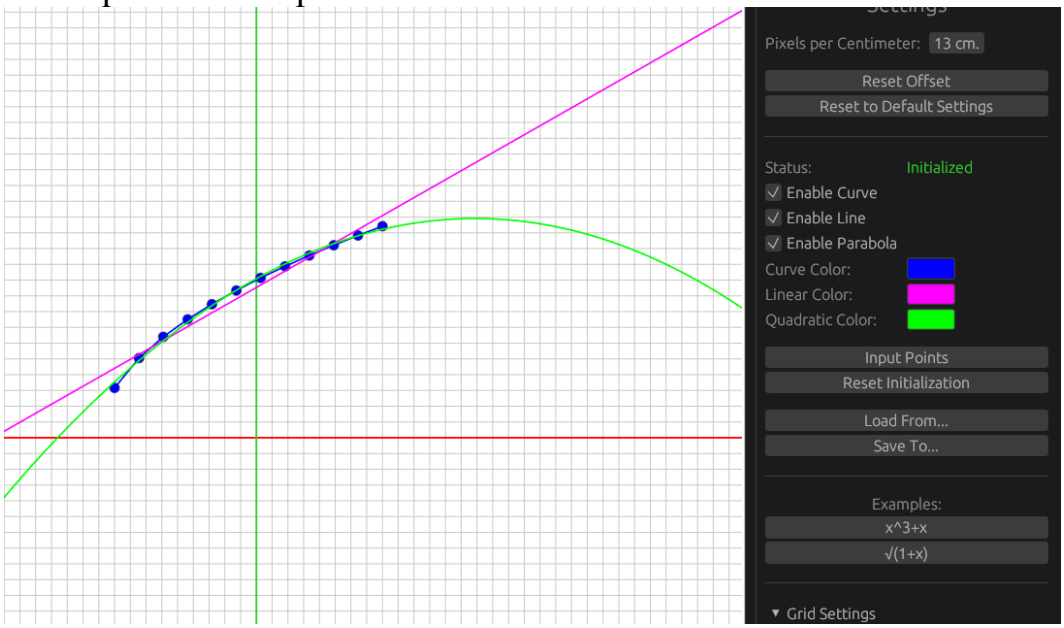
Функція 2 з прямою:



Функція 2 з параболою:



Функція 2 з прямою та параболою:



Додаток. Програмний код:

approximation::math.rs

```
use crate::geometry::point2d::Point2D;
use nalgebra::{Matrix2, Matrix3, Vector2, Vector3};
use thiserror::Error;

pub fn linear(points: &[Point2D]) -> Result<(f32, f32), MathError> {
    let n = points.len() as f32;
    let sum_xs = points.iter().map(|p| p.x).sum::<f32>();
    let sum_xs_quadratic = points.iter().map(|p| p.x.powi(2)).sum::<f32>();

    let a = Matrix2::new(n, sum_xs, sum_xs, sum_xs_quadratic);

    let sum_ys = points.iter().map(|p| p.y).sum::<f32>();
    let sum_xys = points.iter().map(|p| p.x * p.y).sum::<f32>();
    let b = Vector2::new(sum_ys, sum_xys);

    match a.lu().solve(&b) {
        Some(matrix) => Ok((matrix[0], matrix[1])),
        None => Err(MathError::CannotSolveMatrix),
    }
}

pub fn quadratic(points: &[Point2D]) -> Result<(f32, f32, f32), MathError> {
    let n = points.len() as f32;
    let sum_xs = points.iter().map(|p| p.x).sum::<f32>();
    let sum_xs_quadratic = points.iter().map(|p| p.x.powi(2)).sum::<f32>();
    let sum_xs_cubic = points.iter().map(|p| p.x.powi(3)).sum::<f32>();
    let sum_xs_quartic = points.iter().map(|p| p.x.powi(4)).sum::<f32>();

    let a = Matrix3::new(
        n,
        sum_xs,
        sum_xs_quadratic,
        sum_xs,
        sum_xs_quadratic,
        sum_xs_cubic,
        sum_xs_quadratic,
        sum_xs_cubic,
        sum_xs_quartic,
    );

    let sum_ys = points.iter().map(|p| p.y).sum::<f32>();
    let sum_xys = points.iter().map(|p| p.x * p.y).sum::<f32>();
    let sum_xs_quadratic_ys = points.iter().map(|p| p.x.powi(2) * p.y).sum::<f32>();
```

```
    let b = Vector3::new(sum_ys, sum_xys, sum_xs_quadratic_ys);

    match a.lu().solve(&b) {
        Some(matrix) => Ok((matrix[0], matrix[1], matrix[2])),
        None => Err(MathError::CannotSolveMatrix),
    }
}

#[derive(Error, Debug)]
pub enum MathError {
    #[error("Cannot solve linear equations.")]
    CannotSolveMatrix,
}
```

approximation::state.rs

```
use crate::approximation;
use crate::approximation::math::MathError;
use crate::geometry::point2d::Point2D;

#[derive(Default)]
pub struct ApproximationState {
    pub points_view: Vec<Point2D>,

    is_initialized: bool,
    points: Vec<Point2D>,

    linear_coefficients: (f32, f32),
    quadratic_coefficients: (f32, f32, f32),
}

impl ApproximationState {
    pub fn initialize(&mut self) -> Result<(), MathError> {
        self.points.clear();
        for point in &self.points_view {
            self.points.push(*point);
        }

        self.linear_coefficients = approximation::math::linear(&self.points)?;
        self.quadratic_coefficients = approximation::math::quadratic(&self.points)?;

        self.is_initialized = true;

        Ok(())
    }
```

```
    pub fn is_initialized(&self) -> bool {
        self.is_initialized
    }

    pub fn points(&self) -> &Vec<Point2D> {
        self.points.as_ref()
    }

    pub fn linear_coefficients(&self) -> (f32, f32) {
        self.linear_coefficients
    }

    pub fn quadratic_coefficients(&self) -> (f32, f32, f32) {
        self.quadratic_coefficients
    }
}
```

approximation::io.rs

```
use crate::approximation::math::MathError;
use crate::approximation::state::ApproximationState;
use crate::geometry::point2d::Point2D;
use std::fs;
use std::path::PathBuf;
use thiserror::Error;

pub fn load_with_file_pick(state: &mut ApproximationState) -> Result<(), FileError>
{
    if let Some(path) = rfd::FileDialog::new().pick_file() {
        load_from_path(state, path)?
    }

    Ok(())
}

pub fn load_from_path(
    state: &mut ApproximationState, path: PathBuf,
) -> Result<(), FileError> {
    let text = fs::read_to_string(&path)?;

    let deserialized: Vec<Point2D> = serde_json::from_str(&text)?;

    state.points_view = deserialized;
    state.initialize()?;

    Ok(())
```

```
    }

    pub fn save_with_file_pick(state: &mut ApproximationState) -> Result<(), FileError>
{
        let filter = FileFilter::json();

        if let Some(path) = rfd::FileDialog::new()
            .add_filter(filter.name, &filter.file_extensions)
            .save_file()
        {
            save_to_path(state, path)?;
        }

        Ok(())
    }

    pub fn save_to_path(
        state: &mut ApproximationState, path: PathBuf,
    ) -> Result<(), FileError> {
        let serialized = serde_json::to_string(&state.points_view)?;
        fs::write(path, serialized).map_err(FileError::Io)
    }

    #[derive(Error, Debug)]
    pub enum FileError {
        #[error("Input/output error.")]
        Io(#[from] std::io::Error),

        #[error("Serialization error.")]
        Json(#[from] serde_json::Error),

        #[error("Math error.")]
        Initialization(#[from] MathError),
    }

    #[derive(Default)]
    pub struct FileFilter {
        pub name: String,
        pub file_extensions: Vec<&'static str>,
    }

    impl FileFilter {
        pub fn json() -> Self {
            FileFilter {
                name: String::from("JSON"),
```

```
            file_extensions: vec!["json"],
        }
    }
}
```

approximation::graphics.rs

```
use crate::geometry::line2d::Line2D;
use crate::geometry::point2d::Point2D;
use egui::Stroke;

pub fn linear_line(coefficients: (f32, f32), stroke: Stroke) -> Line2D {
    let (a, b) = coefficients;

    let x_start = -100.0;
    let x_end = 100.0;

    let start = Point2D::new(x_start, a + b * x_start);
    let end = Point2D::new(x_end, a + b * x_end);

    Line2D::new(start, end, stroke)
}

pub fn parabola_lines(coefficients: (f32, f32, f32), stroke: Stroke) -> Vec<Line2D> {
    let (a, b, c) = coefficients;

    let mut points: Vec<Point2D> = Vec::new();

    let x_start = -100.0;
    let x_end = 100.0;

    let mut x = x_start;
    while x <= x_end + 0.0001 {
        let point = Point2D::new(x, a + b * x + c * x.powi(2));
        points.push(point);

        x += 0.05;
    }

    points
        .windows(2)
        .map(|p| Line2D::new(p[0], p[1], stroke))
        .collect()
}
```