МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ
КАФЕДРА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Лабораторна робота № 3

з дисципліни «Системи реального часу»

на тему *«Дослідження генетичного алгоритму, реалізація задачі розкладання числа на прості множники, дослідження нейронних мереж. Модель Perceptron»*

Виконав:
студент гр. ІП-83
Бойко А.О.

Перевірив:
Регіда П.Г.

Київ 2021

Завдання

1. Розробити програма для факторизації заданого числа методом Ферма. Реалізувати користувацький інтерфейс з можливістю вводу даних.
2. Поріг спрацювання: P = 4 Дано точки: A(0,6), B(1,5), C(3,3), D(2,4). Швидкості навчання: δ = {0,001; 0,01; 0,05; 0.1; 0.2; 0,3} Дедлайн: часовий = {0.5c; 1c; 2c; 5c}, кількість ітерацій = {100;200;500;1000} Обрати швидкість навчання та дедлайн. Налаштувати Перцептрон для даних точок. Розробити відповідний мобільний додаток і вивести отримані значення. Провести аналіз витрати часу та точності результату за різних параметрах навчання.
3. Налаштувати генетичний алгоритм для знаходження цілих коренів діофантового рівняння ax1+bx2+cx3+dx4=y. Розробити відповідний мобільний додаток і вивести отримані значення. Провести аналіз витрат часу на розрахунки.

Програмний код

1.

```
import 'dart:math';

List<int> fermat(int n) {
  int x = sqrt(n).ceil();
  int y2 = x * x - n;
  int y = sqrt(y2).toInt();
  while (y2 != y * y) {
    x++;
    y2 = x * x - n;
    y = sqrt(y2).toInt();
  }
  return [x - y, x + y];
}
```

```
import 'package:lab_3/core/fermat_decomposition.dart';
import 'package:flutter/services.dart';
import 'package:flutter/material.dart';

class FermatScreen extends StatefulWidget {
  @override
  State<StatefulWidget> createState() => FermatScreenState();
}

class FermatScreenState extends State<FermatScreen> {
  final _formKey = GlobalKey<FormState>();
  int number;
  dynamic result;
  var time;
```

```
Widget build(BuildContext context) {
 return Form(
  key: _formKey,
  child: Column(
   children: [
    Row(children: [
     Container(
      padding: EdgeInsets.all(10),
      child: Text(
       'Number:',
       style: TextStyle(fontSize: 24),
      ),
     ),
     Expanded(
      child: Container(
       padding: EdgeInsets.all(10),
       child: TextFormField(
        validator: (value) {
         if (value == null) {
          return null;
         }
         number = num.tryParse(value);
         if (number == null) {
          return '"$value" is not a valid number';
         }
         return null;
        },
        keyboardType: TextInputType.number,
       ),
      ),
     ),
    ]),
    SizedBox(height: 10),
    Container(
     padding: EdgeInsets.all(8),
     child: ElevatedButton(
      onPressed: () {
       if (_formKey.currentState.validate()) {
        setState(() {
         var timer = Stopwatch();
         timer.start();
         result = fermat(number);
         timer.stop();
         time = timer.elapsedMicroseconds;
        });
       }
      },
      child: Text('Find'),
     ),
    ),
    Container(
     child: Row(
      children: [
       Text(
        'Results',
```

```
              style: TextStyle(fontSize: 24),
            ),
            Spacer(),
            Text(
              result == null ? '' : 'p = ${result[0]}',
              style: TextStyle(fontSize: 24),
            ),
            Spacer(),
            Text(
              result == null ? '' : 'q = ${result[1]}',
              style: TextStyle(fontSize: 24),
            ),
            Spacer(),
          ],
        ),
      ),
      SizedBox(height: 20),
      Container(
        child: Row(
          children: [
            Text(
              'Time',
              style: TextStyle(fontSize: 24),
            ),
            Spacer(),
            Text(
              time == null ? '' : '$time ms',
              style: TextStyle(fontSize: 24),
            ),
            Spacer(),
          ],
        ),
      )
    ],
  ),
 );
 }
}
```

2)

```
dynamic perceptron(List<List<int>> points, int p) =>
   (int maxItr, double maxTime, double sigma) {
     double w1 = 0;
     double w2 = 0;
     double time = 0;
     int itr = 0;
     var timer = Stopwatch();
     timer..start();
     while (maxItr > itr && maxTime > time) {
       for (var point in points) {
         var y = w1 * point[0] + w2 * point[1];
         var delta = p - y;
         var getDelta = delta > 0 ? delta : 0;
         w1 = w1 + point[0] * getDelta * sigma;
```

```
        w2 = w2 + point[1] * getDelta * sigma;
      }
      time = (timer.elapsedMilliseconds) / 1000;
      itr++;
    }
    return [w1, w2, itr, time];
  };

import 'package:flutter/services.dart';
import 'package:flutter/material.dart';
import 'package:lab_3/core/perceptron.dart';

class PerceptronScreen extends StatefulWidget {
 @override
 State<StatefulWidget> createState() => PerceptronScreenState();
}

class PerceptronScreenState extends State<PerceptronScreen> {
 final _formKey = GlobalKey<FormState>();
 dynamic start = perceptron([
  [0, 6],
  [1, 5],
  [3, 3],
  [2, 4]
 ], 4);
 int maxItr;
 double maxTime;
 double sigma;
 dynamic result;
 Widget build(BuildContext context) {
  return Form(
    key: _formKey,
    child: Column(crossAxisAlignment: CrossAxisAlignment.start, children: [
     Row(
      children: [
       Container(
         padding: EdgeInsets.all(8.0),
         child: Text(
          'Iter num:',
          style: TextStyle(fontSize: 20.0),
         )),
       Expanded(
         child: Container(
           padding: EdgeInsets.all(8.0),
           child: TextFormField(
            validator: (value) {
             if (value == null) {
              return null;
             }
             maxItr = num.tryParse(value);
             if (maxItr == null) {
              return '"$value" is not a valid number';
             }
             return null;
            },
```

```
            keyboardType: TextInputType.number,
          ))),
  ],
),
Row(
  children: [
    Container(
      padding: EdgeInsets.all(8.0),
      child: new Text(
        'Max time:',
        style: TextStyle(fontSize: 20.0),
      )),
    Expanded(
      child: Container(
        padding: EdgeInsets.all(8.0),
        child: TextFormField(
          validator: (value) {
            if (value == null) {
              return null;
            }
            maxTime = num.tryParse(value);
            if (maxTime == null) {
              return '"$value" is not a valid number';
            }
            return null;
          },
          keyboardType: TextInputType.number,
        ))),
  ],
),
Row(
  children: [
    Container(
      padding: EdgeInsets.all(8.0),
      child: new Text(
        'Learning speed:',
        style: TextStyle(fontSize: 20.0),
      )),
    Expanded(
      child: Container(
        padding: EdgeInsets.all(8.0),
        child: TextFormField(
          validator: (value) {
            if (value == null) {
              return null;
            }
            sigma = num.tryParse(value);
            if (sigma == null) {
              return '"$value" is not a valid number';
            }
            return null;
          },
          keyboardType: TextInputType.number,
        ))),
  ],
```

```
        ),
        SizedBox(height: 10.0),
        Container(
          padding: EdgeInsets.all(8.0),
          child: RaisedButton(
            onPressed: () {
              if (_formKey.currentState.validate()) {
                setState(() {
                  result = start(maxItr, maxTime, sigma);
                });
              }
            },
            child: Text('Find'),
            color: Colors.blue,
            textColor: Colors.white,
          ),
        ),
        SizedBox(height: 50.0),
        Text(
          result == null
              ? ''
              : ' w1 = ${result[0]} \n '
                'w2 = ${result[1]}  \n '
                'Iterations = ${result[2]} \n '
                'Time = ${result[3]} s',
          style: TextStyle(fontSize: 30.0),
        ),
      ]));
  }
}
```

3)

```
import 'dart:math';

List<List<int>> generatePopulation(int y, int len, int maxPop){
  List<List<int>> population = List.generate(maxPop, (index) => List.generate(len, (index) =>
Random().nextInt((y).ceil())));
  return population;
}

int fitness(List<int> gen, List<int> data){
  int y = data.last;
  int sum = 0;
  for(int i = 0; i < gen.length; i++){
    sum += ((gen[i] * data[i]));
  }
  return (sum - y).abs();
}

List<int> createFitness(List<List<int>> population, List<int> data){
  int len = population.length;
  return List.generate(len, (index) => fitness(population[index], data));
}
```

```
List<double> generateLikelihoods(List<int> fitnesses){
  int len = fitnesses.length;
  dynamic last = 0;
  double mul = fitnesses.map((e) => 1/e).toList().reduce((value, element) => value + element);
  List<double> res = List.filled(len, 0);
  for(int i = 0; i < len; i++){
    res[i] = (last + (1/fitnesses[i])/mul);
    last = res[i];
  }
  return res;
}

int getIndex(double val, List<List<int>> population, List<double>likelihood) {
  double last = 0;
  int len = population.length;
  for(int i = 0; i < len; i++) {
    if (last <= val && val <= likelihood[i]) return i;
    else last = likelihood[i];
  }
  return len - 1;
}

List<List<int>> createNewPopulation(List<List<int>> population, List<double>likelihood, int y, double chance){
  int len = population.length;
  for(int i = 0; i < len; i++) {
    int parent1 = 0, parent2 = 0, iterations = 0;
    while(parent1 == parent2 || population[parent1] == population[parent2]){
      parent1 = getIndex((Random().nextDouble()), population, likelihood);
      parent2 = getIndex((Random().nextDouble()), population, likelihood);
      if (++iterations > (pow(len, 2))) break;
    }
    population[i] = crossover(population[parent1], population[parent2], y, chance);
  }
  return population;
}

List<int> crossover(List<int> p1, List<int> p2, int y, double chance) {
  int len = p1.length;
  int flag = Random().nextInt(len);
  List<int> child = List.filled(len, 0);
  for(int i = 0; i < len; i++) {
    if(flag < i){
      child[i] = p1[i];
    }
    child[i] = p2[i];
    if (Random().nextDouble() < chance) child[i] = Random().nextInt(y);
  }
  return child;
}

dynamic solve(List<int> data, int maxItr, int maxPop, double chance){
  int len = data.length - 1;
  int y = data.last;
  int numberPop = 0;
  List<List<int>> population = generatePopulation(y, len, maxPop);
```

```
  while(maxItr > 0){
    List<int> fitnesses = createFitness(population, data);
    int i = fitnesses.indexOf(0);
    if(i != -1){
      return [population[i], numberPop];
    }
    List<double> likelihood = generateLikelihoods(fitnesses);
    List<List<int>> child = createNewPopulation(population, likelihood, y, chance);
    numberPop++;
    population = child;
    maxItr--;
  }
  return ["Max iterations", numberPop];
}

import 'package:flutter/services.dart';
import 'package:flutter/material.dart';
import 'package:lab_3/core/genetic.dart';

class GeneticScreen extends StatefulWidget {
  @override
  State<StatefulWidget> createState() => GeneticScreenState();
}

class GeneticScreenState extends State<GeneticScreen> {
  final _formKey = GlobalKey<FormState>();
  int a, b, c, d, res;
  var time;
  dynamic result;
  Widget build(BuildContext context) {
    return Form(
      key: _formKey,
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          Container(
            padding: EdgeInsets.all(8.0),
            child: Text(
              'a',
              style: TextStyle(fontSize: 20.0),
            ),
          ),
          Container(
            padding: EdgeInsets.all(8.0),
            child: TextFormField(
              validator: (value) {
                if (value == null) {
                  return null;
                }
                a = num.tryParse(value);
                if (a == null) {
                  return '"$value" is not a valid number';
                }
                return null;
              },
```

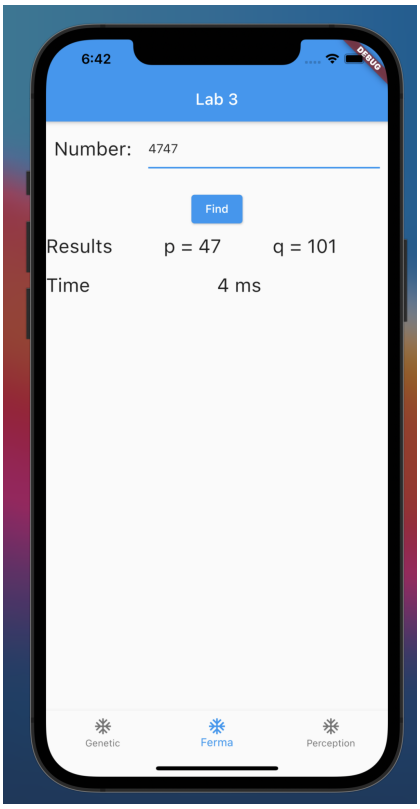
```
    keyboardType: TextInputType.number,
  ),
),
Container(
  padding: EdgeInsets.all(8.0),
  child: Text(
    'b',
    style: TextStyle(fontSize: 20.0),
  ),
),
Container(
  padding: EdgeInsets.all(8.0),
  child: TextFormField(
    validator: (value) {
      if (value == null) {
        return null;
      }
      b = num.tryParse(value);
      if (b == null) {
        return '"$value" is not a valid number';
      }
      return null;
    },
    keyboardType: TextInputType.number,
  ),
),
Container(
  padding: EdgeInsets.all(8.0),
  child: Text(
    'c',
    style: TextStyle(fontSize: 20.0),
  ),
),
Container(
  padding: EdgeInsets.all(8.0),
  child: TextFormField(
    validator: (value) {
      if (value == null) {
        return null;
      }
      c = num.tryParse(value);
      if (c == null) {
        return '"$value" is not a valid number';
      }
      return null;
    },
    keyboardType: TextInputType.number,
  ),
),
Container(
  padding: EdgeInsets.all(8.0),
  child: Text(
    'd',
    style: TextStyle(fontSize: 20.0),
  ),
```

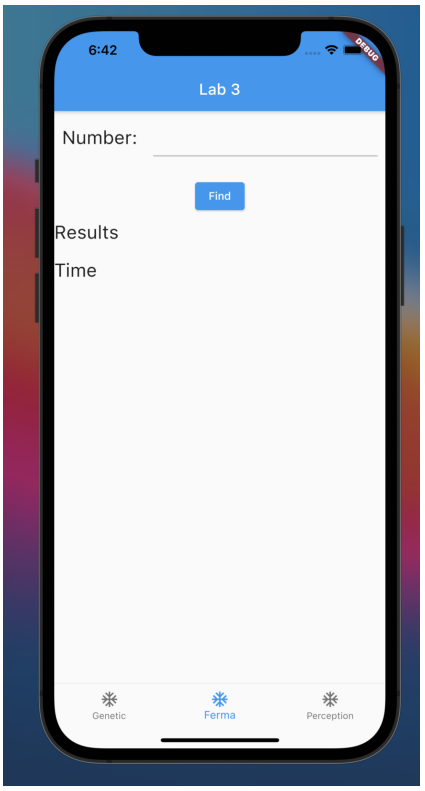
```
),
Container(
  padding: EdgeInsets.all(8.0),
  child: TextFormField(
    validator: (value) {
      if (value == null) {
        return null;
      }
      d = num.tryParse(value);
      if (d == null) {
        return '"$value" is not a valid number';
      }
      return null;
    },
    keyboardType: TextInputType.number,
  ),
),
Container(
  padding: EdgeInsets.all(8.0),
  child: Text(
    'y',
    style: TextStyle(fontSize: 20.0),
  ),
),
Container(
  padding: EdgeInsets.all(8.0),
  child: TextFormField(
    validator: (value) {
      if (value == null) {
        return null;
      }
      res = num.tryParse(value);
      if (res == null) {
        return '"$value" is not a valid number';
      }
      return null;
    },
    keyboardType: TextInputType.number,
  ),
),
Container(
  padding: EdgeInsets.all(8.0),
  child: ElevatedButton(
    onPressed: () {
      if (_formKey.currentState.validate()) {
        setState(
          () {
            var timer = Stopwatch();
            timer.start();
            result = solve([a, b, c, d, res], 2000, 4, 0.05);
            timer.stop();
            time = timer.elapsedMilliseconds;
            result = result[0];
          },
        );
```

```
            }
          },
          child: Text('Find'),
        ),
      ),
      Container(
        child: Text(
          'Results',
          style: TextStyle(fontSize: 20.0),
        ),
      ),
      SizedBox(height: 10.0),
      Container(
        child: Text(
          result == null ? '' : ' [x1, x2, x3, x4] = $result',
          style: TextStyle(fontSize: 20.0),
        ),
      ),
      SizedBox(height: 10.0),
      Container(
        child: Text(
          time == null ? '' : 'Time = $time ms',
          style: TextStyle(fontSize: 20.0),
        ),
      )
    ],
  ),
 );
 }
}
```

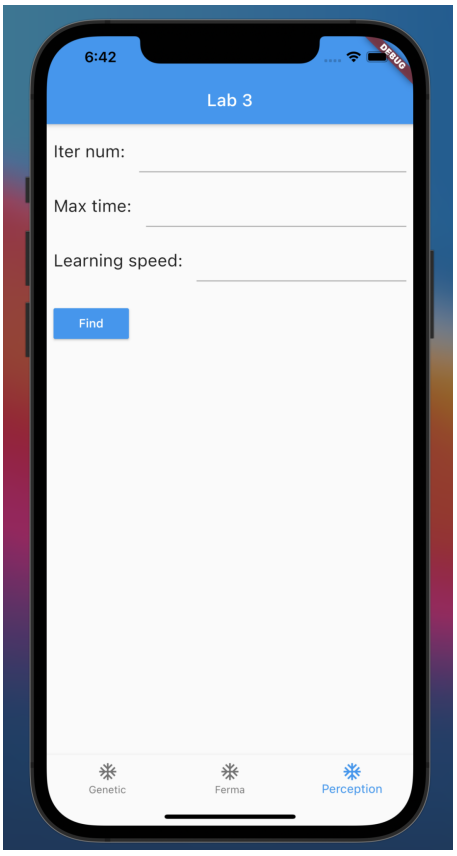
Отримані результати:

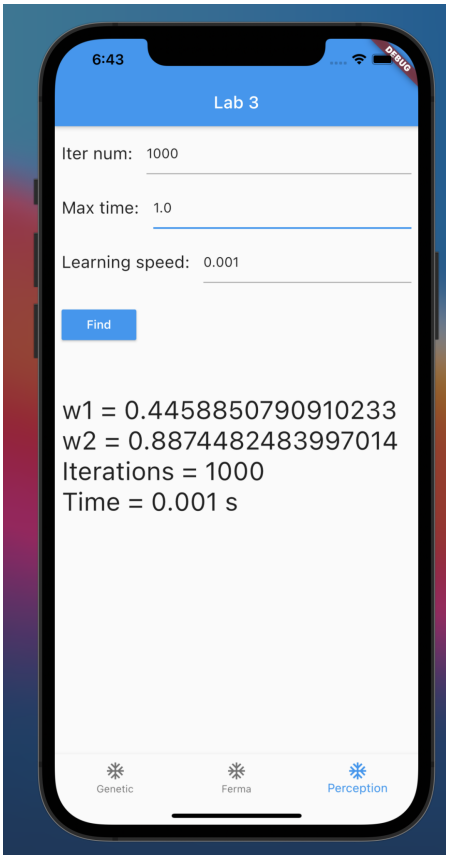
Lab 1


6:42
Lab 3
Number: 4747
Find
Results p = 47 q = 101
Time 4 ms
Genetic
Ferma
Perception

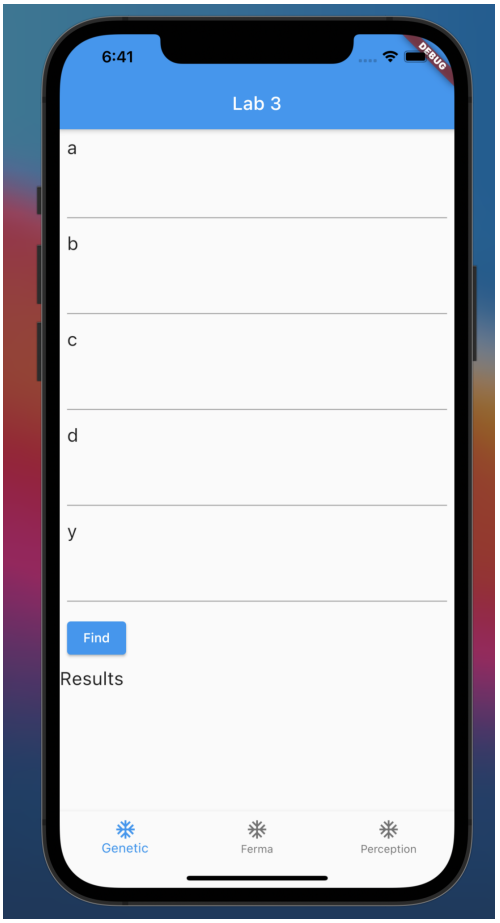

6:42
Lab 3
Number:
Find
Results
Time
Genetic
Ferma
Perception

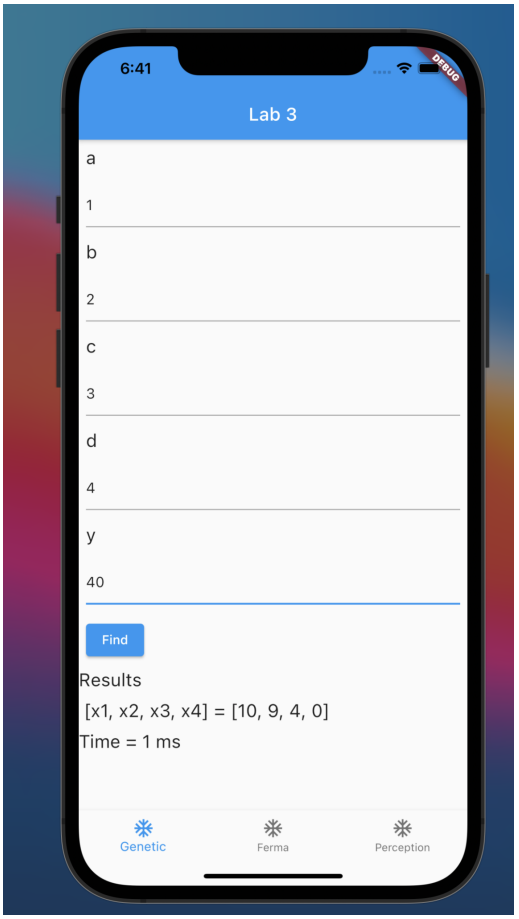
Lab 2


6:42
Lab 3
Iter num:
Max time:
Learning speed:
Find
Genetic
Ferma
Perception


6:43
Lab 3
Iter num: 1000
Max time: 1.0
Learning speed: 0.001
Find
w1 = 0.4458850790910233
w2 = 0.8874482483997014
Iterations = 1000
Time = 0.001 s
Genetic
Ferma
Perception

Lab 3





Висновки

Під час виконання лабораторних робіт 3.1, 3.2 та 3.3 ми ознайомились з основними принципами розкладання числа на прості множники з використанням різних алгоритмів факторизаці, з принципами машинного навчання за допомогою математичної моделі сприйняття інформації Перцептрон(Perceptron) та з принципами реалізації генетичного алгоритму, вивчення та дослідження особливостей даного алгоритму з використанням засобів моделювання і сучасних програмних оболонок.