ГУАП

КАФЕДРА № 42

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ _____
ПРЕПОДАВАТЕЛЬ

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

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №4

Дискретные сигналы. БПФ.

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

f = 3N, T = 10/F, где N – номер по списку. Написать программу, которая позволит:

1. Провести дискретизацию функции $u(t) = \sin(2\pi ft)$ на заданном интервале с частотой дискретизации 3f.
2. Вычислить прямое и обратное быстрое преобразование Фурье исследуемой функции.
3. Произвести декодирование аудио-файла с записью тонального сигнала (Dual-Tone Multi-Frequency (DTMF)) сигнала в формате WAV PCM 16 bit, mono. Данный способ кодирования предполагает, что кодируемое значение представляется в виде пары различных частот (f1/f2) в соответствии с приведенной таблицей 4.1. Затем, сигнал представляется в виде отсчетов суммы двух синусоид соответствующих частот. Для декодирования сигнала необходимо произвести прямое быстрое преобразование Фурье для имеющегося набора отсчетов и определить частоты используемых

2 Выполнение работы

Построим график функции и реализуем быстрое преобразование Фурье и обратное к нему.

Результат представлен на рис. 1.

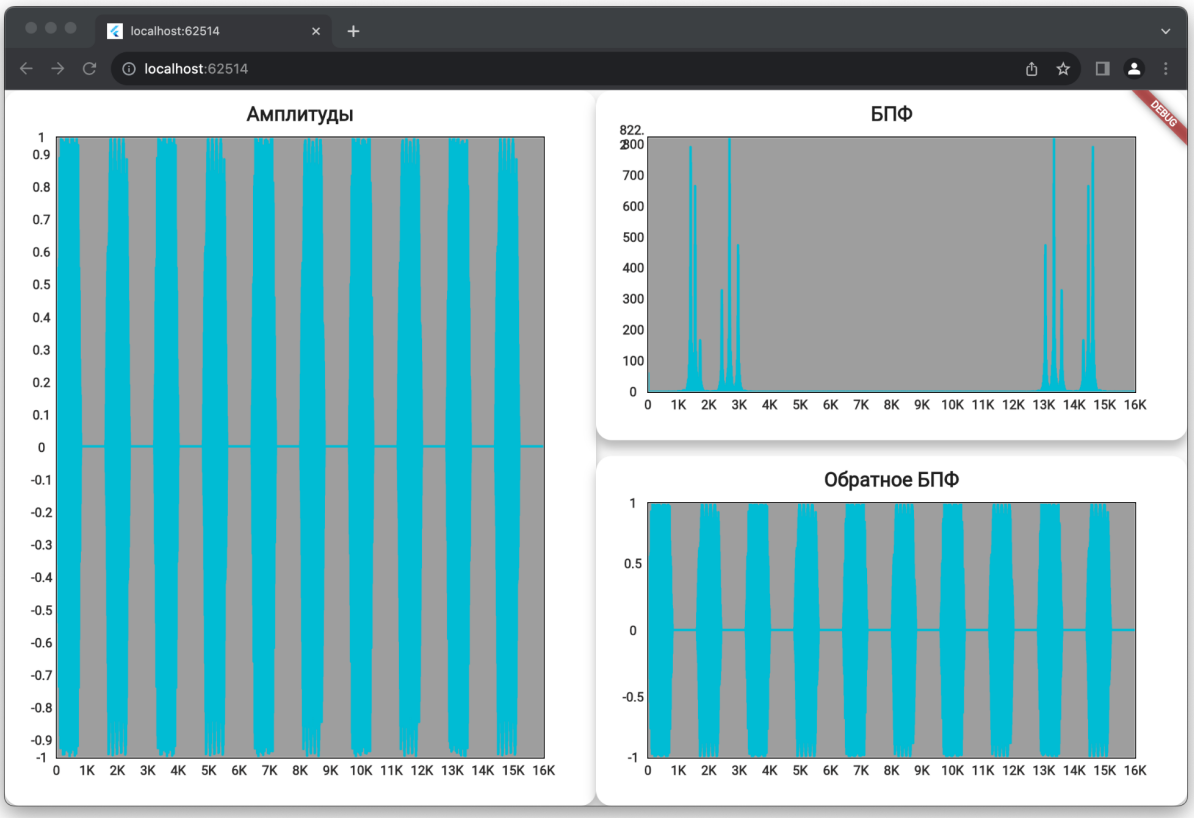

Рисунок 1 – Результат работы программы

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.

preview_app.dart

```
import 'package:extend_math/extend_math.dart';
import 'package:flutter/widgets.dart';
import 'package:lab4/logic/calculations.dart';
import 'package:ui_kit/ui_kit.dart';

class PreviewApp extends StatelessWidget {
  const PreviewApp({super.key});

  @override
  Widget build(BuildContext context) {
    return Row(
      children: [
        Expanded(
          child: Column(
            children: [
              Expanded(
                child: KitTitleContainer(
                  title: 'Амплитуды',
                  child: KitLineChart(
                    lines: [
                      KitLineData(
                        dots: Calculations.points
                            .map((e) => KitDot(e.x, e.y))
                            .toList(),
                      ),
                    ],
                  ),
                ),
              ),
```

```
    ],
  ),
),
Expanded(
  child: Column(
    children: [
      Expanded(
        child: KitTitleContainer(
          title: 'БПФ',
          child: KitLineChart(
            lines: [
              KitLineData(
                dots: Calculations.bft
                    .map((e) => KitDot(e.x, e.y))
                    .toList(),
              ),
            ],
          ),
        ),
      ),
      const SizedBox(height: 16),
      Expanded(
        child: KitTitleContainer(
          title: 'Обратное БПФ',
          child: KitLineChart(
            lines: [
              KitLineData(
                dots: Calculations.points.bft.inverseBft
                    .map((e) => KitDot(e.x, e.y))
                    .toList(),
              ),
```

```
                ],
              ),
            ),
          ),
        ],
      ),
    )
  ],
 );
 }
}
```

variant.dart

```
abstract final class Variant {
  static const String filePath = 'test1.wav';
}
```

audio.dart

```
import 'package:flutter/services.dart';
import 'package:wav/wav_file.dart';

Future<List<double>> extractAudioData(String filePath) async {
  final data = await rootBundle.load('assets/test1.wav');
  final bytes = data.buffer.asUint8List();
  final waveFile = Wav.read(bytes);
  final List<double> audioData = waveFile.toMono().toList();
  return audioData;
}
```

calculations.dart

```
import 'package:complex/complex.dart';
```

```
import 'package:extend_math/extend_math.dart';

abstract final class Calculations {
  // Yep, its not good but fast!
  static late List<double> readedAudio;

  static List<Point2> get points {
    return readedAudio
        .asMap()
        .entries
        .map((e) => Point2(e.key.toDouble(), e.value))
        .toList();
  }

  static List<Point2>? cachedDft;
  static List<Point2> get dft {
    if (cachedDft != null) return cachedDft!;
    cachedDft = points.dft
        .asMap()
        .entries
        .map((e) => Point2(e.key.toDouble(), e.value.abs()))
        .toList();
    return cachedDft!;
  }

  static List<String> get match {
    final spectralComponents = points.dft;
    // Определите пороги для определения наличия частот в спектре
    double magnitudeThreshold = 200.0; // Подстройте подходящий порог
    double frequencyThreshold = 150.0; // Подстройте подходящий порог
```

```
List<String> dtmfCodes = [];

for (int i = 1; i < spectralComponents.length - 1; i++) {
  Complex current = spectralComponents[i];

  double magnitude = current.abs();
  double frequency = i * 16000 / spectralComponents.length;

  if (magnitude > magnitudeThreshold) {
    if ((frequency % 697) < frequencyThreshold ||
        (frequency % 1209) < frequencyThreshold) {
      dtmfCodes.add('1');
    } else if ((frequency % 697) < frequencyThreshold ||
        (frequency % 1336) < frequencyThreshold) {
      dtmfCodes.add('2');
    } else if ((frequency % 697) < frequencyThreshold ||
        (frequency % 1477) < frequencyThreshold) {
      dtmfCodes.add('3');
    } else if ((frequency % 697) < frequencyThreshold ||
        (frequency % 1633) < frequencyThreshold) {
      dtmfCodes.add('A');
    } else if ((frequency % 770) < frequencyThreshold ||
        (frequency % 1209) < frequencyThreshold) {
      dtmfCodes.add('4');
    } else if ((frequency % 770) < frequencyThreshold ||
        (frequency % 1336) < frequencyThreshold) {
      dtmfCodes.add('5');
    } else if ((frequency % 770) < frequencyThreshold ||
        (frequency % 1477) < frequencyThreshold) {
      dtmfCodes.add('6');
    } else if ((frequency % 770) < frequencyThreshold ||
```

```
        (frequency % 1633) < frequencyThreshold) {
      dtmfCodes.add('B');
    } else if ((frequency % 852) < frequencyThreshold ||
        (frequency % 1209) < frequencyThreshold) {
      dtmfCodes.add('7');
    } else if ((frequency % 852) < frequencyThreshold ||
        (frequency % 1336) < frequencyThreshold) {
      dtmfCodes.add('8');
    } else if ((frequency % 852) < frequencyThreshold ||
        (frequency % 1477) < frequencyThreshold) {
      dtmfCodes.add('9');
    } else if ((frequency % 852) < frequencyThreshold ||
        (frequency % 1633) < frequencyThreshold) {
      dtmfCodes.add('C');
    } else if ((frequency % 941) < frequencyThreshold ||
        (frequency % 1209) < frequencyThreshold) {
      dtmfCodes.add('*');
    } else if ((frequency % 941) < frequencyThreshold ||
        (frequency % 1336) < frequencyThreshold) {
      dtmfCodes.add('0');
    } else if ((frequency % 941) < frequencyThreshold ||
        (frequency % 1477) < frequencyThreshold) {
      dtmfCodes.add('#');
    } else if ((frequency % 941) < frequencyThreshold ||
        (frequency % 1633) < frequencyThreshold) {
      dtmfCodes.add('D');
    }
  }
}

return dtmfCodes.toList();
```

```
}

static List<int> findFrequencyPeaks(
    List<Complex> spectralComponents, int sampleRate) {
  List<int> peaks = [];
  double threshold =
      200.0; // Установите подходящий порог для определения пиков

  for (int i = 1; i < spectralComponents.length - 1; i++) {
    double magnitude = spectralComponents[i].abs();
    if (magnitude > threshold &&
        magnitude > spectralComponents[i - 1].abs() &&
        magnitude > spectralComponents[i + 1].abs()) {
      double frequency = i * sampleRate / spectralComponents.length;
      peaks.add(frequency.round());
    }
  }

  return peaks;
}

static List<String> matchFrequenciesToDTMF(List<int> frequencies) {
  final dtmfFrequencies = {
    697 / 1209: '1',
    697 / 1336: '2',
    697 / 1477: '3',
    697 / 1633: 'A',
    770 / 1209: '4',
    770 / 1336: '5',
    770 / 1477: '6',
    770 / 1633: 'B',
```

```
    852 / 1209: '7',
    852 / 1336: '8',
    852 / 1477: '9',
    852 / 1633: 'C',
    941 / 1209: '*',
    941 / 1336: '0',
    941 / 1477: '#',
    941 / 1633: 'D'
  };

  List<String> dtmfCodes = [];
  for (int frequency in frequencies) {
    String closestFrequency = findClosestFrequency(
      frequency,
      dtmfFrequencies.keys.toList(),
    ).toString();
    dtmfCodes.add(dtmfFrequencies[closestFrequency] ?? "???");
  }

  return dtmfCodes;
}

static double findClosestFrequency(
    int targetFrequency, List<double> frequencies) {
  var minDifference = 99999999.0;
  late double closestFrequency;

  for (var frequency in frequencies) {
    var difference = (frequency - targetFrequency).abs();
    if (difference < minDifference) {
      minDifference = difference;
```

```
        closestFrequency = frequency;
      }
    }

    return closestFrequency;
  }
}
```

main.dart

```
import 'package:flutter/material.dart';
import 'package:lab4/logic/audio.dart';
import 'package:lab4/logic/calculations.dart';
import 'package:lab4/logic/variant.dart';
import 'package:lab4/ui/preview_app.dart';

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  Calculations.readedAudio = await extractAudioData(Variant.filePath);
  print(Calculations.match);
  runApp(const MainApp());
}

class MainApp extends StatelessWidget {
  const MainApp({super.key});

  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      home: Scaffold(
        body: Center(
          child: 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);
}
```