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

Лабораторна робота №2

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 інтерполяційну криву за допомогою кусково-лінійної інтерполяції та поліному Лагранжа за такими даними:

$$5x^5 - 3x^3 + x$$

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	-1.665	-0.652	-0.332	-0.266	-0.224	-0.121	0.027	0.165	0.244	0.279	0.398	0.902

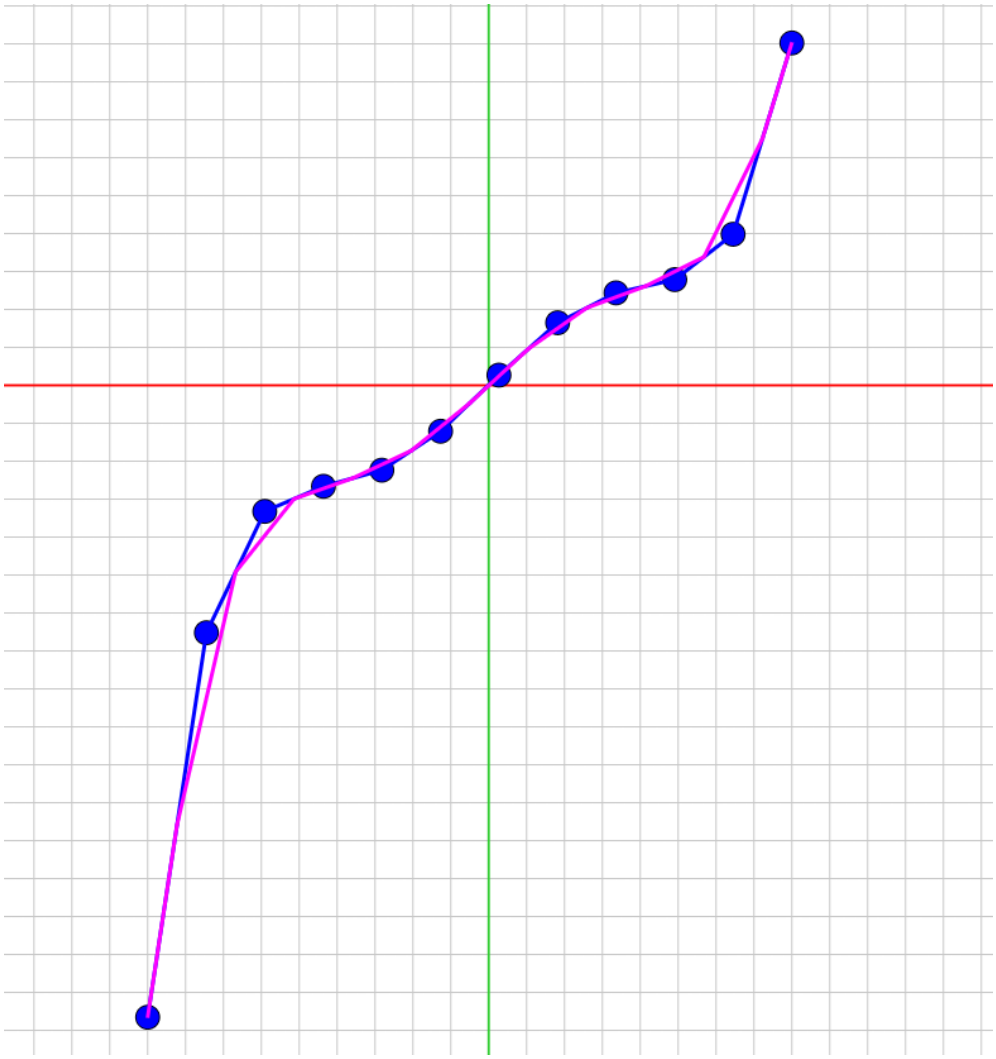
$$3 * cos(2x) - 2 * sin(5x)$$

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	-2.637	-0.860	1.508	3.570	4.510	4.090	2.726	1.224	0.360	0.398	0.998	1.426

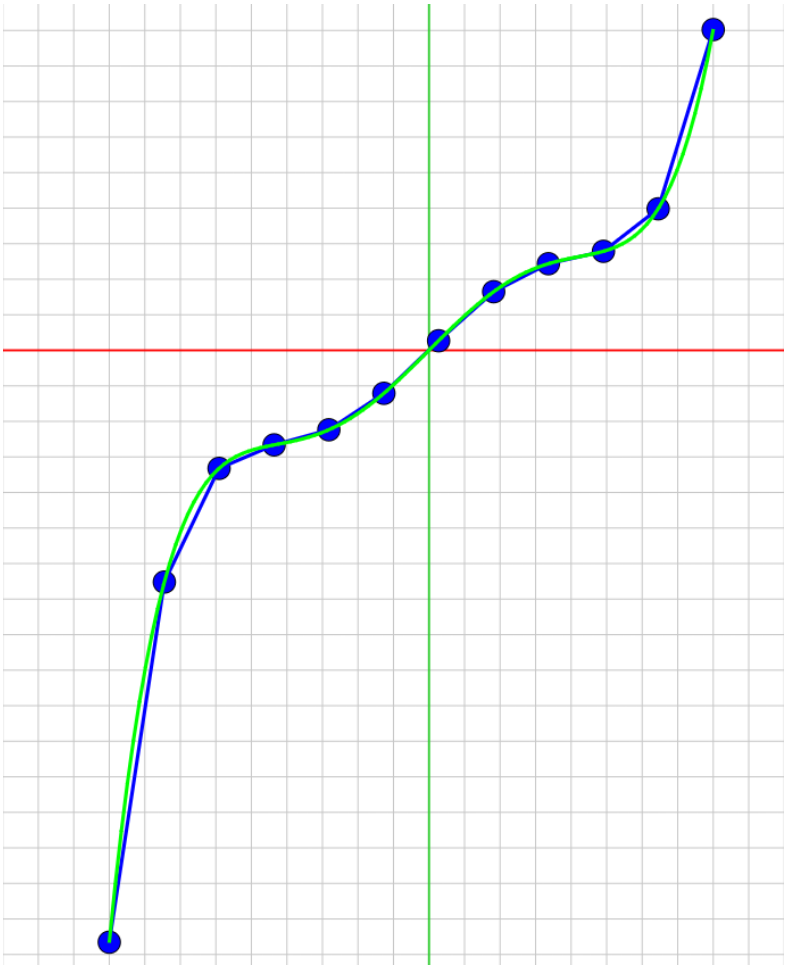
Олександр – 9 букв, A = -0.9
Ковальов – 8 букв, B = 0.8
Олексійович – 11 букв, C = 11

Результати:

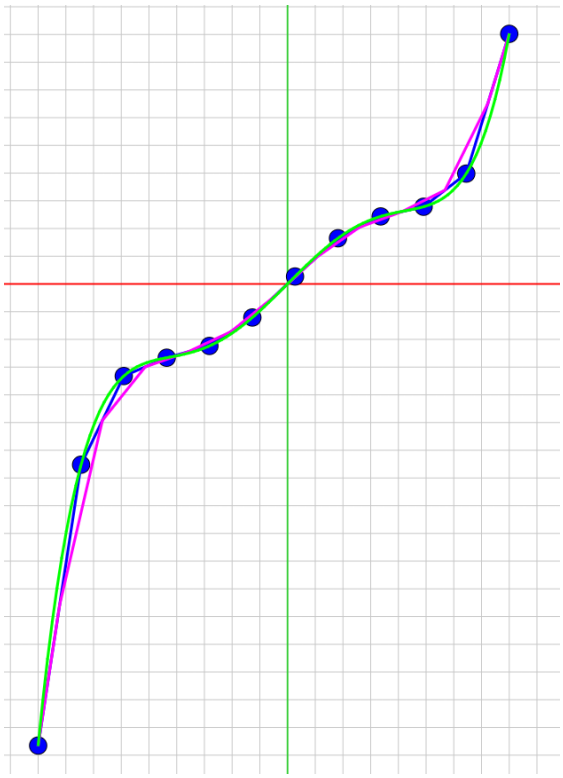
Функція 1 з ламаною (кусково-лінійна інтерполяція), одна поділка – 0.1 см.:



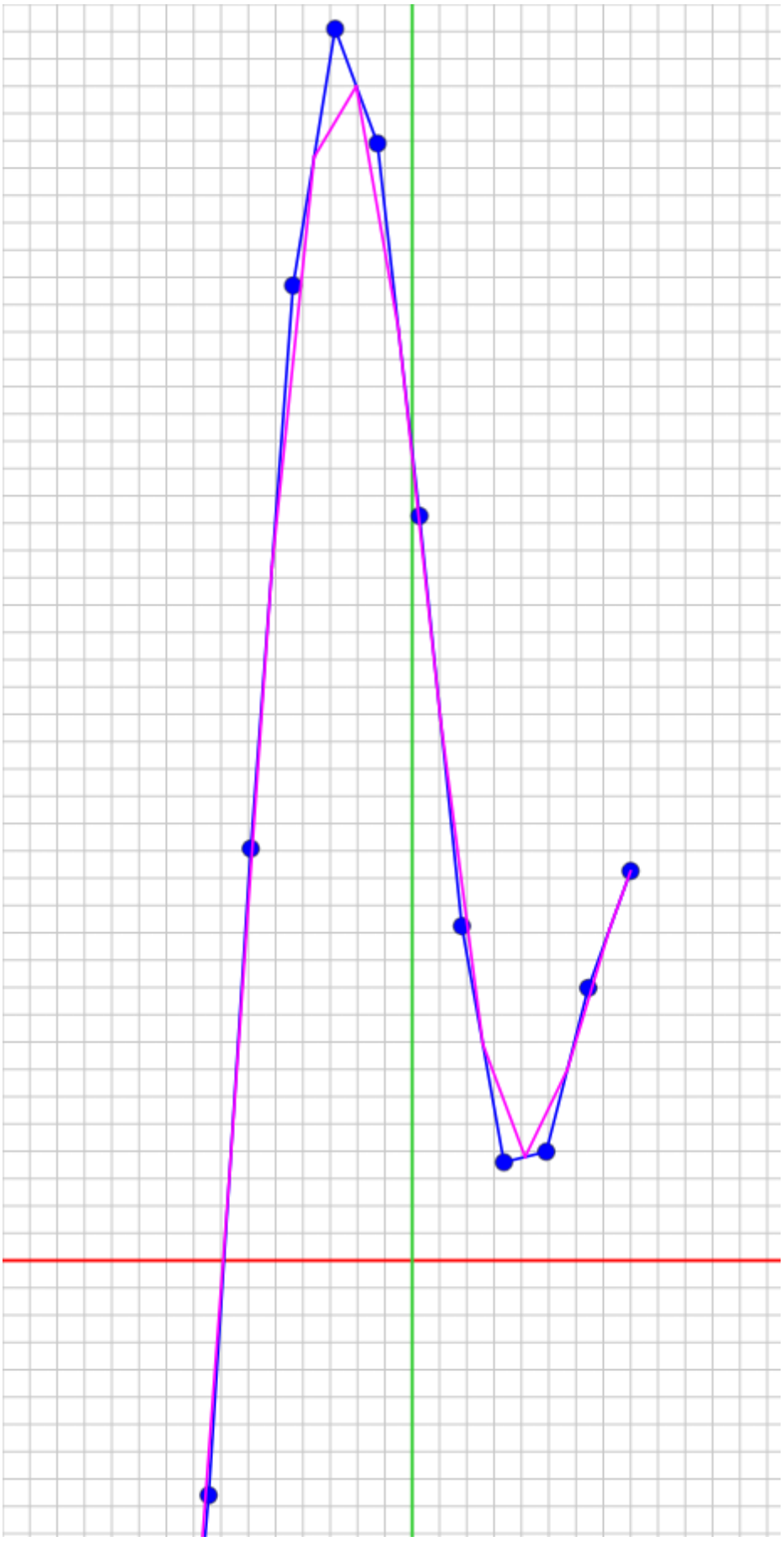
Функція 1 з ламаною побудованою за допомогою поліному Лагранжа:



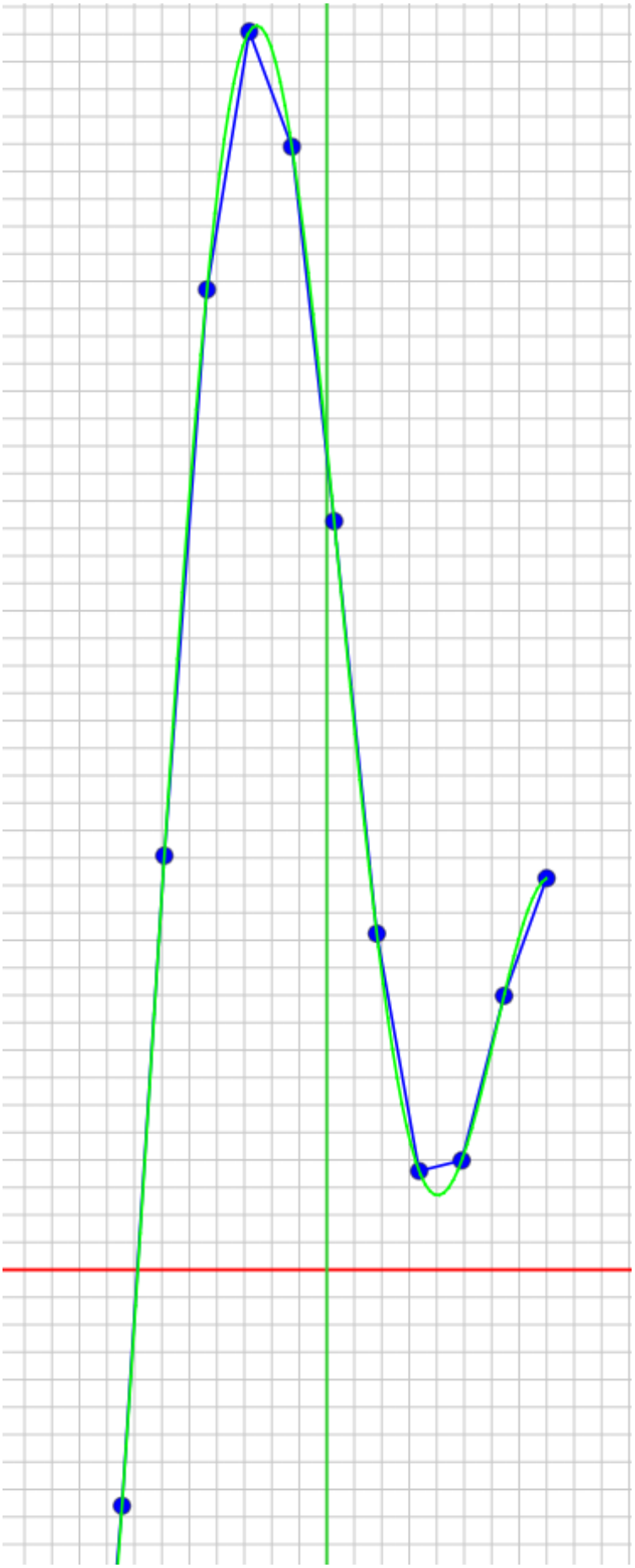
Функція 1 з ламаними побудованими за допомогою кусково-лінійної інтерполяції та поліному Лагранжа:



Функція 2 з ламаною (кусково-лінійна інтерполяція):



Функція 2 з ламаною побудованою за допомогою поліному Лагранжа:



Функція 2 з ламаними побудованими за допомогою кусково-лінійної інтерполяції та поліному Лагранжа:

Додаток. Програмний код:

interpolation::math.rs

```
use crate::geometry::point2d::Point2D;

pub fn linear(points: &[Point2D]) -> Vec<Point2D> {
    let mut result: Vec<Point2D> = Vec::with_capacity(points.len() + 1);

    if let Some(first_point) = points.first() {
        result.push(*first_point);
    }

    points.windows(2).for_each(|pair| {
        let x = (pair[1].x + pair[0].x) / 2.0;
        let y = pair[0].y
            + (x - pair[0].x) / (pair[1].x - pair[0].x) * (pair[1].y - pair[0].y);
        result.push(Point2D::new(x, y));
    });

    if let Some(last_point) = points.last() {
        result.push(*last_point);
    }

    result
}

pub fn lagrange(points: &[Point2D]) -> Vec<Point2D> {
    let mut result = Vec::new();
    let n = points.len();

    if n < 2 {
        return result;
    }

    let min_x = points.first().unwrap().x;
    let max_x = points.last().unwrap().x;

    let num_points = 100;
    let step = (max_x - min_x) / num_points as f32;

    for i in 0..num_points {
        let x = min_x + step * i as f32;
        let mut y = 0.0;

        for i in 0..n {
            let (x_i, y_i) = (points[i].x, points[i].y);
```

```
            let mut L = 1.0;
            for j in 0..n {
                if i != j {
                    let (x_j, _) = (points[j].x, points[j].y);
                    L *= (x - x_j) / (x_i - x_j);
                }
            }

            y += y_i * L;
        }

        result.push(Point2D::new(x, y));
    }

    if let Some(last_point) = points.last() {
        result.push(*last_point);
    }

    result
}
```

interpolation::state.rs

```
use crate::geometry::point2d::Point2D;
use crate::interpolation;

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

    is_initialized: bool,
    points: Vec<Point2D>,

    linear_points: Vec<Point2D>,
    lagrange_points: Vec<Point2D>,
}

impl InterpolationContext {
    pub fn initialize(&mut self) {
        self.points.clear();
        for point in &self.points_view {
            self.points.push(*point);
        }

        self.linear_points = interpolation::math::linear(&self.points);
```

```
        self.lagrange_points = interpolation::math::lagrange(&self.points);

        self.is_initialized = true;
    }

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

    pub fn points(&self) -> &[Point2D] {
        self.points.as_ref()
    }

    pub fn linear_points(&self) -> &[Point2D] {
        &self.linear_points
    }

    pub fn lagrange_points(&self) -> &[Point2D] {
        &self.lagrange_points
    }
}
```

interpolation::io.rs

```
use crate::geometry::point2d::Point2D;
use crate::interpolation::state::InterpolationContext;
use std::fs;
use std::path::PathBuf;
use thiserror::Error;

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

    Ok(())
}

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

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

    ctx.points_view = deserialized;
```

```
        ctx.initialize();

        Ok(())
    }

    pub fn save_with_file_pick(ctx: &mut InterpolationContext) -> 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(ctx, path)?;
        }

        Ok(())
    }

    pub fn save_to_path(
        ctx: &mut InterpolationContext, path: PathBuf,
    ) -> Result<(), FileError> {
        let serialized = serde_json::to_string(&ctx.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),
    }

    #[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"],
        }
    }
}
```