โครงงาน Project Website

เรื่อง **Numerical Method (Root of Equation)**

จัดทำโดย

นายณัทพล หาญจันทร์ 6704062612090

เสนอ

ผู้ช่วยศาสตราจารย์สถิตย์ ประสมพันธ์

โครงงานนี้เป็นส่วนหนึ่งของรายวิชา Object Oriented Programming

ภาควิชาวิทยาการคอมพิวเตอร์และสารสนเทศ คณะวิทยาศาสตร์ประยุกต์

มหาวิทยาลัยเทคโนโลยีพระจอมเหล้าพระนครเหนือ

ภาคเรียนที่ 1 ปีการศึกษา 2568

สารบัญ

บทที่ 1
บทนำ

1.1 เกี่ยวกับโครงงาน

ชื่อโครงงาน : Numerical Method (Root of Equation)
นำเสนอโดย : นายณัทพล หาญจันทร์
อาจารย์ผู้สอน : ผู้ช่วยศาสตราจารย์สถิตย์ ประสมพันธ์
Source Code : https://github.com/nanonorth/nanonorth.github.io/tree/main

1.2 ที่มาและความสัมคัญ

โครงงานนี้จัดขึ้นเพื่อวัดผลการเรียนในรายวิชา Object Oriented Programming (OOP) โดยมีจุดประสงค์เพื่อให้นักศึกษาได้นำความรู้ที่ได้จากบทเรียนมาประยุกต์ใช้จริง ผ่านการสร้างผลงานในรูปแบบของเว็บไซต์ ซึ่งช่วยเสริมความเข้าใจแนวคิดของการเขียนโปรแกรมเชิงวัตถุ

1.3 ประเภทของโครงงาน

โปรแกรมเว็บแอปพลิเคชัน Frontend

1.4 ประโยชน์

1) เพื่อให้สามารถคำรวณปัญหาทาง Numerical ได้สะดวก
2) เพื่อนำความรู้จากวิชา Numerical Method และ Object Oriented Programming (OOP) มาประยุกต์ใช้
3) เพื่อนำแนวคิดการเขียนโปรแกรมแบบเชิงวัตถุมาประยุกต์ใช้

1.5 ขอบเขตของโครงการ

1) ความต้องการของระบบ (Functional Requirements)

- สามารถคำนวณปัญหาทาง Numerical ดังนี้ได้
  - Root of Equation
    - Graphical Method
    - Bisection Method
    - False Position Method
    - One Point Iteration Method
    - Taylor Series Method
    - Newton Raphson Method
    - Secant Method
- สามารถแสดงผลลัพธ์ออกเป็นกราฟแบบโต้ตอบ

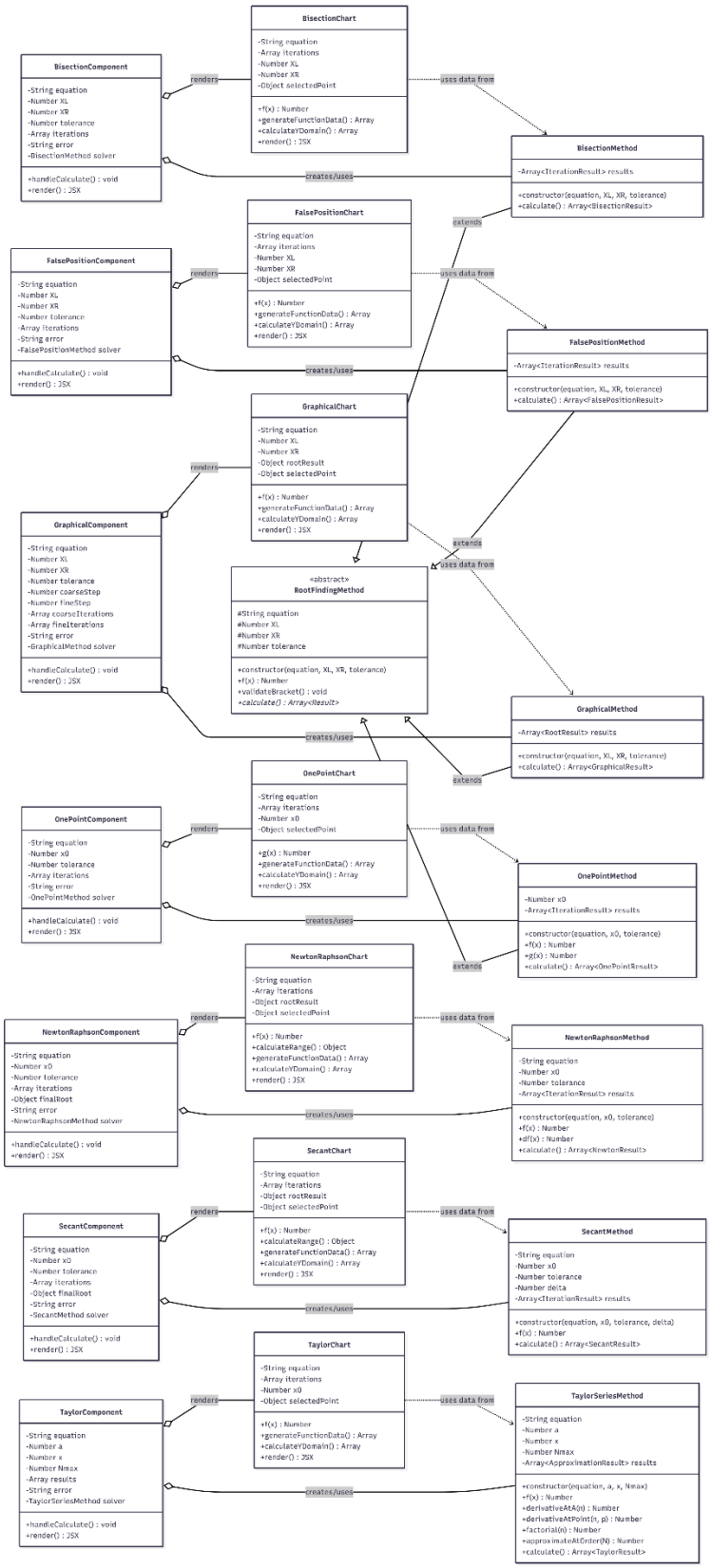
2) ตารางเวลาการดำเนินโครงการ

ลำดับ	รายการ	5 – 10 (ตุลาคม)	11 – 20 (ตุลาคม)	20 – 25 (ตุลาคม)	25 – 31 (ตุลาคม)
1	ศึกษาข้อมูล				
2	Root of Equation				
3	Frontend				
4	จัดทำเอกสารรายงาน				
5	ตรวจสอบและแก้ไขข้อผิดพลาด				

บทที่ 2

ส่วนการพัฒนา

2.1 แผนภาพ Class Diagram



แบ่งเป็น 3 ส่วนหลัก คือ

Component

ส่วนติดต่อผู้ใช้ (React UI)

- Graphical.jsx
- Bisection.jsx
- False_Position.jsx
- OnePoint.jsx
- Taylor.jsx
- Newton_Raphson.jsx
- Secant.jsx

Chart

ส่วนแสดงผลกราฟ

- Graphical_Chart.jsx
- Bisection_Chart.jsx
- False_Position_Chart.jsx
- OnePoint_Chart.jsx
- Taylor_Chart.jsx
- Newton_Raphson_Chart.jsx
- Secant_Chart.jsx

Method

ส่วนคำนวณหลักทาง Numerical

- RootFindingMethod
  - เป็นคลาสแม่ของทุกวิธี ประกาศตัวแปรหลัก (equation, XL, XR, tolerance) และเมธอดพื้นฐาน f(x) เพื่อใช้คำนวณค่าของสมการ
- BisectionMethod
  - ใช้หลักการแบ่งครึ่งช่วง (Midpoint) เพื่อหาค่าราก โดยสืบทอดจาก RootFindingMethod
  - มีเมธอด calculate() สำหรับคำนวณรอบซ้ำ (iteration) และส่งผลลัพธ์เป็น array

- FalsePositionMethod
    - ใช้หลักการคล้าย Bisection แต่ปรับค่าช่วงตามสมการเส้นตรงแทนการแบ่งครึ่ง
- OnePointMethod
    - ใช้สมการรูปแบบ x = g(x) เพื่อหาค่ารากโดยการวนซ้ำ
- TaylorSeriesMethod
    - ใช้การขยายอนุกรมเทย์เลอร์ในการประมาณค่าฟังก์ชัน
    - ใช้การหาค่าอนุพันธ์หลายระดับ และบันทึกผลลัพธ์ต่อเนื่อง
- NewtonRaphsonMethod
    - ใช้สูตรอนุพันธ์ในการปรับค่า x (Newton Step) เพื่อหาค่าราก
    - มีความซับซ้อนขึ้น ใช้แนวคิดการประมาณอนุพันธ์
- SecantMethod
    - ใช้แนวคิดจาก Newton แต่ไม่ต้องใช้อนุพันธ์จริง โดยคำนวณจากสองจุดก่อนหน้า

2.2 รูปแบบการพัฒนา

ภาษา: Java React

Framework: Vite

โปรแกรม: Visual Studio Code, GitHup

2.3) แนวคิดการเขียนโปรแกรมเชิงวัตถุ

- Constructor

```
export default class Root_Finding_Method {
  constructor(equation, XL, XR, tolerance) {
    this.equation = equation;
    this.XL = XL;
    this.XR = XR;
    this.tolerance = tolerance;
  }
}
```

คลาสนี้เป็น “แม่แบบ” ของทุกวิธีหาค่าราก (Root-Finding Method) ใช้ Encapsulation ซ่อนค่าพารามิเตอร์พื้นฐานไว้ในอ็อบเจ็กต์เป็นฐานสำหรับ Inheritance ให้คลาสลูกทุกตัวสืบทอด

- Encapsulation

```
export default class SecantMethod {
  constructor(equation, x0, tolerance, delta = 0.01) {
    this.equation = equation;
    this.x0 = parseFloat(x0);
    this.tolerance = parseFloat(tolerance);
    this.delta = parseFloat(delta);
    this.results = [];
  }

  f(x) {
    const eq = this.equation.replace(/\^/g, "**");
    return new Function("x", `return ${eq}`)(x);
  }
}
```

ตัวอย่างจาก จาก Secant_Method.js ตัวแปร equation, x0, tolerance, delta, results ถูกเก็บไว้ภายในอ็อบเจกต์ และเรียกใช้ผ่านเมธอด f(x) เท่านั้น ภายนอกไม่สามารถเข้าถึงโดยตรง

```
const [equation, setEquation] = useState("x**2 - 7");
const [x0, setX0] = useState(1);
const [tolerance, setTolerance] = useState(0.000001);
```

ตัวอย่างจาก React Component OnePoint.jsx ค่าทั้งหมดถูก “ห่อหุ้ม” ใน useState ของ component ไม่ให้ส่วนอื่นของแอปเข้าถึงโดยตรง

- Abstraction

```
calculate() {
  this.validateBracket();
  let xl = this.XL, xr = this.XR;
  let xm, error;

  do {
    xm = (xl + xr) / 2;
    if (this.f(xl) * this.f(xm) < 0) xr = xm;
    else xl = xm;
    error = Math.abs((xr - xl) / xr);
  } while (error > this.tolerance);

  return xm;
}
```

ตัวอย่างจาก Bisection_Method.js ผู้ใช้แค่เรียก calculate() โดยไม่ต้องรู้ว่าในนั้นมีการวนลูปและตรวจค่า f(x) อย่างไร

```
calculate() {
  for (let N = 0; N <= this.Nmax; N++) {
    const approx = this.approximateAtOrder(N);
    this.results.push({ order: N, approx });
  }
  return this.results;
}
```

ตัวอย่างจาก Taylor_Method.js การหาค่าอนุพันธ์หลายระดับถูกซ่อนไว้ใน approximateAtOrder() ผู้ใช้เห็นแค่ผลลัพธ์สุดท้ายของ Taylor Series

- Inheritance

```
import RootFindingMethod from "./Root_Finding_Method";

export default class BisectionMethod extends RootFindingMethod {
  constructor(equation, XL, XR, tolerance) {
    super(equation, XL, XR, tolerance);
    this.results = [];
  }
}
```

ตัวอย่างจากจาก Bisection_Method.js BisectionMethod สืบทอด (extends) จาก RootFindingMethod และเรียกใช้ตัวสร้าง (super) ของคลาสแม่เพื่อรับค่า equation, XL, XR, tolerance

```
export default class FalsePositionMethod extends RootFindingMethod {
  constructor(equation, XL, XR, tolerance) {
    super(equation, XL, XR, tolerance);
    this.results = [];
  }
}
```

ตัวอย่างจาก False_Position_Method.js สืบทอดโครงสร้างเดียวกันจากคลาสแม่ แต่จะมีการคำนวณเฉพาะของตัวเองใน calculate()

- Polymorphism

```
calculate() {
  // ใช้ midpoint
  let xm = (this.XL + this.XR) / 2;
  // ...
}
```

```
calculate() {
 // ใช้เส้นตรงระหว่างสองจุดแทน midpoint
 const fx = this.f(this.x0);
 const fx_delta = this.f(this.x0 + this.delta * this.x0);
 const xNew = this.x0 - (fx * this.delta * this.x0) / (fx_delta - fx);
}
```

ตัวอย่างจาก Bisection_Method.js และ Secant_Method.js ทั้งสองคลาสมีเมธอดชื่อ calculate() เหมือนกันแต่เนื้อหาภายในต่างกันตามสูตรของแต่ละวิธี

- Composition

```
import BisectionMethod from "./Method/Bisection_Method";
import BisectionChart from "./Chart/Bisection_Chart";

function Bisection() {
  const solver = new BisectionMethod(equation, XL, XR, tolerance);
  const results = solver.calculate();
  return <BisectionChart equation={equation} iterations={results} />;
}
```

ตัวอย่างจาก Bisection.jsx Component Bisection ประกอบด้วย BisectionMethod.jsx สำหรับคำนวณ และ BisectionChart สำหรับแสดงกราฟ

```
const solver = new TaylorMethod(equation, a, x, Nmax);
const resultsData = solver.calculate();
<TaylorGraph equation={equation} results={resultsData} />;
```

ตัวอย่างจาก Taylor.jsx ประกอบด้วยทั้งคลาสคำนวณและคอมโพเนนต์กราฟ พื่อแสดงผล

2.4) Algorithm

- Bisection Method

เป็นการแบ่งครึ่งช่วงระหว่างค่า xLx_LxL และ xRx_RxR ที่ทำให้ f(xL)f(x_L)f(xL) และ f(xR)f(x_R)f(xR) มีเครื่องหมายตรงข้ามกัน จากนั้นจะคำนวณค่ากึ่งกลาง xM=(xL+xR)/2x_M = (x_L + x_R)/2xM=(xL+xR)/2 แล้วตรวจสอบว่ารากอยู่ช่วงใด และทำซ้ำจนกว่าความคลาดเคลื่อนจะน้อยกว่า Tolerance

ขั้นตอน (Algorithm):

1. รับสมการ f(x)f(x)f(x), ค่า xLx_LxL, xRx_RxR และ Tolerance
2. ตรวจสอบว่า f(xL)×f(xR)<0f(x_L) \times f(x_R) < 0f(xL)×f(xR)<0 หรือไม่ (เพื่อยืนยันว่ามีรากอยู่ในช่วงนั้น)
3. คำนวณ xM=(xL+xR)/2x_M = (x_L + x_R)/2xM=(xL+xR)/2
4. ถ้า f(xL)×f(xM)<0f(x_L) \times f(x_M) < 0f(xL)×f(xM)<0 ⟶ รากอยู่ด้านซ้าย ⟶ ตั้งค่า xR=xMx_R = x_MxR=xM

   มิฉะนั้น ⟶ รากอยู่ด้านขวา ⟶ ตั้งค่า xL=xMx_L = x_MxL=xM
5. คำนวณค่าความคลาดเคลื่อน

   1. $$error = \frac{|x_{new} - x_M|}{x_{new}}$$
6. ทำซ้ำจนกว่า error $\leq$ Tolerance

```
calculate() {
  this.validateBracket();
  let xl = this.XL, xr = this.XR;
  let xm, error;
  do {
    xm = (xl + xr) / 2;
    if (this.f(xl) * this.f(xm) < 0) xr = xm;
    else xl = xm;
    error = Math.abs((xr - xl) / xr);
  } while (error > this.tolerance);
  return xm;
}
```

- False Position Method

คล้ายกับ Bisection แต่แทนที่จะใช้ midpoint จะใช้เส้นตรงเชื่อมระหว่าง (xL,f(xL))(x_L, f(x_L))(xL,f(xL)) และ (xR,f(xR))(x_R, f(x_R))(xR,f(xR)) เพื่อหาค่ารากประมาณ

$$x_{new} = \frac{x_L f(x_R) - x_R f(x_L)}{f(x_R) - f(x_L)}$$

```
xNew = (this.XL * this.f(this.XR) - this.XR * this.f(this.XL)) /
       (this.f(this.XR) - this.f(this.XL));
```

แทนที่จะหารครึ่งเหมือน Bisection วิธีนี้ใช้สมการเชิงเส้นระหว่างสองจุดเพื่อหาค่ารากที่แม่นยำกว่า

- One-Point Iteration Method

ใช้การแทนรูปสมการให้อยู่ในรูป x=g(x)x = g(x)x=g(x) แล้วทำการวนซ้ำจนกว่าจะได้ค่าที่นิ่ง (convergent)

$$x_{i+1} = g(x_i)$$

```
let xOld = this.x0;
let xNew;
do {
  xNew = this.f(xOld);
  error = Math.abs((xNew - xOld) / xNew);
  xOld = xNew;
} while (error > this.tolerance);
```

ใช้การวนซ้ำและคำนวณค่าฟังก์ชันซ้ำ ๆ จนกว่าความแตกต่างของผลลัพธ์จะน้อยกว่าเกณฑ์

- Newton–Raphson Method

เป็นวิธีที่ใช้ค่าอนุพันธ์ของฟังก์ชันในการหาค่าราก โดยอาศัยแนวคิดของการหาเส้นสัมผัส (tangent line) ที่จุดปัจจุบันเพื่อประมาณค่ารากใหม่

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

```
while (true) {
  const fx = this.f(x);
  const dfx = (this.f(x + h) - this.f(x - h)) / (2 * h);
  const xNew = x - fx / dfx;
  if (Math.abs(xNew - x) < this.tolerance) break;
  x = xNew;
}
```

ใช้อนุพันธ์ในการหาทิศทางของเส้นสัมผัสเพื่อปรับค่า x ให้เข้าใกล้รากมากที่สุด

- Secant Method

เป็นการปรับปรุงจาก Newton-Raphson โดยไม่ต้องหาค่าอนุพันธ์ ใช้เส้นตรงเชื่อมระหว่างสองจุดแทนเส้นสัมผัส

$$x_{i+1} = x_i - f(x_i) \times \frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})}$$

```
const fx = this.f(x);
const fx_delta = this.f(x + this.delta * x);
const xNew = x - (fx * this.delta * x) / (fx_delta - fx);
```

ใช้ส่วนต่างของฟังก์ชันเพื่อประมาณค่าราก โดยไม่ต้องคำนวณอนุพันธ์

- Taylor Series Method

การขยายฟังก์ชันในรูปของอนุกรมเทย์เลอร์รอบจุดขยาย a เพื่อประมาณค่าฟังก์ชันที่จุด x

$$f(x) \approx f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + \dots$$

```
approximateAtOrder(N) {
  let sum = 0;
  const dx = this.x - this.a;
  for (let n = 0; n <= N; n++) {
    const f_n = this.derivativeAtA(n);
    sum += (f_n * Math.pow(dx, n)) / this.factorial(n);
  }
  return sum;
}
```

ใช้การอนุพันธ์หลายลำดับร่วมกับ factorial เพื่อคำนวณค่า f(x) แบบประมาณ

- Graphical Method

ใช้การวาดกราฟของสมการ f(x)f(x)f(x) แล้วค้นหาจุดที่กราฟตัดแกน x โดยตรวจสอบการเปลี่ยนเครื่องหมายของค่า f(x)

```
for (let x = start; x <= end; x += step) {
  if (this.f(x) * this.f(x + step) < 0) {
    this.results.push({ rootRange: [x, x + step] });
  }
}
```

หาช่วงที่ค่าฟังก์ชันเปลี่ยนเครื่องหมาย เพื่อระบุบริเวณที่น่าจะมีรากของสมการ

บทที่ 3

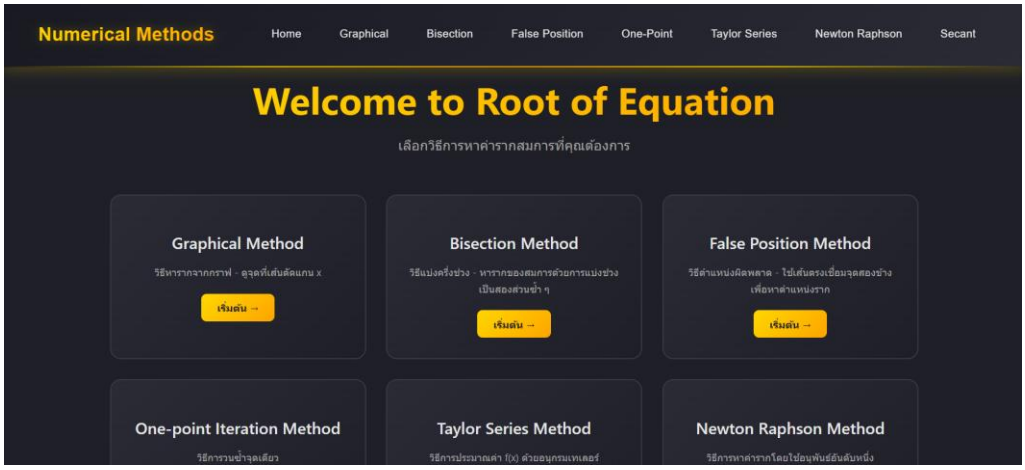
สรุปผลและข้อเสนอแนะ

ปัญหาที่พบระหว่างการพัฒนา

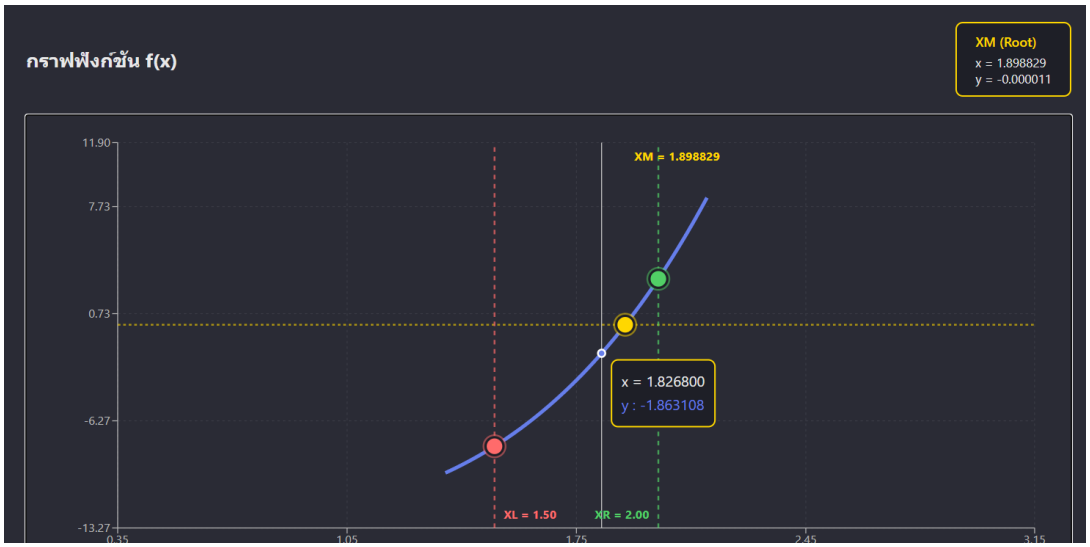
1) เนื่องจาก Notebook พังตอนที่ยังไม่ได้เชื่อม GitHub ทำให้ต้องเริ่มงานใหม่ตั้งแต่การทำเกม ด้วยเวลาที่น้อยเกินไปการเขียนโค้ดจึงทำได้ไม่ดี
2) การรองรับความผิดพลาดจากการพิมพ์ข้อมูลสมการที่ไม่ตรงกับ logic ของการคำนวณบางครั้ง อาจทำให้เกิด error

จุดเด่นของเว็บ

1) มี UI ที่สวยงาม เรียบหรู น่าใช้



2) มีกราฟแบบโต้ตอบ สามารถเลื่อนได้



ข้อเสนอแนะ - นำไปพัฒนาต่อเป็น Full-Stack Website Development