ĐẠI HỌC BÁCH KHOA HÀ NỘI

TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

BÁO CÁO PROJECT I

Đề tài:Tính biểu thức toán học

Nhóm: 4

Mã lớp học: 139350-IT3150

Giáo viên hướng dẫn: Thầy Trần Đình Khang

Danh sách sinh viên thực hiện:

STT	Họ và Tên	MSSV
1	**Ngô Đức Quang Anh**	**20215259**
2	**Hà Thế Hiển**	**20215575**
3	**Nguyễn Hiệp Hồng Quân**	**20215633**

Mục lục :

1.Danh mục công việc

Họ và tên	MSSV	Công việc
Ngô Đức Quang Anh	**20215259**	Tính giá trị biểu thức,thiết kế giao diện,controller
Hà Thế Hiển	**20215575**	Tìm nghiệm của biểu thức
Nguyễn Hiệp Hồng Quân	**20215633**	Kiểm thử, sửa lỗi và tinh gọn

2.Tổng quan về đồ án

Dự án này nhằm mục tiêu xây dựng một ứng dụng máy tính tích hợp tính toán giá trị biểu thức f(x) và tìm nghiệm từ biểu thức đã cho, cung cấp một công cụ hữu ích cho việc tính toán và giải quyết các bài toán liên quan đến hàm số và phương trình.

-Công cụ và ngôn ngữ lập trình: Java,JavaFX

-Tính năng chính:

1. Tính giá trị của biểu thức *f*(*x*):

- Chương trình sẽ cho phép người dùng nhập biểu thức *f*(*x*) trong một ô nhập liệu.
- Hỗ trợ các phép toán cơ bản (+, -, *, /), dấu ngoặc, và các hàm toán học cơ bản (sin, cos, tan, log, sqrt, abs,cosh...).
- Cung cấp khả năng nhập giá trị của *x* để tính giá trị của *f*(*x*).

2. Tính Nghiệm :

- Hỗ trợ tính năng tìm nghiệm *x* trong biểu thức *f*(*x*)=0.
- Sử dụng thuật toán di truyền để xác định nghiệm của biểu thức.

-Thiết Kế và Triển Khai:

1. **Giao Diện Người Dùng (UI):**
    - Thiết kế giao diện người dùng thân thiện và dễ sử dụng.
    - Cung cấp ô nhập biểu thức *f*(*x*) và ô nhập giá trị *x* để người dùng có thể dễ dàng nhập liệu.

2. **Phân Tích và Tính Toán:**
    - Xây dựng các hàm để phân tích biểu thức $f(x)$ và tính giá trị của nó dựa trên giá trị x.
    - Triển khai các thuật toán để tìm nghiệm của x dựa trên biểu thức $f(x)$ được cung cấp.
3. **Xử Lý Ngoại Lệ và Bảo Mật:**
    - Xử lý các trường hợp ngoại lệ, chẳng hạn như biểu thức không hợp lệ, chia cho 0, hoặc giá trị không thể tính toán được.

3.Phân tích triển khai biểu thức fx

3.1 Mô tả thuật toán

Thuật toán có 3 phương thức đảm nhận việc phân tích các phép tính cơ bản.

- Phương thức 1:xử lý các phép cộng và trừ.

- Phương thức 2:xử lý các phép nhân và chia.

- Phương thức 3: xử lý các yếu tố trong biểu thức như số, biểu thức trong dấu ngoặc, hoặc các hàm toán học(vd: sin,cos,sqrt,...)

Độ ưu tiên của phương thức 3 > phương thức 2 > phương thức 1

Vì thế

+ phương thức 1: trước khi thực hiện phép +,- sẽ gọi phương thức 2 thực hiện trước

+ phương thức 2 trước khi thực hiện phép *,/ sẽ gọi phương thức 3(xử lý số,dấu ngoặc,hàm toán học) thực hiện trước

Duyệt lần lượt từng ký tự của chuỗi biểu thức, với mỗi loại kí tự được duyệt sẽ được xử lý theo 3 phương thức ở trên(gọi đệ quy)

3.2 Áp dụng thuật toán tính giá trị biểu thức fx

-Đầu vào là chuỗi biểu thức và giá trị của x

```
public static double evaluateExp(String exp, double n) {
```

-Khởi tạo: vị trí bắt đầu duyệt pos =-1, int ch để lưu trữ 1 ký tự trong quá trình duyệt

```
int pos = -1, ch; // ch lưu 1 ký tự trong quá trình duyệt
6 usages
void nextChar() {
    ++pos;
    if (pos < exp.length()) {
        ch = exp.charAt(pos);
    } else {
        ch = -1;
    }
}
11 usages
boolean eat(int c) {
    while (ch == ' ') {
        nextChar();
    }
    if (ch == c) {
        nextChar();
        return true;
    } else {
        return false; // if char dang duyet != char in exp
    }
}
```

Hàm nextChar(): để duyệt kí tự tiếp theo bằng cách tăng pos lên 1 đơn vị và gán ch= kí tự đang duyệt, nếu đã duyệt hết gán ch=-1 để biểu thị trạng thái kết thúc chuỗi

Hàm eat(): kiểm tra kí tự đang xét có trùng với kí tự mong muốn hay không.Nó sẽ luôn bỏ qua dấu cách, nếu kí tự xét trùng kí tự mong muốn thì gọi hàm nextChar() , ngược lại nếu không phải thì giữ nguyên vị trí đọc chuỗi

Ví dụ, nếu hàm eat('+') được gọi, kiểm tra xem ký tự có phải là dấu '+' hay không. Nếu là dấu '+', nó sẽ ăn ký tự này và di chuyển tới ký tự tiếp theo trong chuỗi. Nếu không phải, hàm sẽ không thay đổi vị trí đọc chuỗi.

Hàm xử lý phép cộng, trừ (parseExpression)

```
3 usages
double parseExpression() {
    double x = parseTerm();
    for (;;) {
        if (eat( c: '+')) {
            x += parseTerm();
        } else if (eat( c: '-')) {
            x -= parseTerm();
        } else {
            return x;
        }
    }
}
```

Trước khi xử lý phép cộng, trừ gọi hàm parseTerm(xử lý *,/) ưu tiên thực hiện trước. Sau đó mới thực hiện phép + hoặc -

Hàm xử lý phép nhân ,chia (parseTerm)

```
3 usages
double parseTerm() {
    double x = parseFactor();
    for (;;) {
        if (eat( c: '*')) {
            x *= parseFactor();
        } else if (eat( c: '/')) {
            x /= parseFactor();
        } else {
            return x;
        }
    }
}
```

Trước khi xử lý phép nhân chia gọi hàm parseFactor ưu tiên thực hiện trước. Sau đó mới thực hiện phép nhân chia

Hàm xử lý các yếu tố trong biểu thức parseFactor()

Hàm này để xử lý dấu dương,âm,xử lý số,biểu thức trong ngoặc , các hàm toán học như (sin,cos,log,sqrt…)

```
double parseFactor() { // xu ly biểu thức âm/dương +bieu thuc trong ngoac +function
    double x;
    int startPos = this.pos;
    if (eat( c: '+')) {
        return parseFactor();
    }
    if (eat( c: '-')) {
        return -parseFactor();
    }

    if (eat( c: '(')) {
        x = parseExpression(); // tinh bieu thuc trong ngoac
        eat( c: ')');
    } else if ((ch >= '0' && ch <= '9') || ch == '.') {
        while ((ch >= '0' && ch <= '9') || ch == '.') {
            nextChar();
        }
        x = Double.parseDouble(exp.substring(startPos, this.pos));
    } else if (ch == 'x') {
        x = n; // Thay thế 'x' bằng giá trị x được truyền vào
        nextChar();
    } // xu ly function:
    else if (ch >= 'a' && ch <= 'z') {
        while (ch >= 'a' && ch <= 'z') {
            nextChar();
        } // after loop -> read character of function
        String func = exp.substring(startPos, this.pos);
        x = parseFactor(); // read next doublevalue after func : sin 90, x=90
```

```
if (func.equals("sin")) {
    x = Math.sin(Math.toRadians(x));
} else if (func.equals("cos")) { // cos(x)^2= cos(x^2)
    x = Math.cos(Math.toRadians(x));
} else if (func.equals("tan")) {
    x = Math.tan(Math.toRadians(x));
} else if (func.equals("cot")) {
    x = 1 / Math.tan(Math.toRadians(x));
} // log can dau ngoac dau tien: log()(exp); ln(exp))
else if (func.equals("log")) {
    eat( c: '(');
    double base = x; // luu ki tu sau log(20)(exp) 20
    x = parseExpression(); // x=exp
    eat( c: ')');
    x = Math.log(x) / Math.log(base);}
else if (func.equals("sinh")) {
    x = Math.sinh(x);
} else if (func.equals("cosh")) {
    x = Math.cosh(x);
} else if (func.equals("log")) {
    x = Math.log(x);
} else if (func.equals("abs")) {
    //  eat('(');
    //  x = parseExpression();
    // eat(')'); //
    x = Math.abs(x);
}
else if (func.equals("ln")) {
    x = Math.log(x);
```

```
    }else if (func.equals("sqrt")) {
        //  eat('(');
        //  x = parseExpression();
        // eat(')'); //
        if (x > 0) {
            x = Math.sqrt(x);
        } else {
            x = Double.POSITIVE_INFINITY;// truong hop sqrt <0
        }
    } else {
        x = Double.NEGATIVE_INFINITY; // truong hop viet sai ten func
    }
} else {
    x =  Double.NEGATIVE_INFINITY; // truong hop ki tu dac biet
}

if (eat( c: '^')) {
    x = Math.pow(x, parseFactor());
}

return x;
}
```

-Xử lý dấu dương, âm : Nếu gặp dấu cộng (+), nó gọi đệ quy xử lý phần tử tiếp theo và trả về kết quả. Tương tự, nếu gặp dấu trừ (-), nó xử lý phần tử tiếp theo và trả về kết quả với giá trị âm

```
double parseFactor() { // xu ly biểu thức âm/dương +bieu thuc trong ngoac +function
    double x;
    int startPos = this.pos;
    if (eat( c: '+')) {
        return parseFactor();
    }
    if (eat( c: '-')) {
        return -parseFactor();
    }
```

-Xử lý dấu ngoặc : ăn dấu ngoặc đơn và gọi phương thức parseExpression() để tính biểu thức trong ngoặc, ăn dấu ngoặc đóng.

```
    if (eat( c: '(')) {
        x = parseExpression(); // tinh bieu thuc trong ngoac
        eat( c: ')');
    } else if ((ch >= '0' && ch <= '9') || ch == '.') {
```

-Xử lý số : chuyển đổi chuỗi số (String) từ biểu thức thành giá trị số thực (double) bằng cách duyệt những kí tự liên tiếp là số.

```
    } else if ((ch >= '0' && ch <= '9') || ch == '.') {
        while ((ch >= '0' && ch <= '9') || ch == '.') {
            nextChar();
        }
        x = Double.parseDouble(exp.substring(startPos, this.pos));
    } else if (ch == 'x') {
        x = n; // Thay thế 'x' bằng giá trị x được truyền vào
        nextChar();
    } // xu ly function:
```

-Xử lý hàm : duyệt các kí tự chữ liên tiếp rồi ghép lại, nếu thỏa mãn các hàm thì xử lý biểu thức ở trong hàm đó bằng cách gán x=parseFactor()

```
else if (ch >= 'a' && ch <= 'z') {
    while (ch >= 'a' && ch <= 'z') {
        nextChar();
    } // after loop -> read character of function
    String func = exp.substring(startPos, this.pos);
    x = parseFactor(); // read next doublevalue after func : sin 90, x=90
    if (func.equals("sin")) {
        x = Math.sin(Math.toRadians(x));
    } else if (func.equals("cos")) { // cos(x)^2= cos(x^2)
        x = Math.cos(Math.toRadians(x));
    } else if (func.equals("tan")) {
        x = Math.tan(Math.toRadians(x));
    } else if (func.equals("cot")) {
```

Kết quả hiện thị

Sau khi phân tích biểu thức, chương trình có thể tính giá trị của biểu thức dựa trên giá trị của biến x được cung cấp.

Với các trường hợp lỗi vi phạm điều kiện xác đinh sẽ trả về kết quả dương vô cùng , nhập sai biểu thức thì trả về kết quả âm vô cùng để xử lý trạng thái.

4.Phân tích tìm nghiệm biểu thức bằng thuật toán di truyền

4.1 Cơ sở lý thuyết giải thuật di truyền (Genetic Algorithm)

Giải thuật di truyền (GA) là một thuật toán tìm kiếm tối ưu dựa trên mô phỏng quá trình tiến hóa tự nhiên. GA bắt đầu với một quần thể các cá thể, mỗi cá thể đại diện cho một giải pháp tiềm năng cho bài toán. Quần thể này sau đó được tiến hóa qua nhiều thế hệ, với mỗi thế hệ bao gồm một tập hợp các cá thể mới được tạo ra từ các cá thể của thế hệ trước.

Quá trình tiến hóa của GA bao gồm bốn bước chính:

- Sàng lọc: Các cá thể trong quần thể được đánh giá dựa trên một hàm mục tiêu, và các cá thể có giá trị hàm mục tiêu cao hơn sẽ được ưu tiên chọn lọc.
- Tái tổ hợp: Hai cá thể được chọn lọc sẽ được tái tổ hợp với nhau để tạo ra một cá thể mới. Quá trình tái tổ hợp có thể được thực hiện theo nhiều cách khác nhau, chẳng hạn như tái tổ hợp điểm, tái tổ hợp tuyến tính, hoặc tái tổ hợp chia đôi.
- Đột biến: Một số cá thể trong quần thể có thể được đột biến, tức là một hoặc nhiều gen của chúng sẽ được thay đổi ngẫu nhiên. Đột biến giúp tạo ra sự đa dạng trong quần thể và giúp GA tránh bị mắc kẹt trong các giải pháp cục bộ tối ưu.
- Chọn lọc: Các cá thể trong quần thể mới được chọn lọc để tiếp tục tiến hóa cho thế hệ tiếp theo.

Quá trình này được lặp lại cho đến khi đạt được một giải pháp tối ưu hoặc cho đến khi đạt được một số giới hạn về số lượng thế hệ.

Các thành phần của giải thuật di truyền

Giải thuật di truyền có bốn thành phần chính:

- Quần thể: Quần thể là một tập hợp các cá thể, mỗi cá thể đại diện cho một giải pháp tiềm năng cho bài toán.
- Hàm mục tiêu: Hàm mục tiêu đánh giá chất lượng của các cá thể trong quần thể.
- Các toán tử di truyền: Các toán tử di truyền bao gồm sàng lọc, tái tổ hợp, và đột biến.
- Các tham số: Các tham số của GA bao gồm kích thước quần thể, tỷ lệ tái tổ hợp, và tỷ lệ đột biến.

Ứng dụng của giải thuật di truyền

Giải thuật di truyền có thể được áp dụng để giải nhiều loại bài toán khác nhau, bao gồm:

- Tối ưu hóa: GA có thể được sử dụng để tìm kiếm các giải pháp tối ưu cho các bài toán tối ưu hóa, chẳng hạn như quy hoạch tuyến tính, tối ưu hóa liên tục, và tối ưu hóa bộ số.
- Tìm kiếm: GA có thể được sử dụng để tìm kiếm các giải pháp cho các bài toán tìm kiếm, chẳng hạn như tìm kiếm đường đi ngắn nhất, tìm kiếm cấu trúc tốt nhất, và tìm kiếm dữ liệu.
- Lập kế hoạch: GA có thể được sử dụng để lập kế hoạch cho các hệ thống phức tạp, chẳng hạn như lập kế hoạch sản xuất, lập kế hoạch vận tải, và lập kế hoạch chiến lược.

Ưu điểm và nhược điểm của giải thuật di truyền

Ưu điểm:

- GA có thể giải quyết nhiều loại bài toán khác nhau.
- GA có thể tìm kiếm các giải pháp toàn cục tối ưu.
- GA có thể hoạt động tốt với các bài toán có hàm mục tiêu phức tạp.

Nhược điểm:

- GA có thể tốn nhiều thời gian để chạy.
- GA có thể bị mắc kẹt trong các giải pháp cục bộ tối ưu.

Kết luận

Giải thuật di truyền là một thuật toán tìm kiếm tối ưu mạnh mẽ có thể được áp dụng để giải nhiều loại bài toán khác nhau. GA có thể tìm kiếm các giải pháp toàn cục tối ưu, nhưng có thể tốn nhiều thời gian để chạy và có thể bị mắc kẹt trong các giải pháp cục bộ tối ưu.

4.2 Áp dụng vào bài toán giải phương trình f(x) = 0

4.2.1 Phương pháp giải

Biểu diễn lời giải dưới dạng gen

Đầu tiên ta cần biểu diễn lời giải bài toán dưới dạng gen. Trong trường hợp này, ta biểu diễn lời giải dưới dạng một chuỗi nhị phân 32 bit tuân theo tiêu chuẩn Single precision floating point format.

Ví dụ chuỗi “01000001101110000000000000000000” là biểu diễn nhị phân biểu diễn cho lời giải x = 23.5.

Khởi tạo quần thể ban đầu

Tiếp theo, ta cần khởi tạo một quần thể ban đầu gồm các cá thể. Mỗi cá thể trong quần thể là một chuỗi nhị phân biểu diễn cho một lời giải. Các cá thể trong quần thể có thể được khởi tạo ngẫu nhiên hoặc dựa trên một số thông tin ban đầu về bài toán.

Trong bài toán này, quần thể bao gồm các cá thể, mỗi cá thể có độ dài 32 bit được tạo ngẫu nhiên.

```
1 usage
private static List<String> initPopulation(int populationSize) {
    List<String> population = new ArrayList<>();
    Random random = new Random();

    for (int i = 0; i < populationSize; i++) {
        StringBuilder individual = new StringBuilder();
        for (int j = 0; j < 32; ++j) {
            char bit = (char) ('0' + random.nextInt( bound: 2));
            individual.append(bit);
        }
        population.add(individual.toString());
    }

    return population;
}
```

Hàm "initPopulation" khởi tạo một quần thể với kích thước đặt trước. Sử dụng đối tượng Random(), mỗi cá thể được tạo ngẫu nhiên bằng cách thêm 32 bit "0" hoặc "1" vào chuỗi.

Đánh giá các cá thể

Mỗi cá thể trong quần thể sẽ được đánh giá bằng cách tính giá trị hàm mục tiêu. Giá trị hàm mục tiêu càng nhỏ thì cá thể càng tốt.

Trong bài toán này, ta đang chọn hàm mục tiêu là lf(x)l với f(x) là hàm số ta cần tìm nghiệm. Giá trị của hàm mục tiêu càng gần 0 thì cá thể càng tốt.

```
public static float func(float x, String str) {
    Tinhfx instance = new Tinhfx();
    double result = instance.exp(str, x);
    return Math.abs((float) result);
}
```

Hàm "func" với giá trị đầu vào là x và hàm số f(x) cần tìm nghiệm sẽ trả về giá trị của hàm mục tiêu của x.

Chọn lọc

Các cá thể trong quần thể sẽ được chọn lọc để tham gia vào quá trình sinh sản. Các cá thể có giá trị hàm mục tiêu càng nhỏ thì khả năng được chọn càng cao.

```
Collections.sort(funcVal, (a, b) -> Float.compare(a.getSecond(), b.getSecond()));
series.add(generation, funcVal.get(0).getSecond());
int newPopulationSize = populationSize / 4;
List<String> newPopulation = new ArrayList<>();

for (int i = 0; i < newPopulationSize; ++i) {
    newPopulation.add(funcVal.get(i).getFirst());
}
```

Trong bài toán này, ta sắp xếp các cá thể theo đáp ứng của hàm mục tiêu và số cá thể của quần thể đáp ứng hàm mục tiêu tốt nhất sẽ được chọn và tham gia vào quá trình tiếp theo.

Sinh sản

Các cá thể được chọn lọc sẽ được sinh sản để tạo ra một quần thể mới. Quá trình sinh sản có thể được thực hiện theo các phương pháp sau:

- Lai: Hai cá thể được chọn ngẫu nhiên sẽ được lai tạo để tạo ra một cá thể mới.
- Đột biến: Một cá thể được chọn ngẫu nhiên sẽ bị đột biến để tạo ra một cá thể mới.

Trong bài toán này, quá trình lai được thực hiện bởi cách chọn ngẫu nhiên một điểm cắt trong bộ gen để tạo ra con từ hai cá thể cha mẹ. Hàm lai được thực hiện trong đoạn code dưới đây.

```
private static String crossover(String parent1, String parent2) {
    int crossoverPoint = new Random().nextInt(parent1.length());
    String child = parent1.substring(0, crossoverPoint) + parent2.substring(crossoverPoint);
    return child;
}
```

Để tăng sự đa dạng cũng như để tránh kẹt ở một điểm nào đó trong quá trình tiến hóa, quá trình đột biến được sử dụng trong một vài cá thể.

```
private static void mutate(StringBuilder individual, double mutationRate) {
    Random random = new Random();
    for (int i = 0; i < individual.length(); i++) {
        double mutationChance = random.nextDouble();

        if (mutationChance < mutationRate) {
            char bit = (individual.charAt(i) == '0') ? '1' : '0';
            individual.setCharAt(i, bit);
        }
    }
}
```

Hàm "mutate" ngẫu nhiên đảo ngược các bit trong chuỗi nhị phân của cá thể với một xác suất là "mutationRate".

Ở bài toán này ta sử dụng phương pháp "Dynamic mutation rate" hay "Xác suất đột biến động", xác suất đột biến sẽ giảm dần theo số thế hệ nhằm đảm bảo sự ổn định của quần thể về sau – khi quần thể dần hội tụ. Xác suất đột biến được biểu diễn như sau

```
double mutationRate = 0.05 * (maxGeneration - generation) / maxGeneration;
```

Sau đó các cá thể mới sẽ thay thế các cá thể bị loại bỏ để tạo thành quần thể mới, các cá thể mới được tạo bởi cách lai từ các cá thể cha mẹ từ quần thể ban đầu sau đó đột biến.

```
    for (int i = newPopulationSize; i < populationSize; ++i) {
        int parentIndex1 = new Random().nextInt(newPopulationSize);
        int parentIndex2 = new Random().nextInt(newPopulationSize);

        String parent1 = newPopulation.get(parentIndex1);
        String parent2 = newPopulation.get(parentIndex2);

        String child = crossover(parent1, parent2);
        StringBuilder individual = new StringBuilder(child);
        mutate(individual, mutationRate);
        newPopulation.add(child);
    }
```

Thay thế

Quần thể mới sẽ thay thế quần thể cũ. Quá trình này sẽ được lặp lại cho đến khi tìm được nghiệm hoặc đạt số thế hệ tối đa.

```
while (generation < maxGeneration)
```

```
if(funcVal.get(0).getSecond() < 0.00001){
    break;
}
```

Kết quả hiển thị

Biểu Đồ và Kết Quả: Hiển thị biểu đồ và kết quả tìm kiếm trong cửa sổ JFrame sử dụng thư viện JFreeChart.

Xử Lý Kết Quả Cuối Cùng: Trả về kết quả cuối cùng, bao gồm giá trị sai số tốt nhất và giá trị x tương ứng.

4.2.2 Kết Luận

Ta đã triển khai một giải thuật di truyền để giải phương trình, sử dụng quần thể của các chuỗi bit để tìm kiếm giá trị x. Biểu đồ giúp theo dõi sự hội tụ của giải thuật và đánh giá hiệu suất của nó qua các thế hệ.

Giải thuật đã giúp giải gần đúng phương trình f(x) = 0 với sai số không vượt quá 10^{-4}.

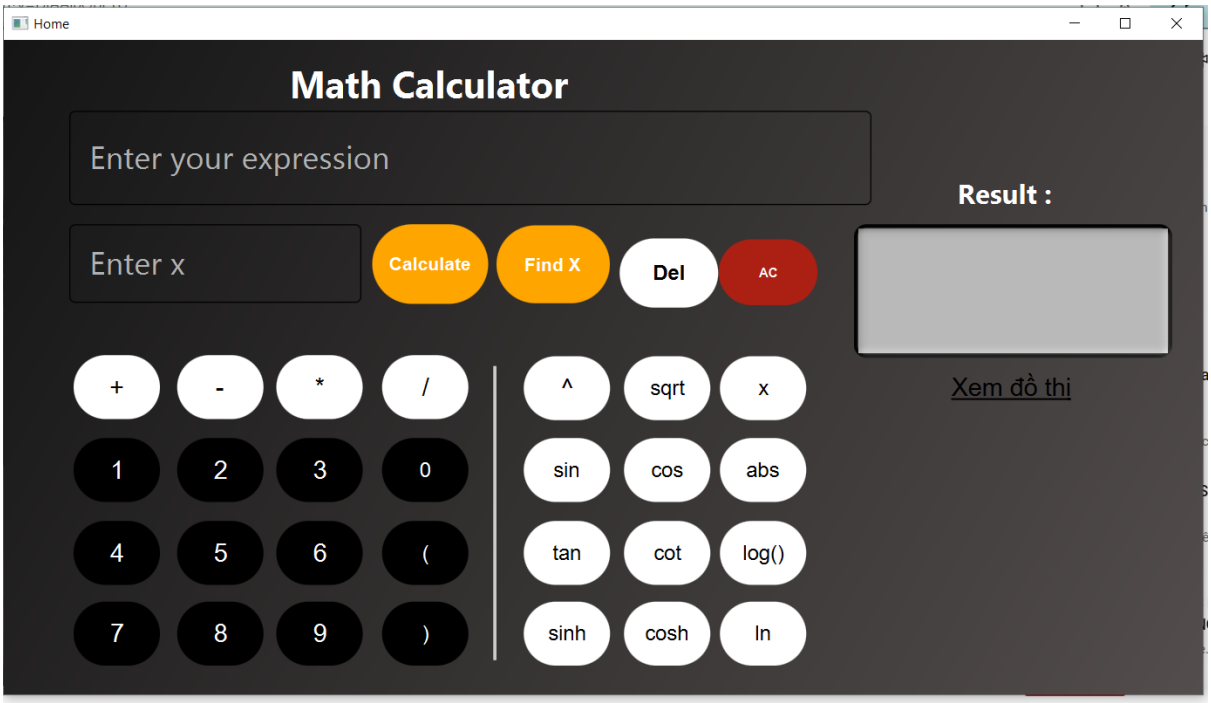
5.Demo sản phẩm

5.1 Demo

5.1.1 Hướng dẫn sử dụng:

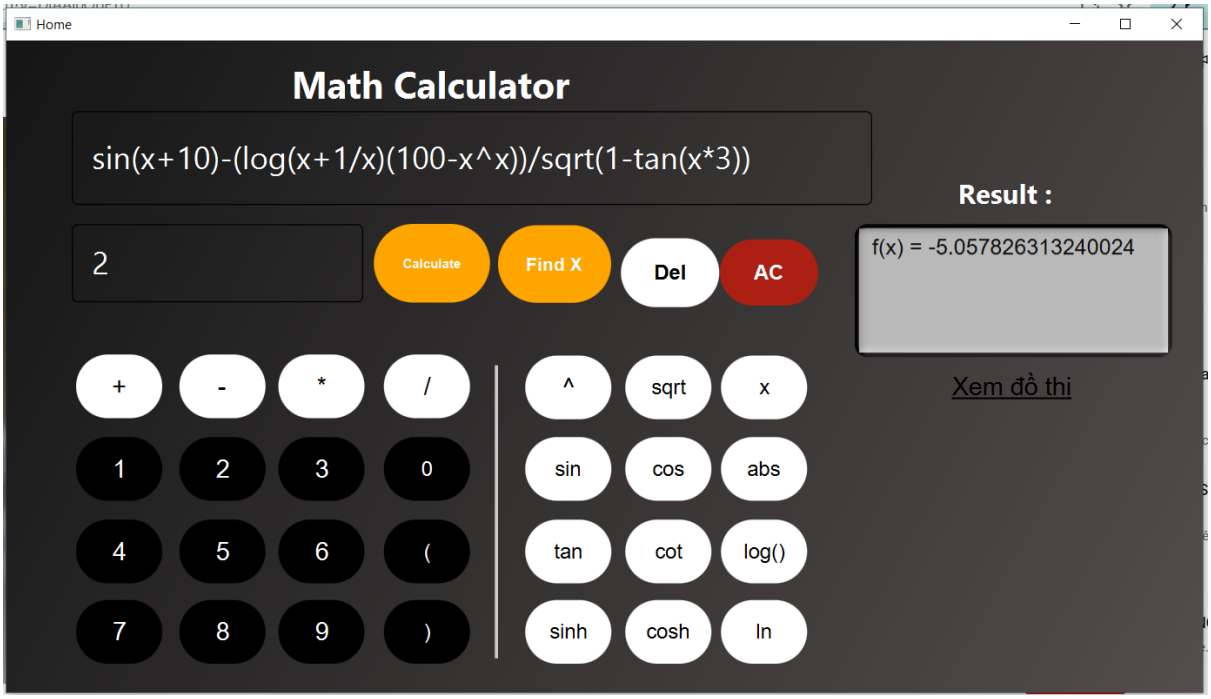
+Ô textfield thứ nhất để nhập biểu thức fx
+Ô textfield thứ 2 để nhập giá trị của x , nếu tìm x thì không nhất thiết phải nhập ô này
+Các nút bấm : gồm các số , toán tử , và các hàm. Khi nhấn nút sẽ xuất lên ô textfield của fx
+Nút Del : xóa kí tự cuối cùng
+Nút AC : xóa toàn bộ dữ liệu
+Nút Calculate : Để tính giá trị fx và xuất ra màn hình Result
+Nút Find X : Để tìm nghiệm x và xuất ra màn hình Result
+Nút Xem đồ thị : Nhấp vào để xem biểu đồ hội tụ của thuật toán di truyền

5.1.2 Màn hình chính:



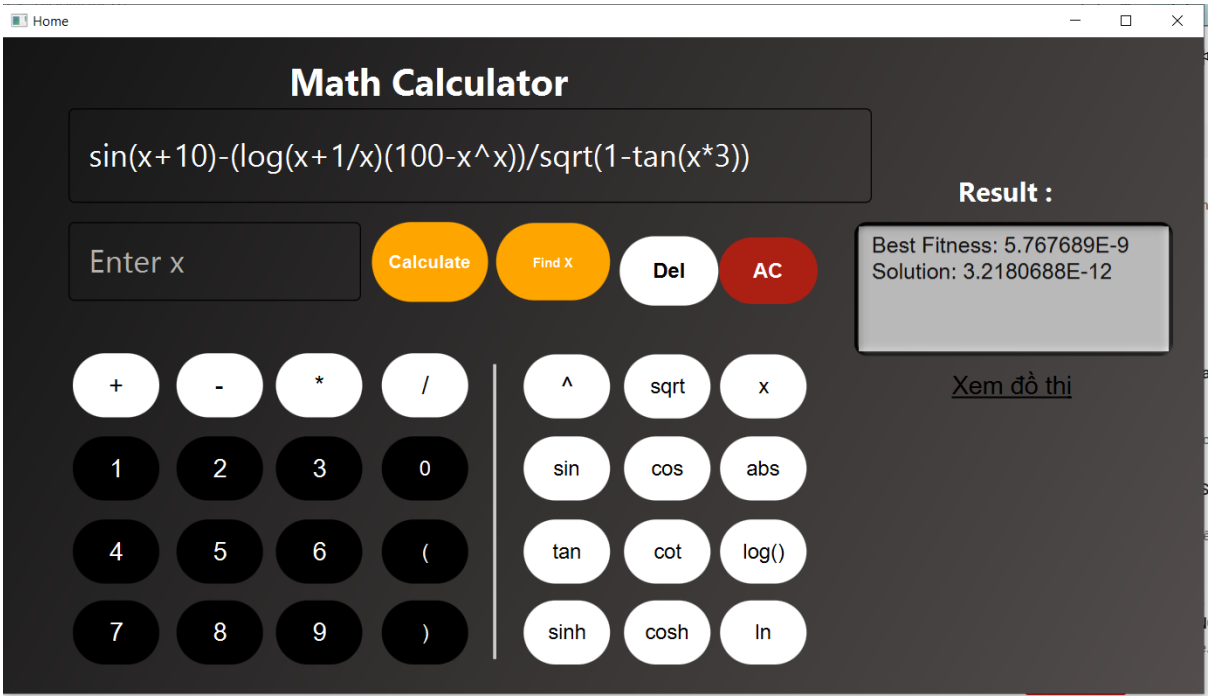
5.1.3 Tính biểu thức fx :

với fx=sin(x+10)-(log(x+1/x)(100-x^x))/sqrt(1-tan(x*3)) và x =2

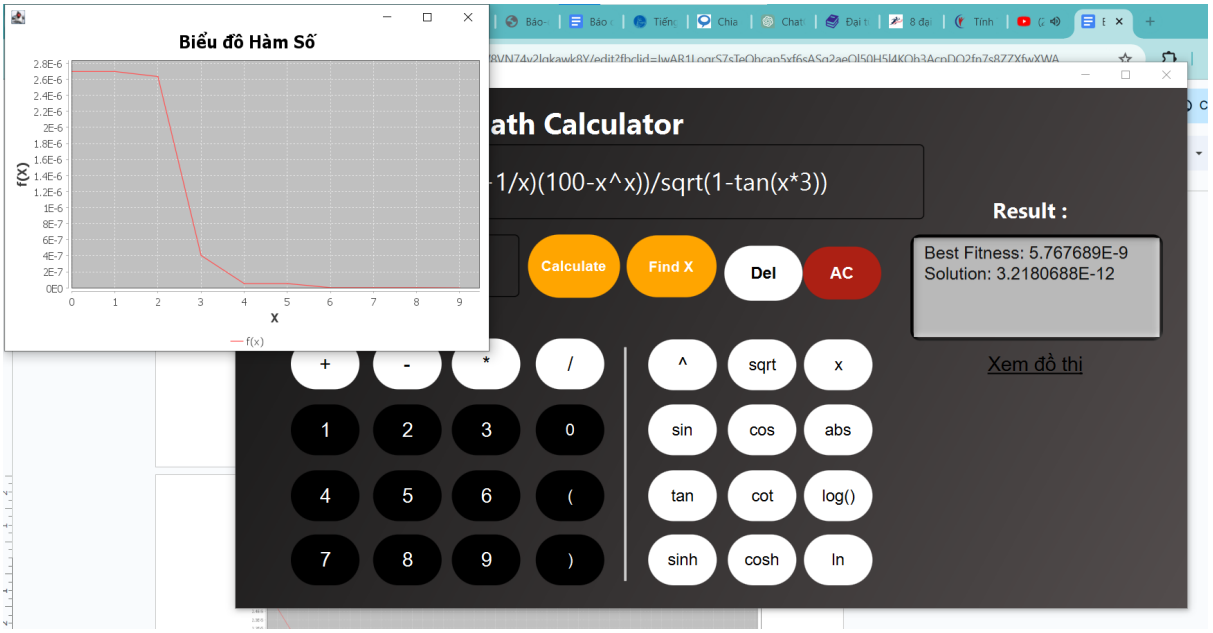


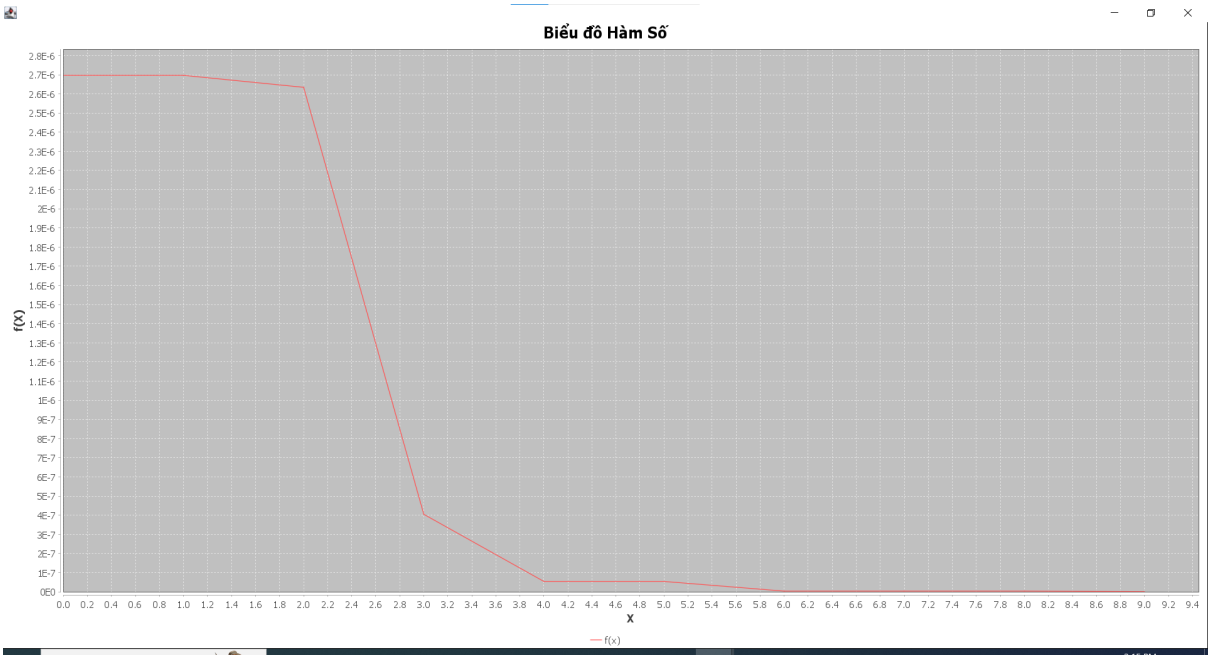
5.1.4 Tìm x:

Biểu thức : sin(x+10)-(log(x+1/x)(100-x^x))/sqrt(1-tan(x*3))=0



Xem biểu đồ hội tụ :

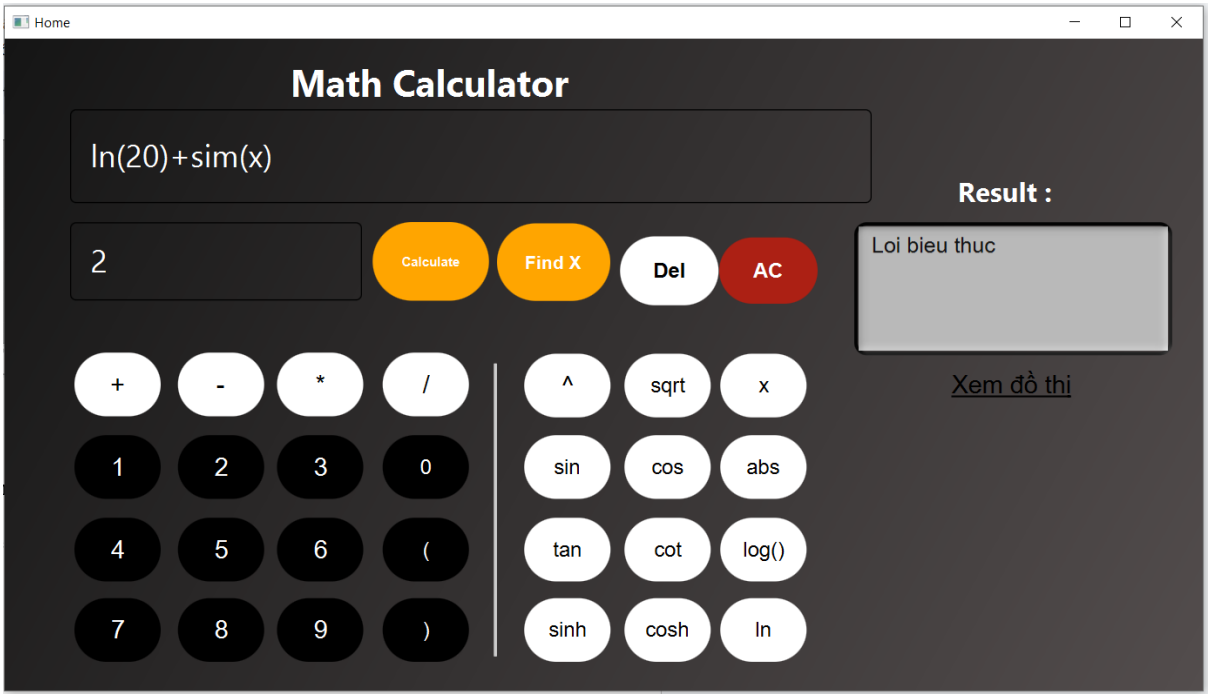




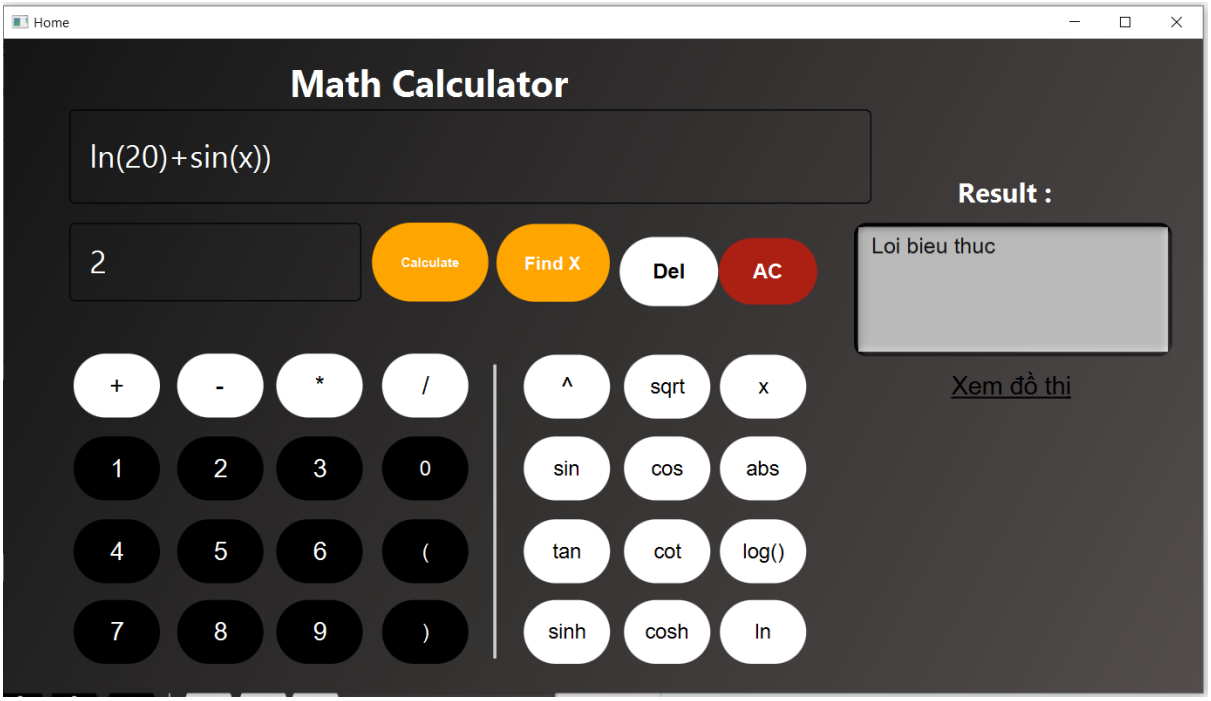
5.2 Xử lý ngoại lệ

5.2.1 Khi biểu thức nhập sai

- Nhập sai hàm không tồn tại: ví dụ nhập sin -> sim:

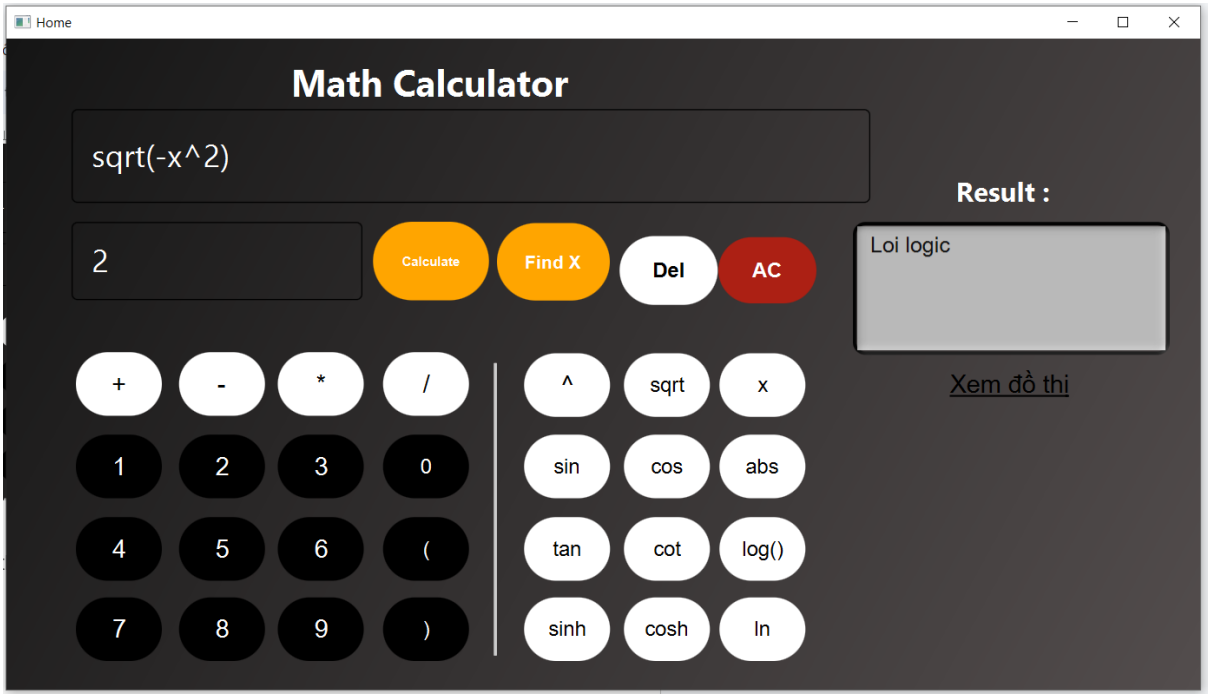


-Thừa dấu ngoặc đơn:



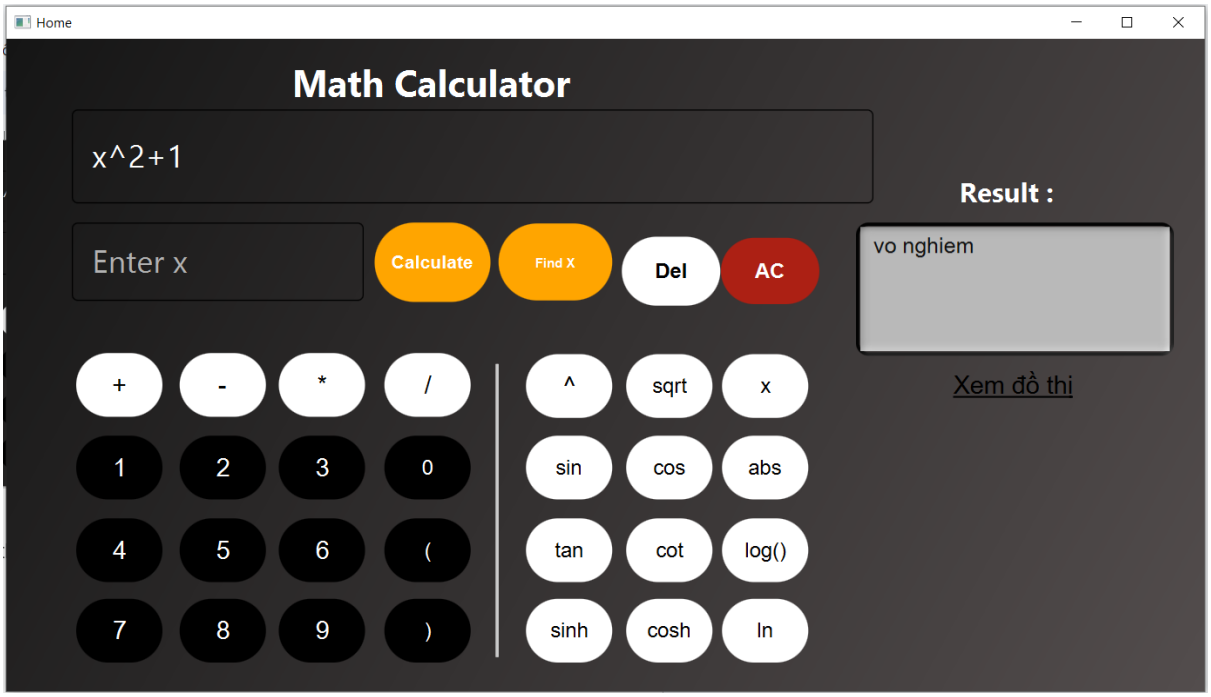
5.2.2 Biểu thức có điều kiện xác định sai

-Ví dụ: biểu thức fx= sqrt(-x^2) và x=2 vi phạm điều kiện xác định



5.2.3 Biểu thức vô nghiệm

Ví dụ : x^2+1=0 không có nghiệm x thỏa mãn



6. Kết luận

Tính năng chính của chương trình:

Chương trình được thiết kế để tính toán giá trị của một biểu thức toán học bất kỳ. Nó hỗ trợ các phép tính cơ bản như cộng, trừ, nhân, chia, cũng như các hàm toán học như sin, cos, tan, log, sqrt và nhiều hàm khác. Người dùng có thể cung cấp biểu thức và giá trị của biến 'x' để tính toán giá trị của biểu thức tại điểm cụ thể.

Các điểm mạnh của chương trình:

1. **Hỗ trợ nhiều phép tính và hàm toán học:** Chương trình có khả năng xử lý nhiều loại biểu thức phức tạp chứa các phép tính và hàm toán học khác nhau, giúp người dùng tính toán các biểu thức phức tạp.
2. **Tính linh hoạt:** Cung cấp khả năng thay thế giá trị của biến 'x' bằng các giá trị cụ thể để tìm giá trị của biểu thức ở các điểm khác nhau.
3. **Độ chính xác cao:** Sử dụng các hàm toán học có sẵn trong Java như Math.sin, Math.cos, Math.log, v.v., giúp chương trình đảm bảo tính chính xác cao trong kết quả tính toán.

Các hạn chế và cải tiến có thể:

1. **Hạn chế trong xử lý lỗi:** Hiện tại, chương trình có thể không cung cấp thông báo lỗi cụ thể khi gặp lỗi trong quá trình phân tích biểu thức. Thông báo lỗi rõ ràng hơn có thể giúp người dùng hiểu rõ hơn về lỗi và cách sửa.
2. **Hỗ trợ biểu thức phức tạp hơn:** Các biểu thức phức tạp có thể đòi hỏi một cách tiếp cận xử lý phức tạp hơn, ví dụ như hỗ trợ biểu thức chứa ma trận, phương trình vi phân, hoặc hàm đa biến.

Chương trình tính toán biểu thức này cung cấp một cách tiếp cận linh hoạt và mạnh mẽ để tính toán giá trị của các biểu thức toán học phức tạp. Mặc dù có những hạn chế nhất định, nhưng tiềm năng phát triển và ứng dụng của chương trình là rất lớn trong nhiều lĩnh vực khác nhau.