ГУАП

КАФЕДРА № 42

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ПРЕПОДАВАТЕЛЬ

| Ассистент |  |  |  | Н.А. Янковский |
| --- | --- | --- | --- | --- |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №3 |
| --- |
| **Дискретные сигналы. ДПФ.**  Вариант 5 |
|  |
| по курсу: Цифровая обработка и передача сигналов |
|  |

РАБОТУ ВЫПОЛНИЛ

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

Санкт-Петербург 2023

**1 Задание**

Исходные данные:

*f =* 3*N, T =* 10*/F*, где *N* – номер по списку.

Написать программу, которая позволит:

1. Провести дискретизацию функции *u(t) =* sin(2π*ft*) на заданном интервале с частотой дискретизации 3*f*.
2. Вычислить прямое и обратное дискретное преобразование Фурье исследуемой функции.
3. Продемонстрировать с помощью написанной программы свойства линейности, сдвига сигнала во времени и равенство Парсеваля.

**2 Выполнение работы**

Построим график функции, а также выполним для функции дискретное преобразование Фурье и затем обратное дискретное преобразование Фурье.

Покажем свойство линейности, сдвига и Парсеваля.

Результат изображен на рис. 1

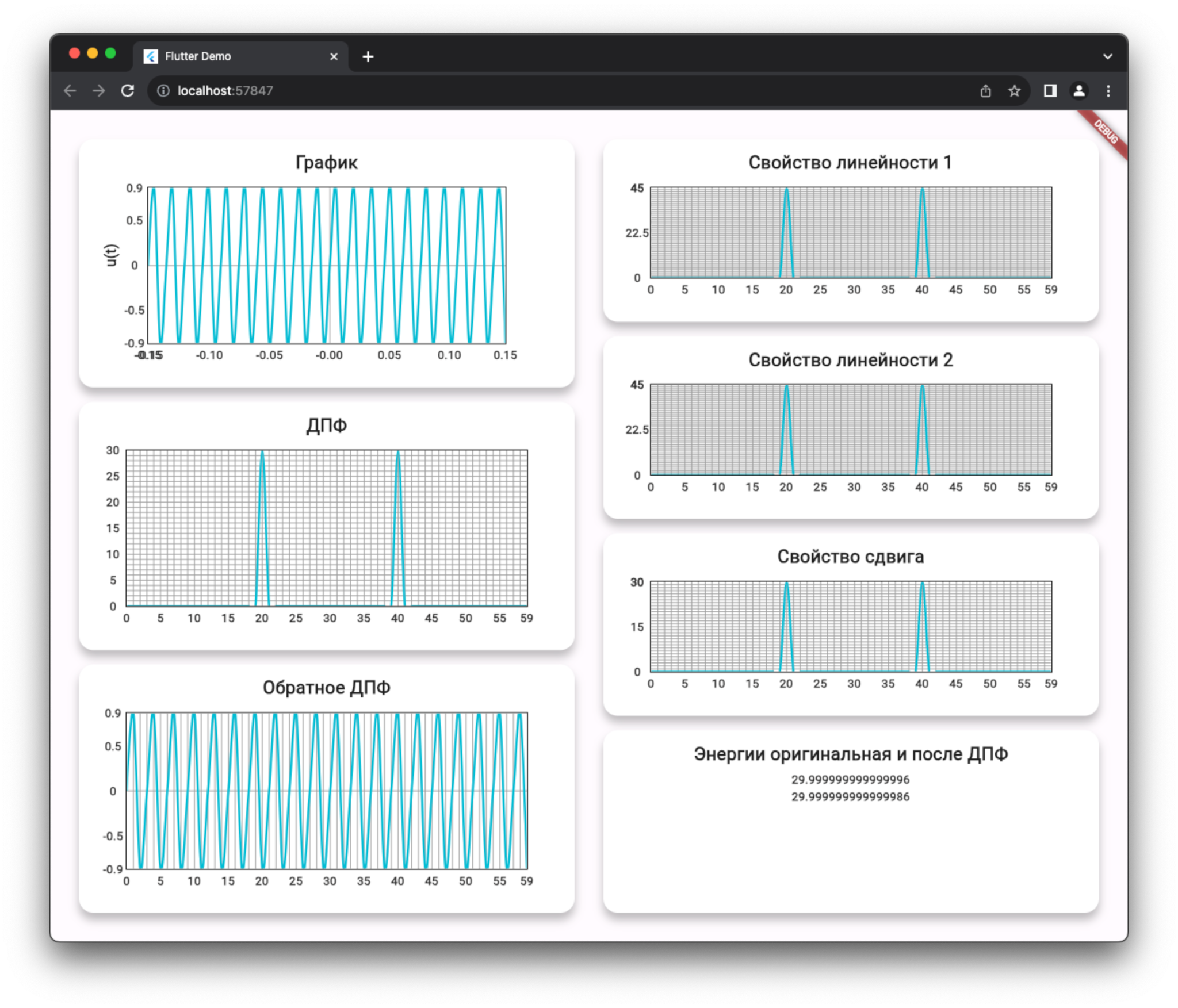


Рисунок 1 – Результат работы программы

**3 Вывод**

В ходе выполнения лабораторной работы мы приобрели практические навыки вычисления и визуализации математических функций, эти навыки в дальнейшем могут быть полезны в анализе данных, обработке сигналов, а также в других областях, где важно понимание и работа с математическими функциями.

ПРИЛОЖЕНИЕ

preview\_app.dart

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

import 'package:flutter/widgets.dart';

import 'package:lab3/logic/math\_calculations.dart';

import 'package:lab3/logic/variant.dart';

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

class PreviewApp extends StatelessWidget {

const PreviewApp({super.key});

@override

Widget build(BuildContext context) {

return Padding(

padding: const EdgeInsets.all(16),

child: Row(

children: [

Expanded(

child: Padding(

padding: const EdgeInsets.all(16),

child: Column(

children: [

Expanded(

child: KitTitleContainer(

title: 'Граффик',

child: KitLineChart(

yAxisName: 'u(t)',

lines: [

KitLineData(

dots: Variant.fxDots

.map((e) => KitDot(e.x, e.y))

.toList(),

),

],

),

),

),

const SizedBox(height: 16),

Expanded(

child: KitTitleContainer(

title: 'ДПФ',

child: KitLineChart(

lines: [

KitLineData(

dots: Variant.fxDots.dft

.asMap()

.entries

.map((e) =>

KitDot(e.key.toDouble(), e.value.abs()))

.toList(),

),

],

),

),

),

const SizedBox(height: 16),

Expanded(

child: KitTitleContainer(

title: 'Обратное ДПФ',

child: KitLineChart(

lines: [

KitLineData(

dots: Variant.fxDots.dft.inverseDft

.map((e) => KitDot(e.x, e.y))

.toList(),

),

],

),

),

),

],

),

),

),

Expanded(

child: Padding(

padding: const EdgeInsets.all(16),

child: Column(

children: [

Expanded(

child: KitTitleContainer(

title: 'Свойство линейности 1',

child: KitLineChart(

lines: [

KitLineData(

dots: MathCalculations.fDotsUnion

.map((e) => KitDot(e.x, e.y))

.toList(),

),

],

),

),

),

const SizedBox(height: 16),

Expanded(

child: KitTitleContainer(

title: 'Свойство линейности 2',

child: KitLineChart(

lines: [

KitLineData(

dots: MathCalculations.fDotsSum

.map((e) => KitDot(e.x, e.y))

.toList(),

),

],

),

),

),

const SizedBox(height: 16),

Expanded(

child: KitTitleContainer(

title: 'Свойство сдвига',

child: KitLineChart(

lines: [

KitLineData(

dots: MathCalculations.shifted

.map((e) => KitDot(e.x, e.y))

.toList(),

),

],

),

),

),

const SizedBox(height: 16),

Expanded(

child: KitTitleContainer(

title: 'Энергии оригинальная и после ДПФ',

child: Center(

child: Column(

children: [

KitText.system(

MathCalculations.originalEnergy.toString()),

KitText.system(

MathCalculations.transformedEnergy.toString(),

),

],

),

),

),

),

],

),

),

),

],

),

);

}

}

variant.dart

import 'dart:math';

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

abstract final class Variant {

static const n = 22;

static const f = 3 \* n;

static const t = 10 / f;

static const interval = MathInterval(-t, t);

static const step = 1 / 3 / f;

static final fxDots = interval.applyFx(fx, step: step);

static double fx(double x) => sin(2 \* pi \* f \* x);

}

math\_calculations.dart

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

import 'package:lab3/logic/variant.dart';

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

abstract final class MathCalculations {

static final f1Dots = Variant.interval.applyFx(

Variant.fx,

step: Variant.step,

);

static final f2Dots = Variant.interval.applyFx(\_fx2, step: Variant.step);

static List<Point2> get fDotsUnion {

final dots = Variant.interval.applyFx(

(x) => Variant.fx(x) + \_fx2(x),

step: Variant.step,

);

return dots.dft

.asMap()

.entries

.map((e) => Point2(e.key.toDouble(), e.value.abs()))

.toList();

}

static List<Point2> get fDotsSum {

final f1Dft = f1Dots.dft;

final f2Dft = f2Dots.dft;

return [

for (int i = 0; i < f1Dft.length; i++)

Point2(

i.toDouble(),

(f1Dft[i] + f2Dft[i]).abs(),

),

];

}

static List<Point2> get shifted => roll(f1Dots, 32)

.dft

.asMap()

.entries

.map(

(e) => Point2(

e.key.toDouble(),

e.value.abs(),

),

)

.toList();

static double get originalEnergy {

return sum(f1Dots.map((e) => e.y \* e.y));

}

static double get transformedEnergy {

return sum(f1Dots.dft.map((e) => e.abs() \* e.abs())) / f1Dots.dft.length;

}

static double \_fx2(double x) {

return Variant.fx(x) \* 0.5;

}

}

main.dart

import 'package:flutter/material.dart';

import 'package:lab3/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: const Scaffold(

body: PreviewApp(),

),

);

}

}

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>[Point2(listEntries.first.key - 1, 0)];

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));

}

res.add(Point2(listEntries.last.key + 1, 1));

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 maxProbability = -1.0;

double mode = 0;

forEach((key, value) {

if (value > maxProbability) {

maxProbability = value;

mode = key;

}

});

return mode.toDouble();

}

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);

}