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)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ: «Информатика и системы управления»
КАФЕДРА: «Теоретическая информатика и компьютерные технологии»

Летучка №349
«Поиск экстремума функции методом покоординатного спуска»

по курсу
«Разработка мобильных приложений»

Выполнил:
студент группы ИУ9-72Б
Караник А.А.

Проверено:
Посевин Д.П.

Москва, 2024

Цель работы

Реализовать поиск экстремума функции методом покоординатного спуска на Flutter с визуализацией графика функции в соответствии с вариантом с помощью библиотеки Ditredi и отображением траектории спуска.

Вариант 14

$f(x_1, x_2) = (x_1 - 2x_2)^2 + (x_2 - 3)^2$
$x_1^0 = 7$
$x_2^0 = 6$
$eps = 0,25$

Реализация

Исходный код:

```
import 'package:ditredi/ditredi.dart';
import 'package:flutter/material.dart';
import 'package:vector_math/vector_math_64.dart' show Vector3;
import 'dart:math';

class SurfacePlot extends StatefulWidget {
  @override
  _SurfacePlotState createState() => _SurfacePlotState();
}

class Curve {
  List<Vector3> firstPart = [];
  List<Vector3> secondPart = [];
}

class _SurfacePlotState extends State<SurfacePlot> {
  final diTreDiController = DiTreDiController()
    ..userScale = 2
    ..rotationX = -90
    ..rotationY = 0
    ..rotationZ = 12.5;

  List<Point3D> fullPath = [];
  double sliderValue = 0;

  double x1_0 = 7.0; // 9.0
  double x2_0 = 6.0; // 5.0
  double eps = 0.25; // 0.01

  double x1_n = 0;
  double x2_n = 0;
  double z_n = 0;

  double x1_min = 6.0;
  double x2_min = 3.0;
  double z_min = 0.0;

  double maxZ = 40.0;

  List<Curve> curves = [];
  bool showPointSurface = true;

  @override
```

```
void initState() {
  super.initState();
  fullPath = _coordinateDescentPath(Vector3(x1_0, x2_0, _function(x1_0, x2_0)), eps);
  z_min = _function(x1_min, x2_min);
  curves = _getCurves();
}

void _handleScaleUpdate(ScaleUpdateDetails details) {
  setState(() {
    if (details.scale != 1.0) {
      diTreDiController.userScale += (details.scale - 1) * 0.01;
      diTreDiController.userScale = diTreDiController.userScale.clamp(0.5, 10.0);
    } else {
      diTreDiController.rotationX += details.focalPointDelta.dy * 0.5;
      diTreDiController.rotationZ += details.focalPointDelta.dx * 0.5;
    }
  });
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('fly349'),
    ),
    body: Column(
      children: [
        Expanded(
          child: Center(
            child: GestureDetector(
              onScaleUpdate: _handleScaleUpdate,
              child: DiTreDi(
                controller: diTreDiController,
                figures: [
                  if (showPointSurface)
                    ..._generatePointSurface()
                  else
                    _generateSurface(),
                  ..._generateAxes(),
                  ..._getPathPoints(),
                  ..._extraLines(),
                  ..._extraPoints(),
                ],
              ),
            ),
          ),
        ),
        Padding(
          padding: const EdgeInsets.all(16.0),
          child: Column(
            children: [
              Row(
                mainAxisAlignment: MainAxisAlignment.center,
                children: [
                  Text("Segments"),
                  Switch(
                    value: showPointSurface,
                    onChanged: (value) {
                      setState(() {
                        showPointSurface = value;
                      });
                    },
                  ),
                  Text("Points"),
                ],
```

```
              ),
              Text("Прогресс покоординатного спуска"),
              Slider(
                value: sliderValue,
                min: 0,
                max: fullPath.length.toDouble() - 1,
                divisions: fullPath.length - 1,
                label: sliderValue.toInt().toString(),
                onChanged: (value) {
                  setState(() {
                    sliderValue = value;
                  });
                },
              ),
            ],
          ),
        ),
      ],
    ),
  );
}

List<Point3D> _generatePointSurface() {
  List<Point3D> points = [];
  double step = 0.1;

  for (double x1 = -4; x1 <= 16; x1 += step) {
    for (double x2 = -7; x2 <= 13; x2 += step) {
      double z = _function(x1, x2);
      if (z < maxZ) {
        points.add(Point3D(Vector3(x1, x2, z), color: Colors.blue));
      }
    }
  }
  return points;
}

Mesh3D _generateSurface() {
  List<Face3D> faces = [];

  for (int i = 1; i < curves[0].firstPart.length; i++) {
    faces.add(Face3D.fromVertices(
      Vector3(x1_min, x2_min, z_min),
      curves[0].firstPart[i],
      curves[0].firstPart[i - 1],
      color: Colors.blue,
    ));
  }

  for (int i = 1; i < curves[0].secondPart.length; i++) {
    faces.add(Face3D.fromVertices(
      Vector3(x1_min, x2_min, z_min),
      curves[0].secondPart[i - 1],
      curves[0].secondPart[i],
      color: Colors.blue,
    ));
  }

  faces.add(Face3D.fromVertices(
    Vector3(x1_min, x2_min, z_min),
    curves[0].firstPart[0],
    curves[0].secondPart[0],
    color: Colors.blue,
  ));
```

```
    faces.add(Face3D.fromVertices(
      Vector3(x1_min, x2_min, z_min),
      curves[0].secondPart[curves[0].secondPart.length - 1],
      curves[0].firstPart[curves[0].firstPart.length - 1],
      color: Colors.blue,
    ));

    for (int i = 1; i < curves.length; i++) {
      faces.add(Face3D.fromVertices(
        curves[i].secondPart[0],
        curves[i - 1].secondPart[0],
        curves[i - 1].firstPart[0],
        color: Colors.blue,
      ));

      faces.add(Face3D.fromVertices(
        curves[i].secondPart[0],
        curves[i - 1].firstPart[0],
        curves[i].firstPart[0],
        color: Colors.blue,
      ));

      faces.add(Face3D.fromVertices(
        curves[i - 1].secondPart[curves[i - 1].secondPart.length - 1],
        curves[i].firstPart[curves[i].firstPart.length - 1],
        curves[i - 1].firstPart[curves[i - 1].firstPart.length - 1],
        color: Colors.blue,
      ));

      faces.add(Face3D.fromVertices(
        curves[i - 1].secondPart[curves[i - 1].secondPart.length - 1],
        curves[i].secondPart[curves[i].secondPart.length - 1],
        curves[i].firstPart[curves[i].firstPart.length - 1],
        color: Colors.blue,
      ));

      int k = 0;

      for (int j = 0; j < curves[i].firstPart.length - 1; j++) {
        if (j + 1 == curves[i - 1].firstPart.length) {
          k = j;
        }
        if (j + 1 >= curves[i - 1].firstPart.length) {
          faces.add(Face3D.fromVertices(
            curves[i - 1].firstPart[k],
            curves[i].firstPart[j + 1],
            curves[i].firstPart[j],
            color: Colors.blue,
          ));
          continue;
        }
        faces.add(Face3D.fromVertices(
          curves[i - 1].firstPart[j],
          curves[i - 1].firstPart[j + 1],
          curves[i].firstPart[j],
          color: Colors.blue,
        ));
        faces.add(Face3D.fromVertices(
          curves[i - 1].firstPart[j + 1],
          curves[i].firstPart[j + 1],
          curves[i].firstPart[j],
          color: Colors.blue,
        ));
      }
```

```
    for (int j = 0; j < curves[i].secondPart.length - 1; j++) {
      if (j + 1 == curves[i - 1].secondPart.length) {
        k = j;
      }
      if (j + 1 >= curves[i - 1].secondPart.length) {
        faces.add(Face3D.fromVertices(
          curves[i].secondPart[j],
          curves[i].secondPart[j + 1],
          curves[i - 1].secondPart[k],
          color: Colors.blue,
        ));
        continue;
      }
      faces.add(Face3D.fromVertices(
        curves[i].secondPart[j],
        curves[i - 1].secondPart[j + 1],
        curves[i - 1].secondPart[j],
        color: Colors.blue,
      ));
      faces.add(Face3D.fromVertices(
        curves[i].secondPart[j],
        curves[i].secondPart[j + 1],
        curves[i - 1].secondPart[j + 1],
        color: Colors.blue,
      ));
    }
  }

  return Mesh3D(faces);
}

List<Curve> _getCurves() {
  double step = 0.5;
  double delta = 0.01;
  double deltaZ = 0.5;
  double firstZ = 1.0;

  List<Curve> curves = [];

  for (double z = firstZ; z < maxZ; z += deltaZ) {
    List<Vector3> firstPart = [];
    List<Vector3> secondPart = [];

    for (double x2 = 3 + sqrt(z) - delta; x2 >= 3 - sqrt(z); x2 -= step) {
      double firstX1 = _findFirstX1(x2, z);
      double secondX1 = _findSecondX1(x2, z);
      firstPart.add(Vector3(firstX1, x2, z));
      secondPart.add(Vector3(secondX1, x2, z));
    }

    var curve = Curve();
    curve.firstPart = firstPart;
    curve.secondPart = secondPart;
    curves.add(curve);
  }

  return curves;
}

List<Line3D> _generateAxes() {
  return [
    Line3D(Vector3(0, 0, 0), Vector3(20, 0, 0),
        color: Colors.red, width: 2.0),
    Line3D(Vector3(0, 0, 0), Vector3(0, 20, 0),
        color: Colors.green, width: 2.0),
```

```
    Line3D(Vector3(0, 0, 0), Vector3(0, 0, 20),
        color: Colors.blue, width: 2.0),
  ];
}

double _function(double x1, double x2) {
  return pow(x1 - 2 * x2, 2).toDouble() + pow(x2 - 3, 2).toDouble();
}

double _findFirstX2(double x1, double z) {
  return (2 * x1 + 3 + sqrt(-pow(x1, 2) + 12 * x1 - 36 + 5 * z)) / 5.0;
}

double _findSecondX2(double x1, double z) {
  return (2 * x1 + 3 - sqrt(-pow(x1, 2) + 12 * x1 - 36 + 5 * z)) / 5.0;
}

double _findFirstX1(double x2, double z) {
  return (2 * x2 + sqrt(z - pow(x2, 2) + 6 * x2 - 9));
}

double _findSecondX1(double x2, double z) {
  return (2 * x2 - sqrt(z - pow(x2, 2) + 6 * x2 - 9));
}

List<Point3D> _getPathPoints() {
  int pointCount = sliderValue.toInt();
  return fullPath.sublist(0, pointCount + 1);
}

List<Line3D> _extraLines() {
  int pointCount = sliderValue.toInt();
  return [
    Line3D(
        fullPath[pointCount].position,
        Vector3(fullPath[pointCount].position.x,
            fullPath[pointCount].position.y, 0),
        color: Colors.amber,
        width: 2.0),
  ];
}

List<Point3D> _extraPoints() {
  int pointCount = sliderValue.toInt();
  return [
    Point3D(
        Vector3(fullPath[pointCount].position.x,
            fullPath[pointCount].position.y, 0),
        color: Colors.black,
        width: 3.0),
  ];
}

List<Point3D> _coordinateDescentPath(Vector3 start, double epsilon) {
  List<Point3D> path = [];
  Vector3 current = start;
  double currentValue = _function(current.x, current.y);
  path.add(Point3D(current, color: Colors.red, width: 3.0));

  double step = 0.1;

  while (true) {
    List<Vector3> directions = [
      Vector3(current.x + step, current.y,
          _function(current.x + step, current.y)),
```

```
        Vector3(current.x - step, current.y,
            _function(current.x - step, current.y)),
        Vector3(current.x, current.y + step,
            _function(current.x, current.y + step)),
        Vector3(current.x, current.y - step,
            _function(current.x, current.y - step)),
      ];

      directions.sort((a, b) => a.z.compareTo(b.z));
      Vector3 next = directions.first;

      double nextValue = _function(next.x, next.y);
      if ((currentValue - nextValue).abs() < epsilon) {
        path.add(Point3D(next, color: Colors.green, width: 6.0));
        x1_n = next.x;
        x2_n = next.y;
        z_n = _function(x1_n, x2_n);
        break;
      }

      path.add(Point3D(next, color: Colors.red, width: 3.0));
      current = next;
      currentValue = nextValue;
    }
    return path;
  }
}

void main() {
  runApp(MaterialApp(
    home: SurfacePlot(),
  ));
}
```

Результаты

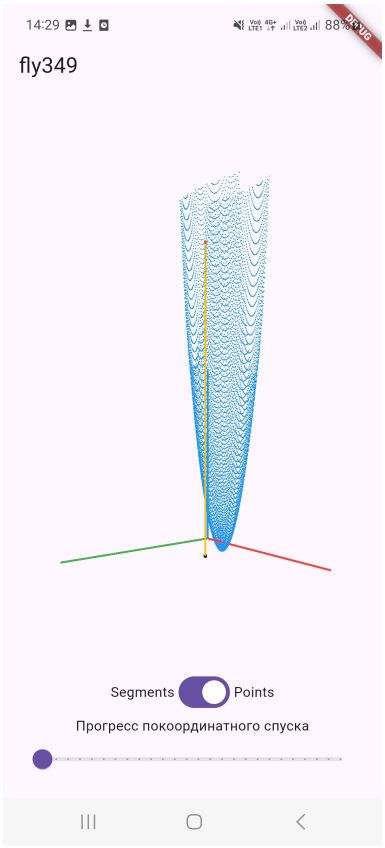

Рис. 1: результаты

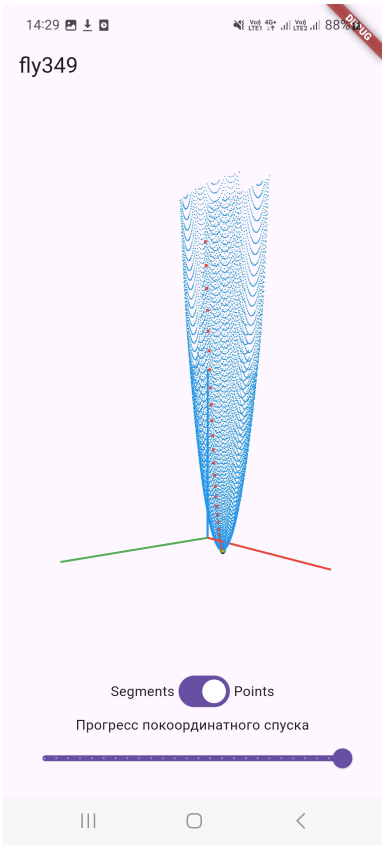

Рис. 2: результаты

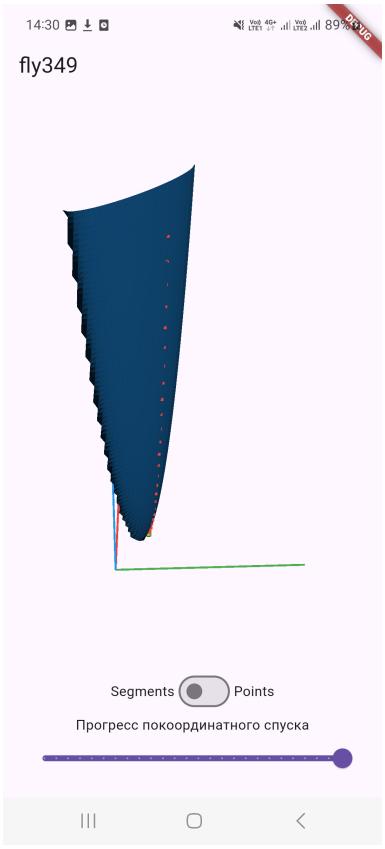

Рис. 3: результаты

Вывод

В ходе выполнения данной работы был успешно реализован метод покоординатного спуска с визуализацией в Flutter. Приложение наглядно демонстрирует процесс нахождения минимума и особенности поведения метода на квадратичных функциях.