ГУАП

КАФЕДРА № 42

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

| ст. преподаватель |  |  |  | В.А. Миклуш |
| --- | --- | --- | --- | --- |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №1 |
| --- |
|  |
| по курсу: Теория информации, данные, знания |
|  |

РАБОТУ ВЫПОЛНИЛ

| СТУДЕНТ ГР. № | 4128 |  |  |  | В. А. Воробьев |
| --- | --- | --- | --- | --- | --- |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2023

***Цель работы***: Произвести вычисление ряда параметров определенной дискретной конечнозначной величины *X.*

**Задание**

1. Найти недостающую вероятность, построить многоугольник распределения.
2. Найти функцию распределения и построить её график.
3. Вычислить МО, моду, медиану, начальные и центральные моменты до 4-го порядка включительно, дисперсию, СКО, асимметрию и эксцесс.
4. Найти *P*(*X*≤*x*0).

Вариант задания №5

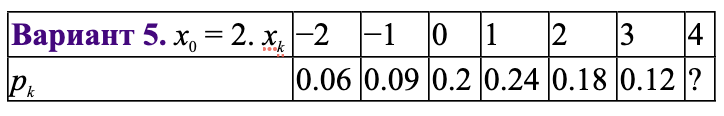


Рисунок 1 – Исходные данные

**Ход работы**

В ходе работы была создана программа на языке Dart, выполняющая все поставленные задача. Исходный код и визуальный интерфейс программы представлены в Приложении.

* 1. Построение многоугольника распределения

Производим поиск недостающей вероятности.

**P7**= 1 – (p1 + p2 + p3 + p4 + p5 + p6) = 1 – (0.06+ 0.09 + 0.2+ 0.24+ 0.18+ 0.12) = 1 – 0.89 = **0.11**

На основе имеющихся и полученных данных строим многоугольник распределения, на рисунке 2.



Рисунок 2 - Многоугольник распределения в созданной программе

* 1. Построение функции распределения

Находим значения функции распределения на основе имеющихся данных вероятности.

На основе полученных данных строим график функции, на рисунке 3.

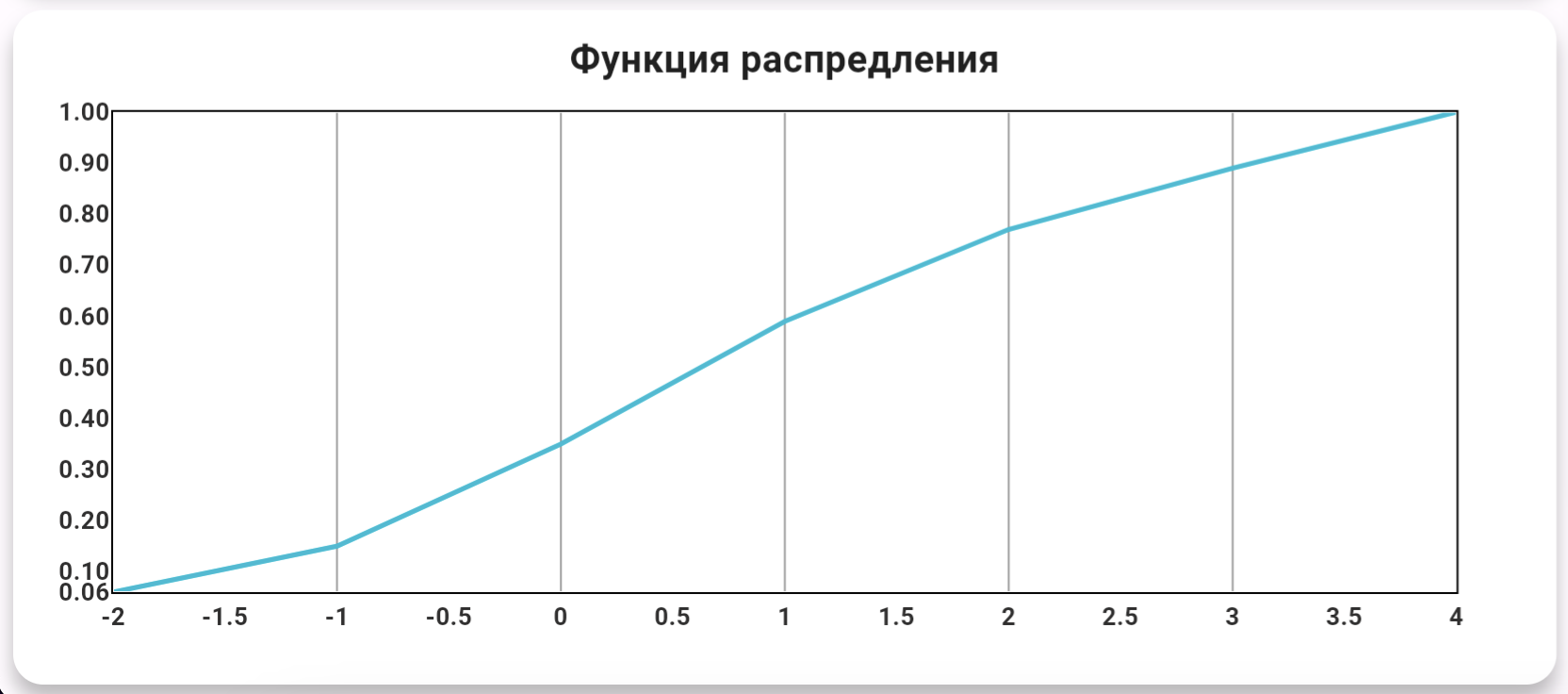


Рисунок 3 - График функции распределения в созданной программе

* 1. Вычисление статистических характеристик

*Математическое ожидание* (среднее значение случайной величины) = 1.19

*Мода* (наиболее часто встречающееся значение) = 1

*Медиана* (серединное значение набора чисел) = 1

*Дисперсия* (отражает меру разброса данных вокруг средней арифметической) = 2,7139

*Среднеквадратичное отклонение* (статистическая характеристика распределения случайной величины, показывающая среднюю степень разброса значений величины относительно математического ожидания) = 1,64739

*Асимметрия* (мера несимметричности распределения случайной величины) = 0,364

*Эксцесс* = 0,0005

Начальные моменты (математическое ожидание k-й степени случайной величины x):

*Первого порядка* = 1,19

*Второго порядка* = 4,13

*Третьего порядка* = 11,39

*Четвертого порядка* = 42,05

Центральные моменты (математическое ожидание величины   
(*X-M(X))k*):

*Первого порядка* = 0

*Второго порядка* = 2,7139

*Третьего порядка* = 0,016

*Четвертого порядка* = 16,908

Результаты построенных графиков и расчетов статистических характеристик представлены на рисунке 4.

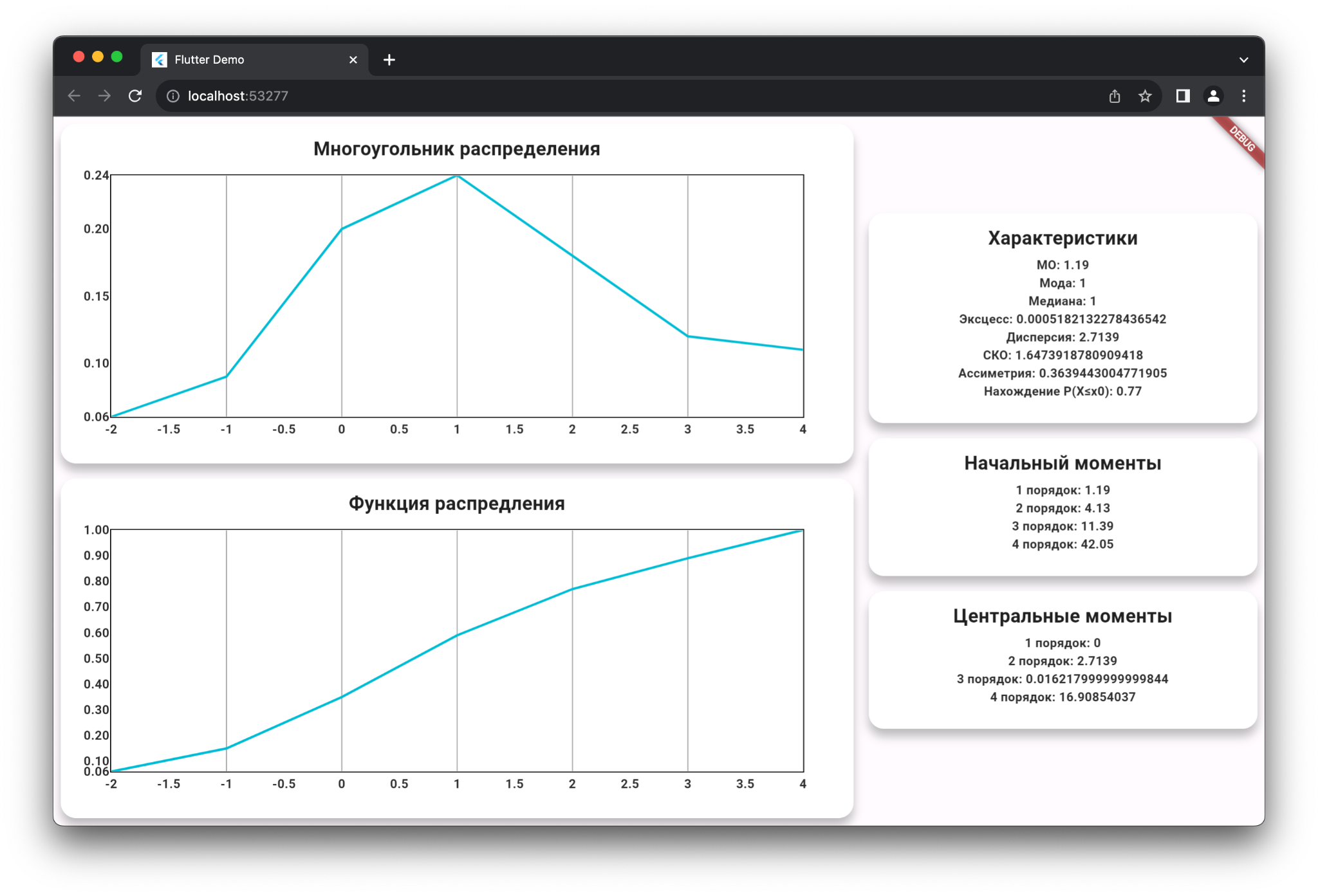


Рисунок 4 – Статистические характеристики в созданной программе

ВЫВОД

В ходе выполнения лабораторной работы, на основе исходных данных, соответствующих варианту с использованием языка программирования Dart и Flutter, была создана программа, вычисляющая основные статистические значения, а также построены многоугольник и функция распределения случайной величины.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД

main.dart

import 'package:flutter/material.dart';

import 'src/ui/preview\_app.dart';

void main() {

runApp(const MyApp());

}

class MyApp extends StatelessWidget {

const MyApp({super.key});

@override

Widget build(BuildContext context) {

return MaterialApp(

title: 'Flutter Demo',

theme: ThemeData(

colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),

useMaterial3: true,

),

home: Scaffold(body: const PreviewApp()),

);

}

}

preview\_app.dart

import 'package:extend\_math/extend\_math.dart';

import 'package:flutter/material.dart';

import 'package:lab1/src/logic/variant.dart';

import 'package:ui\_kit/ui\_kit.dart';

class PreviewApp extends StatelessWidget {

const PreviewApp({Key? key}) : super(key: key);

@override

Widget build(BuildContext context) {

return Row(

children: [

Expanded(

flex: 2,

child: KitColumn(

childFit: FlexFit.tight,

children: [

KitTitleContainer(

title: "Многоугольник распределения",

child: KitLineChart(

lines: [

KitLineData(

curved: false,

dots: Variant.distributionMap.entries

.map(

(pair) => KitDot(pair.key.toDouble(), pair.value))

.toList(),

),

],

),

),

KitTitleContainer(

title: "Функция распредления",

child: KitLineChart(

lines: [

KitLineData(

curved: false,

dots: Variant.distributionMap.cumulativeDistribution

.map((pair) =>

KitDot(pair.x.toDouble(), pair.y.toDouble()))

.toList(),

),

],

),

),

],

),

),

Expanded(

child: KitColumn(

mainAxisSize: MainAxisSize.min,

childFit: FlexFit.loose,

children: [

KitTitleContainer(

title: "Характеристики",

child: Column(

mainAxisSize: MainAxisSize.min,

children: [

KitText.system("МО: ${Variant.distributionMap.mean}"),

KitText.system("Мода: ${Variant.distributionMap.mode}"),

KitText.system(

"Медиана: ${Variant.distributionMap.median}"),

KitText.system(

"Эксцесс: ${Variant.distributionMap.excess}"),

KitText.system(

"Дисперсия: ${Variant.distributionMap.variance}"),

KitText.system(

"СКО: ${Variant.distributionMap.standardDeviation}"),

KitText.system(

"Ассиметрия: ${Variant.distributionMap.skewness}"),

KitText.system(

"Нахождение P(X≤x0): ${Variant.distributionMap.calcCumulativeProbability(-2)}"),

],

),

),

KitTitleContainer(

title: "Начальный моменты",

child: Column(

mainAxisSize: MainAxisSize.min,

children: [

KitText.system(

"1 порядок: ${Variant.distributionMap.mean}"),

KitText.system(

"2 порядок: ${Variant.distributionMap.secondMoment}"),

KitText.system(

"3 порядок: ${Variant.distributionMap.thirdMoment}"),

KitText.system(

"4 порядок: ${Variant.distributionMap.fourthMoment}"),

],

),

),

KitTitleContainer(

title: "Центральные моменты",

child: Column(

mainAxisSize: MainAxisSize.min,

children: [

KitText.system("1 порядок: 0"),

KitText.system(

"2 порядок: ${Variant.distributionMap.centralSecondMoment}"),

KitText.system(

"3 порядок: ${Variant.distributionMap.centralThirdMoment}"),

KitText.system(

"4 порядок: ${Variant.distributionMap.centralFourthMoment}"),

],

),

),

],

),

),

],

);

}

}

variant.dart

abstract final class Variant{

static final distributionMap = {

-2.0 : 0.06,

-1.0 : 0.09,

0.0 : 0.2,

1.0 : 0.24,

2.0 : 0.18,

3.0: 0.12,

4.0 : 0.11,

};

}

extend\_math.dart

library extend\_math;

export 'src/extension/amplitude\_spectrum\_ext.dart';

export 'src/extension/distribution\_map\_ext.dart';

export 'src/extension/fft\_extension.dart';

export 'src/extension/math\_interval\_ext.dart';

export 'src/extension/sprectrum\_energy\_ext.dart';

export 'src/logic/list\_functions.dart';

export 'src/models/point2.dart';

export 'src/models/math\_interval.dart';

distribution\_map\_ext.dart

import 'dart:core';

import 'dart:math';

import '../models/point2.dart';

extension DistributionMapStatistics on Map<double, double> {

List<Point2> get cumulativeDistribution {

final listEntries = entries.toList();

final res = <Point2>[];

var cumulative = listEntries.first.value;

res.add(Point2(listEntries.first.key, cumulative));

for (int i = 1; i < listEntries.length; i++) {

cumulative = (cumulative + listEntries[i].value);

res.add(Point2(listEntries[i].key, cumulative));

}

return res;

}

double calcCumulativeProbability(double x0) {

double cumulativeProbability = 0.0;

forEach((key, value) {

if (key <= x0) {

cumulativeProbability += value;

}

});

return cumulativeProbability;

}

double get mean {

double mean = 0.0;

forEach((key, value) {

mean += key \* value;

});

return mean;

}

double get secondMoment {

double secondMoment = 0.0;

forEach((key, value) {

secondMoment += (key \* key) \* value;

});

return secondMoment;

}

double get thirdMoment {

double thirdMoment = 0.0;

forEach((key, value) {

thirdMoment += (key \* key \* key) \* value;

});

return thirdMoment;

}

double get fourthMoment {

double fourthMoment = 0.0;

forEach((key, value) {

fourthMoment += (key \* key \* key \* key) \* value;

});

return fourthMoment;

}

double get mode {

double mode = entries.first.key;

double maxProbability = entries.first.value;

// Пройдитесь по всем парам ключ-значение в вашей вероятностной карте

forEach((key, probability) {

if (probability > maxProbability) {

mode = key;

maxProbability = probability;

}

});

return mode;

}

double get median {

final sortedEntries = entries.toList()

..sort((a, b) => a.key.compareTo(b.key));

final numEntries = sortedEntries.length;

if (numEntries % 2 == 0) {

final middle1 = sortedEntries[numEntries ~/ 2 - 1].key;

final middle2 = sortedEntries[numEntries ~/ 2].key;

return (middle1 + middle2) / 2.0;

} else {

return sortedEntries[numEntries ~/ 2].key.toDouble();

}

}

double get excess {

double mean = 0.0;

double variance = 0.0;

forEach((key, value) {

mean += key \* value;

});

forEach((key, value) {

variance += (key - mean) \* (key - mean) \* value;

});

final stdDev = sqrt(variance);

final numEntries = length.toDouble();

double excess = 0.0;

forEach((key, value) {

excess += ((key - mean) \* (key - mean) \* (key - mean) \* value) /

(stdDev \* stdDev \* stdDev);

});

return excess / numEntries;

}

double get variance {

double mean = 0.0;

double variance = 0.0;

forEach((key, value) {

mean += key \* value;

});

forEach((key, value) {

variance += ((key - mean) \* (key - mean)) \* value;

});

return variance;

}

double get standardDeviation => sqrt(variance);

double get skewness {

double thirdMoment = this.thirdMoment;

final stdDev = standardDeviation;

final numEntries = length.toDouble();

double skewness = thirdMoment / (stdDev \* stdDev \* stdDev \* numEntries);

return skewness;

}

double get centralSecondMoment {

double centralSecondMoment = 0.0;

forEach((key, value) {

centralSecondMoment += ((key - mean) \* (key - mean)) \* value;

});

return centralSecondMoment;

}

double get centralThirdMoment {

double centralThirdMoment = 0.0;

forEach((key, value) {

centralThirdMoment +=

((key - mean) \* (key - mean) \* (key - mean)) \* value;

});

return centralThirdMoment;

}

double get centralFourthMoment {

double centralFourthMoment = 0.0;

forEach((key, value) {

centralFourthMoment +=

((key - mean) \* (key - mean) \* (key - mean) \* (key - mean)) \* value;

});

return centralFourthMoment;

}

}

sprectrum\_energy\_ext.dart

import 'dart:math';

import 'package:extend\_math/extend\_math.dart';

extension SpectrumAmplEnergyExt on List<double> {

double get energy {

final total = sum(map((e) => e \* e));

final normalize = map((e) => e \* sqrt(0.5 / total));

return sum(normalize.map((e) => e \* e));

}

}

extension SpectrumPointEnergyExt on List<Point2> {

double calculateEnergy(MathInterval interval) {

double integral = 0;

for (final point in this) {

integral += pow(point.y, 2);

}

final energy = integral / interval.length;

return energy;

}

}

math\_interval\_ext.dart

import 'package:extend\_math/extend\_math.dart';

import '../utils/typedefs.dart';

extension MathIntervalExt on MathInterval {

List<Point2> applyFx(Func1 fx, {required double step}) {

final count = length ~/ step;

return [

for (var x = start; x <= end; x += length / count) Point2(x, fx(x))

];

}

}

amplitude\_spectrum\_ext.dart

import 'dart:math';

import '../models/point2.dart';

extension AmplitudeSpectrumExtension on List<Point2> {

double amplitudeSpectrumFor(

double freq, {

required double step,

}) {

double realPart = 0.0;

double imagPart = 0.0;

for (int j = 0; j < length; j++) {

double value = this[j].y;

double angle = 2 \* pi \* freq \* this[j].x;

realPart += value \* cos(angle) \* step;

imagPart += value \* sin(angle) \* step;

}

return sqrt(realPart \* realPart + imagPart \* imagPart);

}

}

fft\_extension.dart

// ignore\_for\_file: prefer\_const\_constructors

import 'dart:math';

import 'package:complex/complex.dart';

import '../models/point2.dart';

extension DFTExtension on List<Point2> {

List<Complex> get dft {

int N = length;

List<Complex> dftResult = List<Complex>.generate(N, (i) {

Complex sum = const Complex(0.0, 0.0);

for (int j = 0; j < N; j++) {

double angle = 2 \* pi \* i \* j / N;

Complex c = Complex.polar(this[j].y, angle);

sum += c;

}

return sum;

});

return dftResult;

}

}

extension InverseDFTExtension on List<Complex> {

List<Point2> get inverseDft {

final spectrum = this;

int N = spectrum.length;

List<Point2> signal = List<Point2>.generate(N, (i) {

Complex sum = Complex(0.0, 0.0);

for (int j = 0; j < N; j++) {

double angle = -2 \* pi \* i \* j / N;

Complex c = spectrum[j] \* Complex.polar(1.0, angle);

sum += c;

}

return Point2(i.toDouble(), sum.real / N);

});

return signal;

}

}

list\_functions.dart

double sum(Iterable<double> list) =>

list.reduce((value, element) => value + element);

List<T> roll<T>(List<T> inputList, int shiftAmount) {

final length = inputList.length;

if (length == 0) {

return inputList;

}

// Calculate the effective shift amount, wrapping around if necessary

final effectiveShift = shiftAmount % length;

if (effectiveShift == 0) {

return inputList;

}

// Split the input list into two parts and rejoin them with the shift

final startIndex = effectiveShift < 0 ? -effectiveShift : length - effectiveShift;

final part1 = inputList.sublist(startIndex);

final part2 = inputList.sublist(0, startIndex);

return [...part1, ...part2];

}

typedefs.dart

typedef Func1 = double Function(double x);

math\_interval.dart

final class MathInterval {

final double start;

final double end;

const MathInterval(this.start, this.end);

double get length => (end - start).abs();

}

point2.dart

class Point2 {

final double x;

final double y;

const Point2(this.x, this.y);

static const zero = Point2(0, 0);

}