Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ имени Н.Э.БАУМАНА
(национальный исследовательский университет)»

Факультет: Информатика и системы управления

Кафедра: Теоретическая информатика и компьютерные технологии

Лабораторная работа № 8

«Геометрическая интерпретация численных методов

линейной алгебры»

по дисциплине «Численные методы линейной алгебры»

Работу выполнил

студент группы ИУ9-72Б

Жук Дмитрий

Москва, 2022

Цель работы

Научиться создавать мобильные приложения решения задач по курсу «Численные методы линейной алгебры» с графическим пользовательским интерфейсом с использованием фреймворка Flutter на языке программирования Dart.

Реализация

В данной лабораторной работе необходимо разработать программу, реализующую на экране мобильного устройства один из алгоритмов численных методов, перечисленных в таблице ниже. Программа должна иметь графический пользовательский интерфейс, через который пользователь может задавать исходные данные задачи: размерность матрицы, значения элементов матрицы, вектора и т.д. Результат решения задачи должен обновляться автоматически при изменении любого параметра формы ввода данных.

Программа реализации численного метода, должна быть переписана на язык программирования Dart.

Решение системы уравнений 3x3 методом Гаусса

В форму ввода данных необходимо ввести коэффициенты матрицы и вектор правой части

Графическое представление неизвестного вектора и вектора правой части в изометрической проекции, вектор решения необходимо вывести цветом отличным от цвета вектора правой части. Значения координат векторов выводятся отдельно. Вывести отдельным цветом оси системы координат, в которой строятся вектора.

Приложение

```dart
// import 'dart:ui';
import 'package:flutter/material.dart';
import 'package:vector_math/vector_math.dart' as math;
import 'dart:math' as math;

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Custom Painter',
      theme: ThemeData(
        primarySwatch: Colors.pink,
      ),
      home: MyPainter(),
    );
  }
}

class MyPainter extends StatefulWidget {
  @override
  MyPainterState createState() => MyPainterState();
}

class MyPainterState extends State<MyPainter> {
  double _rotateA = 3 * math.pi / 4;
  double _rotateB = 2 * math.pi - math.asin(math.sqrt(3) / 3);
  double _zoom = 100;
  var m3 = math.Matrix3.zero();
  var v3 = math.Vector3.zero();
  math.Vector3? saveV3;
  math.Vector3? resV3;

  Widget numberMatrix(int i, int j) {
    return Padding(
      padding: const EdgeInsets.all(4),
      child: Row(
        children: [
          Text('${j == 3 ? 'b_${i + 1}' : 'A_${i + 1}_${j + 1}'} = '),
          Expanded(
            child: TextFormField(
```

```
            onChanged: (text) {
              var myDouble = double.tryParse(text);
              if (myDouble is! double) return;

              if (j == 3) {
                v3[i] = myDouble;
              } else {
                m3[i * 3 + j] = myDouble;
              }
            },
            decoration: const InputDecoration(
              hintText: '0',
              contentPadding:
                  EdgeInsets.symmetric(vertical: 0, horizontal: 12),
              border: OutlineInputBorder(),
            ),
          ),
        ),
      ],
    ),
  );
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text('lab_8_Zhuk'),
      actions: [
        IconButton(
          icon: const Icon(Icons.calculate),
          onPressed: () => setState(() {
            resV3 = eliminationGaussian(m3, saveV3 = v3.clone());
          }),
        ),
      ],
    ),
    body: SafeArea(
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: <Widget>[
          if (saveV3 != null) Row(children: [
            Text(saveV3.toString()),
            const Text(' '),
            Text(resV3.toString()),
```

```
          ]),
          Expanded(
            child: CustomPaint(
              painter: ShapePainter(
                rotateA: _rotateA,
                rotateB: _rotateB,
                zoom: _zoom,
                saveV3: saveV3,
                resV3: resV3,
              ),
              child: Container(),
            ),
          ),
          Row(
            children: [
              const Padding(
                padding: EdgeInsets.only(left: 16.0),
                child: Text('Rotation A'),
              ),
              Expanded(
                child: Slider(
                  value: _rotateA,
                  min: 0,
                  max: math.pi * 2,
                  onChanged: (value) => setState(() {
                    _rotateA = value;
                  }),
                ),
              ),
            ],
          ),
          Row(
            children: [
              const Padding(
                padding: EdgeInsets.only(left: 16.0),
                child: Text('Rotation B'),
              ),
              Expanded(
                child: Slider(
                  value: _rotateB,
                  min: 0,
                  max: math.pi * 2,
                  onChanged: (value) => setState(() {
                    _rotateB = value;
                  }),
```

```
                  ),
                ),
              ],
            ),
            Row(
              children: [
                const Padding(
                  padding: EdgeInsets.only(left: 16.0),
                  child: Text('Scale'),
                ),
                Expanded(
                  child: Slider(
                    value: _zoom,
                    min: 10,
                    max: 300,
                    onChanged: (value) => setState(() {
                      _zoom = value;
                    }),
                  ),
                ),
              ],
            ),
            ...[0, 1, 2].map((i) => Row(
                  crossAxisAlignment: CrossAxisAlignment.start,
                  children: [0, 1, 2, 3]
                      .map((j) => Expanded(child: numberMatrix(i, j)))
                      .toList(),
                )),
          ],
        ),
      ),
    );
  }
}

eliminationGaussian(math.Matrix3 matrix, math.Vector3 other) {
  var b = other.clone();
  var a = matrix.clone();
  const n = 3;

  // Прямой ход
  for (var k = 0; k < n - 1; k++) {
    for (var j = k + 1; j < n; j++) {
      var m = a[k * n + j] / a[k * n + k];
      for (var i = 0; i < n; i++) {
```

```
        a[i * n + j] -= m * a[i * n + k];
      }
      b[j] -= m * b[k];
    }
  }

  var x = math.Vector3.zero();

  // Обратный ход
  for (var i = n - 1; i >= 0; i--) {
    x[i] = b[i] / a[i * n + i];
    for (var c = n - 1; c > i; c--) {
      x[i] -= a[c * n + i] * x[c] / a[i * n + i];
    }
  }

  return x;
}

class LineMistic3 {
  final math.Vector3 point1;
  final math.Vector3 point2;
  final Color color;

  double get zIndex => (point1.z + point2.z) / -2;

  LineMistic3(this.point1, this.point2, this.color);
}

// FOR PAINTING POLYGONS
class ShapePainter extends CustomPainter {
  final double rotateA;
  final double rotateB;
  final double zoom;
  final math.Vector3? saveV3;
  final math.Vector3? resV3;
  ShapePainter({
    required this.rotateA,
    required this.rotateB,
    required this.zoom,
    required this.saveV3,
    required this.resV3,
  });

  @override
```

```
  void paint(Canvas canvas, Size size) {
    var matrix = math.Matrix4.identity()
      ..rotateX(rotateA)
      ..rotateY(rotateB)
      ..scaleAdjoint(zoom);

    var paint = Paint()
      ..color = Colors.teal
      ..strokeWidth = 5
      ..style = PaintingStyle.stroke
      ..strokeCap = StrokeCap.round;

    var lines = <LineMistic3>[];

    Offset forCanvas(math.Vector3 point, Size size) {
      return Offset(size.width / 2 + point.x, size.height / 2 - point.y);
    }

    void drawLine(math.Vector3 point1, math.Vector3 point2) {
      canvas.drawLine(
        forCanvas(point1, size),
        forCanvas(point2, size),
        paint,
      );
    }

    void drawLine2(math.Vector3 point1, math.Vector3 point2, Color color) {
      lines.add(LineMistic3(
        matrix.transform3(point1),
        matrix.transform3(point2),
        color,
      ));
    }

    drawLine2(math.Vector3(0, 0, 0), math.Vector3(1, 0, 0), Colors.red);
    drawLine2(math.Vector3(0, 0, 0), math.Vector3(0, 1, 0), Colors.green);
    drawLine2(math.Vector3(0, 0, 0), math.Vector3(0, 0, 1), Colors.blue);
    if (resV3 != null) {
      drawLine2(math.Vector3(0, 0, 0), (resV3 as math.Vector3).clone(),
Colors.pink);
    }
    if (saveV3 != null) {
      drawLine2(math.Vector3(0, 0, 0), (saveV3 as math.Vector3).clone(),
Colors.brown);
    }
```

```
    lines.sort((a, b) => a.zIndex > b.zIndex ? 1 : -1);

    for (var line in lines) {
      paint.color = line.color;
      drawLine(line.point1, line.point2);
    }
  }

  @override
  bool shouldRepaint(CustomPainter oldDelegate) {
    return true;
  }
}
```