

# Лабораторная работа No7 «Геометрическая интерпретация численных методов линейной алгебры»

## 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.

## Задание и порядок выполнения

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.

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

import 'package:vector_math/vector_math.dart' as mv;

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> {
  var ax = 5.0;
  var ay = 5.0;
  var az = 5.0;

  var bx = 5.0;
  var by = 5.0;
  var bz = 5.0;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Triangle'),
      ),
      body: SafeArea(
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: <Widget>[
            Expanded(
              child: CustomPaint(
                painter: CrossProductPainter(ax, ay, az,
                  bx, by, bz),
                child: Container(),
              ),
            ),

            const Padding(
              padding: EdgeInsets.only(left: 16.0),
              child: Text('first vector', style: TextStyle(fontSize: 30)),
            ),

            const SizedBox(height: 20),

            Row(
              children: const <Widget>[
                Expanded(
                  child: Padding(
                    padding: EdgeInsets.only(left: 16.0),
                    child: Text('x'),
                  ),
                ),
                Expanded(
                  child: Padding(
                    padding: EdgeInsets.only(left: 16.0),
                    child: Text('y'),
                  ),
                ),
                Expanded(
                  child: Padding(
                    padding: EdgeInsets.only(left: 16.0),

```

```

        child: Text('z'),
      ),
    ),
  ],
),
Row(
  children: <Widget>[
    Expanded(
      child: Slider(
        value: ax,
        min: 5.0,
        max: 200.0,
        label: ax.toInt().toString(),
        divisions: 50,
        onChanged: (value) {
          setState(() {
            ax = value;
          });
        },
      ),
    ),
    Expanded(
      child: Slider(
        value: ay,
        min: 5.0,
        max: 200.0,
        label: ay.toInt().toString(),
        divisions: 50,
        onChanged: (value) {
          setState(() {
            ay = value;
          });
        },
      ),
    ),
    Expanded(
      child: Slider(
        value: az,
        min: 5.0,
        max: 200.0,
        label: az.toInt().toString(),
        divisions: 50,
        onChanged: (value) {
          setState(() {
            az = value;
          });
        },
      ),
    ),
  ],
),

const SizedBox(height: 20),

const Padding(
  padding: EdgeInsets.only(left: 16.0),
  child: Text('second vector', style: TextStyle(fontSize: 30)),
),

```

```

const SizedBox(height: 20),

Row(
  children: const <Widget>[
    Expanded(
      child: Padding(
        padding: EdgeInsets.only(left: 16.0),
        child: Text('x'),
      ),
    ),
    Expanded(
      child: Padding(
        padding: EdgeInsets.only(left: 16.0),
        child: Text('y'),
      ),
    ),
    Expanded(
      child: Padding(
        padding: EdgeInsets.only(left: 16.0),
        child: Text('z'),
      ),
    ),
  ],
),
Row(
  children: <Widget>[
    Expanded(
      child: Slider(
        value: bx,
        min: 5.0,
        max: 200.0,
        label: bx.toInt().toString(),
        divisions: 50,
        onChanged: (value) {
          setState(() {
            bx = value;
          });
        },
      ),
    ),
    Expanded(
      child: Slider(
        value: by,
        min: 5.0,
        max: 200.0,
        label: by.toInt().toString(),
        divisions: 50,
        onChanged: (value) {
          setState(() {
            by = value;
          });
        },
      ),
    ),
    Expanded(
      child: Slider(

```

```

        value: bz,
        min: 5.0,
        max: 200.0,
        label: bz.toInt().toString(),
        divisions: 50,
        onChanged: (value) {
            setState(() {
                bz = value;
            });
        },
    )),
],
),
    ],
),
),
);
}
}

```

```
// FOR PAINTING POLYGONS
```

```
class CrossProductPainter extends CustomPainter {
```

```
    final double ax;
    final double ay;
    final double az;
```

```
    final double bx;
    final double by;
    final double bz;
```

```
CrossProductPainter(this.ax, this.ay, this.az,
    this.bx, this.by, this.bz);
```

```
var z = cos(pi/4);
```

```
@override
```

```
void paint(Canvas canvas, Size size) {
```

```
    var paint = Paint()
        ..color = Colors.black
        ..strokeWidth = 2
        ..style = PaintingStyle.stroke
        ..strokeCap = StrokeCap.round;
```

```
    var path = Path();
    var pathVec1 = Path();
    var pathVec2 = Path();
```

```
    Offset center = Offset(size.width / 2, size.height / 2);
```

```
    path.moveTo(center.dx, center.dy);
```

```

    path.lineTo(center.dx + 200, center.dy);
    path.moveTo(center.dx, center.dy);
    path.lineTo(center.dx, center.dy - 200);
    path.moveTo(center.dx, center.dy);
    path.lineTo(center.dx - 200*cos(pi/4), center.dy + 200*cos(pi/4));
    path.moveTo(center.dx, center.dy);
    canvas.drawPath(path, paint);

    pathVec1.moveTo(center.dx, center.dy);
    pathVec1.lineTo(center.dx + ax - az*z,
        center.dy - ay + az*z);

    paint.color = Colors.red;
    canvas.drawPath(pathVec1, paint);

    pathVec2.moveTo(center.dx, center.dy);
    pathVec2.lineTo(center.dx + bx - bz*z,
        center.dy - by + bz*z);

    paint.color = Colors.green;
    canvas.drawPath(pathVec2, paint);

    var cx = ay * bz - az * by;
    var cy = az * bx - ax * bz;
    var cz = ax * by - ay * bx;

    var pathVec3 = Path();
    pathVec3.moveTo(center.dx, center.dy);
    pathVec3.lineTo(center.dx + cx - cz*z,
        center.dy - cy + cz*z);

    paint.color = Colors.blue;

    print("$cx $cy $cz");
    mv.Vector3 vec1 = mv.Vector3(ax, ay, az);
    mv.Vector3 vec2 = mv.Vector3(bx, by, bz);
    print(vec2.cross(vec1));
    print("");
    canvas.drawPath(pathVec3, paint);

    path.close();
    pathVec1.close();
    pathVec2.close();
}

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

```

