Matte Oblig 1

Mathias Mohn Mørch

Januar 2023

1 Innledning

I de følgende oppgave hvor vi skal nummerisk regne ut en polynom med minste kvadrats metode, og regne ut et polynom men interpolasjon av 3 grad.
Vi kommer også til å bruke openGL til å visualere polynomene vi får.

2 Oppgave 1

I denne oppgaven brukte jeg standardbasisen E.

1

Bestemte 8 punkter for grafen.

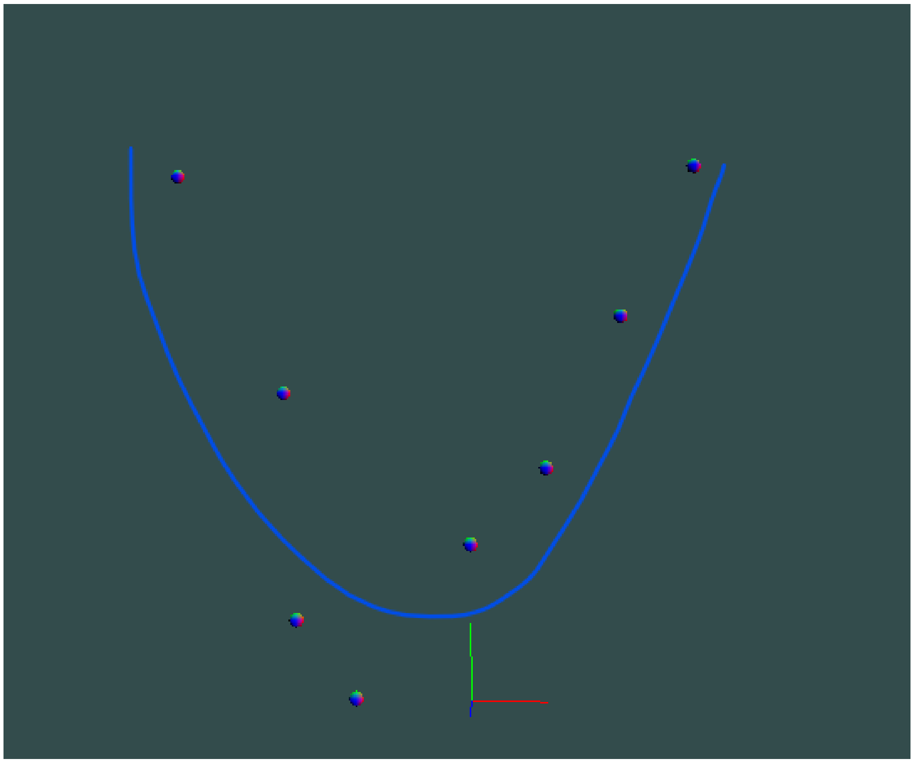
Listing 1: C++ example

```
std::vector<glm::vec3> MathComp2Handler::GenerateRandomPoints() const {
    std::vector<glm::vec3> ps{};

    ps.push_back(glm::vec3(-4, 7, 0));
    ps.push_back(glm::vec3(-2.5, 4, 0));
    ps.push_back(glm::vec3(-2.3, 1, 0));
    ps.push_back(glm::vec3(-1.5, 0, 0));
    ps.push_back(glm::vec3(0, 2, 0));
    ps.push_back(glm::vec3(1, 3, 0));
    ps.push_back(glm::vec3(2, 5, 0));
    ps.push_back(glm::vec3(3, 7, 0));

    return ps;
}
```

Skisee av punktene.



2

Brukte mattebibloteket Eigen til å skaffe of regne ut matrisene og løse ligningssystemet

$$A \cdot x = b \qquad\qquad x = A^{-1} \cdot b$$

Listing 2: Funksjon som genererer punktene

```
void MathComp2Handler::HandleTask1() {
    points = GenerateRandomPoints();

    Eigen::MatrixXd A = Eigen::MatrixXd(points.size(), 3);
    A.setZero();
    for (int i = 0; i < points.size(); ++i) {
        A(i, 0) = pow(points[i].x, 2.f);
        A(i, 1) = points[i].x;
        A(i, 2) = 1.f;
    }
    std::cout << A << std::endl;
    Eigen::MatrixXd y = Eigen::MatrixXd(points.size(), 1);
    for (int i = 0; i < points.size(); ++i) {
        y(i, 0) = points[i].y;
    }

    std::cout << y << std::endl;

    Eigen::MatrixXd B = Eigen::Transpose<Eigen::MatrixXd>(A) * A;
```

```
    std::cout << B << std::endl;
    Eigen::MatrixXd c = Eigen::Transpose<Eigen::MatrixXd>(A) * y;

    Eigen::MatrixXd x = Eigen::Inverse<Eigen::MatrixXd>(B)*c;
    std::cout << "result" << std::endl << x << std::endl;

    // writing mVerices
    mVertices.clear();
    float min = -4.f;
    float max = 3.f;
    int divisions = 10;
    float delta = (max - min)/divisions;

    for (float i = min; i <= max+0.001f; i+=delta) {
        float y =
        x(0,0)*i*i +
        x(1,0)*i +
        x(2,0);
        Vertex v = Vertex(glm::vec3(i, y,0), glm::vec3(0,0,0));
        mVertices.push_back(v);
    }
}
```

Dette ga oss følgene kolonne vektor

$$\begin{bmatrix} 0.4975 \\ 0.6513 \\ 1.2513 \end{bmatrix}$$

Som gir oss følgende polynom

$$f(x) = 0.4975 \cdot x^2 + 0.6513 \cdot x + 1.2513$$

3

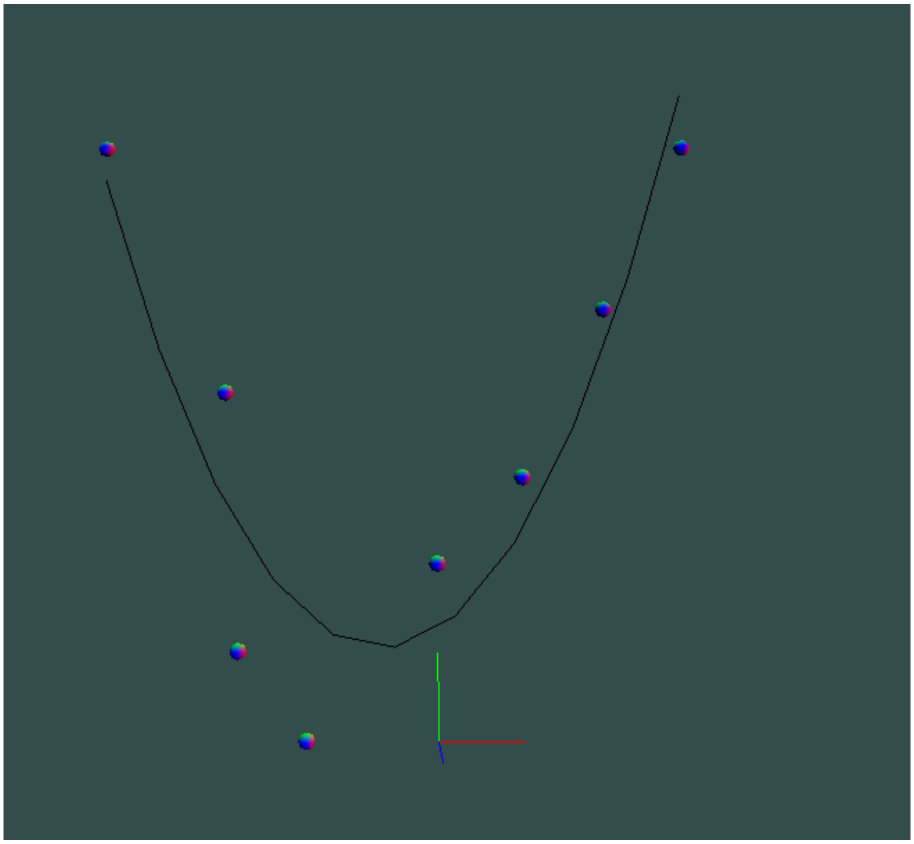
Skrev dette til fil, og leste inn i en Graph2D klasse
(i main().)

Listing 3: Funksjon som genererer punktene

```
KT::MathComp2Handler* handler = new KT::MathComp2Handler();
handler->HandleTask1();
handler->ToFile("math2_2");
mMap.insert(MapPair("math2", handler));

KT::Graph2D* graph_2 = new KT::Graph2D(KT::FileHandler::VertexFromFile("math2_2"));
mMap.insert(MapPair("graph_2", graph_2));
```

(ToFile og FromFile har blitt gjort i oblig1, velger derfor å ikke vise her.) Visualiseringen fra programmet



3 Oppgave 2

I denne oppgaven brukte jeg standardbasisen E.

1

Skrev inn punkter, fant A matrisen, lagde y matrise (y verdiene til punktene). Regnet derreter ut løsningen av

$$A \cdot x = b$$
$$x = A^{-1} \cdot b$$

og skrev til fil.

Listing 4: Oppgave 2 numerisk utregning

```
void MathComp2Handler::HandleTask2() {
    std::vector<glm::vec3> ps{};
    ps.push_back(glm::vec3(0,0,0));
    ps.push_back(glm::vec3(1,2,0));
    ps.push_back(glm::vec3(2,5,0));
    ps.push_back(glm::vec3(3,6,0));
    points = ps;
```

```
    Eigen::MatrixXd A = Eigen::MatrixXd(4, 4);
    for (int i = 0; i < ps.size(); ++i) {
        A(i,0) = ps[i].x * ps[i].x * ps[i].x;
        A(i,1) = ps[i].x * ps[i].x;
        A(i,2) = ps[i].x;
        A(i,3) = 1.0;
    }

    std::cout << A <<  std::endl;

    Eigen::Vector4d y{};
    for (int i = 0; i < ps.size(); ++i) {
        y(i) = ps[i].y;
    }

    std::cout << y << std::endl;
    Eigen::Vector4d x = A.inverse() * y;
    std::cout << x << std::endl;

    mVertices.clear();
    float min = 0.f;
    float max = 3.f;
    int divisions = 10;
    float delta = (max - min)/divisions;

    for (float i = 0; i <= max+0.001f; i+=delta) {
        float y =
            x(0)*i*i*i +
            x(1)*i*i +
            x(2)*i +
            x(3);
        Vertex v = Vertex(glm::vec3(i, y,0), glm::vec3(0,0,0));
        mVertices.push_back(v);
    }

}
```

Kjørte programmet og dette ga følgende kolonne vektor

$$\begin{bmatrix} -0.5 \\ 2 \\ 0.5 \\ 0 \end{bmatrix}$$

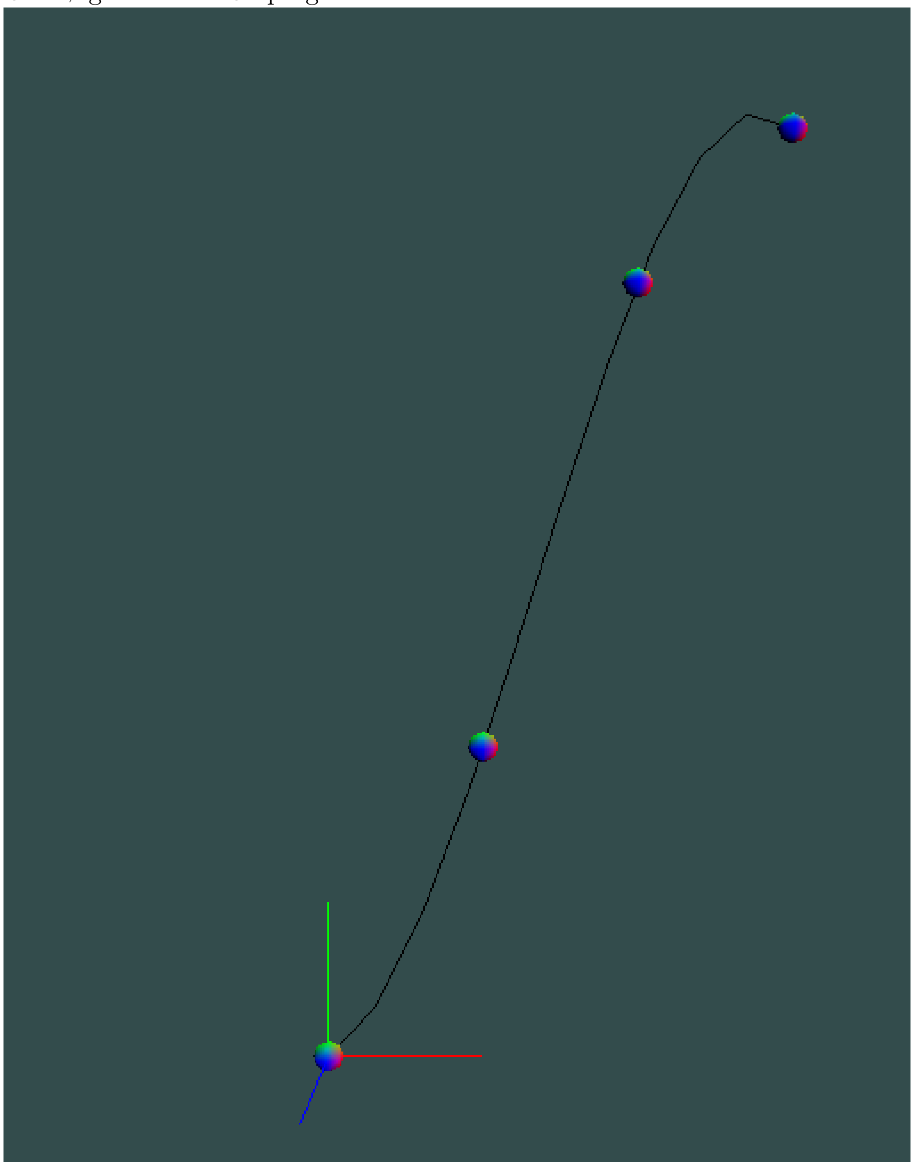
Som ga oss polynomet

$$f(x) = -0.5 \cdot x^3 + 2\ cdot x^2 + 0.5 \cdot x + 0$$

2

Brukte punktene regner ut fra del 1 av oppgave 2 og leste fildata inn i en Graph2D klasse

Gav følgene bilde i 3d program.



4 Diskusjon

Har lært mer om åssen man finner løsningen av matrise likningsystemer og hvordan det kan gjøres nummerisk.
Har også lært mer om komposisjon, dette er i hovedgrad grunnet at mange klasser trenger å skrive KT::Vertex data til fil. Da lærte jeg hvor nytting det kan være å lage en FileHandler klasse slik at ikke alle klasser trenger å arve funksjonaliteten, og samt hvordan man bruker dette via komposisjon.
Lærte også mer om hvordan interpolasjon fungerer. Grunnet at dette gir mer repitisjon av konseptene vi har lært i timen samt at når man programmerer det må man ha ”tunga rett i munnen”

5 Lenker

Github lenke : https://github.com/Skyress-s/BaseOpenGL/tree/Math3-Compulsory-1