МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Кафедра компьютерных технологий и систем

Козлов Владислав Дмитриевич

АЛГОРИТМЫ И ИНСТРУМЕНТЫ ФОРМИРОВАНИЯ И ВИЗУАЛИЗАЦИИ ВЕКТОРНЫХ ЦИФРОВЫХ ПОЛЕЙ

Дипломная работа

Научный руководитель:
профессор кафедры КТС,
доктор физ.-мат. наук,
профессор Таранчук В. Б.

Допущена к защите
«____» _____ 2025 г.
Зав. кафедрой компьютерных систем и технологий
доктор педагогических наук, профессор,
Казаченок В. В.

Минск, 2025

ОГЛАВЛЕНИЕ

РЕФЕРАТ

Дипломная работа содержит: 52 страницы, 37 иллюстраций (рисунков), 16 используемых источников.

Ключевые слова: Wolfram Mathematica, визуализация, векторные поля, скалярные поля, параллельные вычисления, цунами, уравнение в частных производных

Объектом исследования являются векторный поля – их математическая модель, формирование, преобразование, интерполяция и визуализация в 2D и 3D графике, а также их применение для моделирования динамических физических процессов.

Целью дипломной работы является разработка, формирование и визуализации векторных полей с использованием Wolfram Mathematica и JavaFX, оптимизация вычислительных процессов с применением параллельных вычислений. Так же моделирование динамических процессов, демонстрируемое на примере волновых моделей, таких как цунами.

В результате исследования проведено сравнение формирования векторного поля с использованием Wolfram Mathematica и Java. В системе компьютерной алгебры проведён сравнительный анализ параллельных алгоритмов, что позволило ускорить обработку данных. С помощью Wolfram Mathematica смоделировано и визуализировано векторное поле цунами.

Область применения: математическое моделирование физических процессов и динамических систем, прогнозирование и анализ природных явлений.

РЭФЕРАТ

Дыпломная праца змяшчае: 52 старонкі, 37 ілюстрацый (малюнкаў), 16 выкарыстаных літаратурных крыніц.

Ключавыя словы: Wolfram Mathematica, візуалізацыя, вектарныя палі, скалярныя палі, паралельныя вылічэнні, цунамі, раўнанне з частковымі вытворнымі.

Аб'ект даследавання: вектарныя палі – іх матэматычная мадэль, фарміраванне, пераўтварэнне, інтэрпаляцыя і візуалізацыя ў 2D і 3D графіцы, а таксама іх прымяненне для мадэлявання дынамічных фізічных працэсаў.

Мэта дыпломнай працы: распрацоўка, фарміраванне і візуалізацыя вектарных палёў з выкарыстаннем Wolfram Mathematica і JavaFX, аптымізацыя вылічальных працэсаў з прымяненнем паралельных вылічэнняў. Таксама мадэляванне дынамічных працэсаў, дэманстраванае на прыкладзе хвалявых мадэляў, такіх як цунамі.

Вынік даследавання: праведзена параўнанне фарміравання вектарнага поля з выкарыстаннем Wolfram Mathematica і Java. У сістэме камп’ютарнай алгебры здзейснены параўнаўчы аналіз паралельных алгарытмаў, што дазволіла паскорыць апрацоўку дадзеных. З дапамогай Wolfram Mathematica змадэлявана і візуалізавана цунамі.

Сфера прымянення: матэматычнае мадэляванне фізічных працэсаў і дынамічных сістэм, прагназаванне і аналіз прыродных з'яў.

ABSTRACT

Diploma work contains: 52 pages, 37 illustrations (figures), 16 references.

Keywords: Wolfram Mathematica, visualization, vector fields, scalar fields, parallel computing, tsunamis, partial differential equations.

Object of study: The focus of the research is on vector fields – their mathematical models, formation, transformation, interpolation, and visualization in 2D and 3D graphics, as well as their application in modeling dynamic physical processes.

Objective: The objective of the thesis is to develop, form, and visualize vector fields using Wolfram Mathematica and JavaFX, optimizing computational processes through parallel computing. Also, modeling dynamic processes demonstrated using wave models such as tsunamis.

Result: The study involved a comparison of the formation of vector fields using Wolfram Mathematica and Java. Within the computer algebra system, a comparative analysis of parallel algorithms was performed, which accelerated data processing. Tsunamis were modeled and visualized using Wolfram Mathematica.

Field of application: Mathematical modeling of physical processes and dynamic systems, prediction, and analysis of natural phenomena.

ВВЕДЕНИЕ

Современные вычислительные методы и информационных технологии играют ключевую роль в моделировании, анализе и визуализации данных. Особое значение среди таких задач занимает исследование векторных полей, которые находят применение в различных областях науки и техники – от физики и математики до инженерных расчётов и геоинформационных систем.

Векторные поля позволяет описать множество природных и технических процессов. С их помощью можно визуализировать распределение скоростей в жидкости и газе, магнитные и электрические поля, тепловые потоки и многое другое.

Примеры применения векторных полей:

- Компания Яндекс использует векторные поля для отображения направлений ветра в своей подсистеме «Яндекс Погода», так же для их дальнейшего анализа и пре;
- Компания Boeing применяет векторные поля для анализа обтекания воздушных потоков вокруг крыла самолёта, для улучшения аэродинамики и предотвращения нежелательных поломок;
- Компания Boston Dynamics анализирует силовые поля и поля гравитации, для моделирования движения роботов. Для планирования траектории робота, обхода препятствий и координации движения применяются векторные поля.

Эффективная визуализация векторных полей позволяет не только наглядно представить их структуру, но и выявить скрытые закономерности, оценить поведение системы в различных условиях, а также повысить точность и скорость анализа.

Целью данной дипломной работы является изучение методов формирования и визуализации векторных полей с использованием различных программных средств, включая Wolfram Mathematica и JavaFX. А также исследование возможностей их параллельной обработки в системе Wolfram Mathematica и сравнительный анализ основных методов. Кроме того, в работе рассматривается прикладной пример моделирования волновых процессов, в частности – цунами, что позволяет продемонстрировать практическое применение разработанных методов.

Актуальность выбранной темы обусловлена как широким спектром областей применения векторных полей, так и необходимостью разработки удобных и наглядных средств для их визуализации и анализа. Особенно это важно при обработке больших объёмов данных, где важна производительность системы, точность выбранных алгоритмов, способность оптимизации

вычислений. Всё это делает задачу создания эффективных инструментов для работы с векторными полями особенно актуальной и востребованной.

Глава 1. ВЕКТОРНОЕ ПОЛЕ

1.1 Формирование векторного поля

Векторное поле – это отображение, которое каждой точке рассматриваемого пространства ставит в соответствие вектор с началом в этой точке. Векторное поле отличается от скалярного (например, температуры) тем, что оно описывает величину и направление в каждой точке пространства, в то время как скалярное поле описывает только величину без учёта направления.

Двухмерное векторное поле можно описать как вектор-функцию $f(x, y)$, заданную на какой-то плоскости. Примером может служить ветер, дующий по плоскости: в каждой точке плоскости он имеет определённое направление и скорость, представимые в виде вектора.

В трёхмерном пространстве векторные поля описываются функцией $f(x, y, z)$. Примером может быть векторное поле скоростей жидкости: в каждой точке пространства они имеет определённое направление движения и скорость, которые можно представить в виде вектора.

1.2 Визуализация аналитически заданных векторных полей

При работе с аналитическими функциями, которые описывают векторные поля, Mathematica предоставляет обширный набор средств для их визуального отображения. С помощью этих инструментов можно изучать структуру полей и проводить анализ их поведения.

В зависимости от конкретных задач Wolfram Mathematica предлагает разные возможности для визуализации:

- VectorPlot – строит 2D график векторного поля.
- VectorPlot3D – строит 3D график векторного поля.
- StreamPlot – строит график линий тока, соответствующих векторному полю на плоскости.
- StreamPlot3D – строит график линий тока, соответствующих векторному полю в трёхмерной области.
- VectorDensityPlot – строит график, показывающий векторное поле с наложенным графиком плотности скалярного поля.
- StreamDensityPlot – строит визуализацию, аналогичную StreamPlot, также, добавляя плотностный фон.
- SliceVectorPlot3D – строит двухмерный срез векторного поля в трёхмерном пространстве.
- LineIntegralConvolutionPlot – строит график, являющийся сверткой векторного поля с другой функцией.

• VectorDisplacementPlot – строит график смещения точки под действием векторного поля.

• VectorDisplacementPlot3D – строит график смещения точки под действием векторного поля в трёхмерной плоскости.

На рисунке 1.1 изображено векторное поле, построенное с помощью функции VectorPlot.

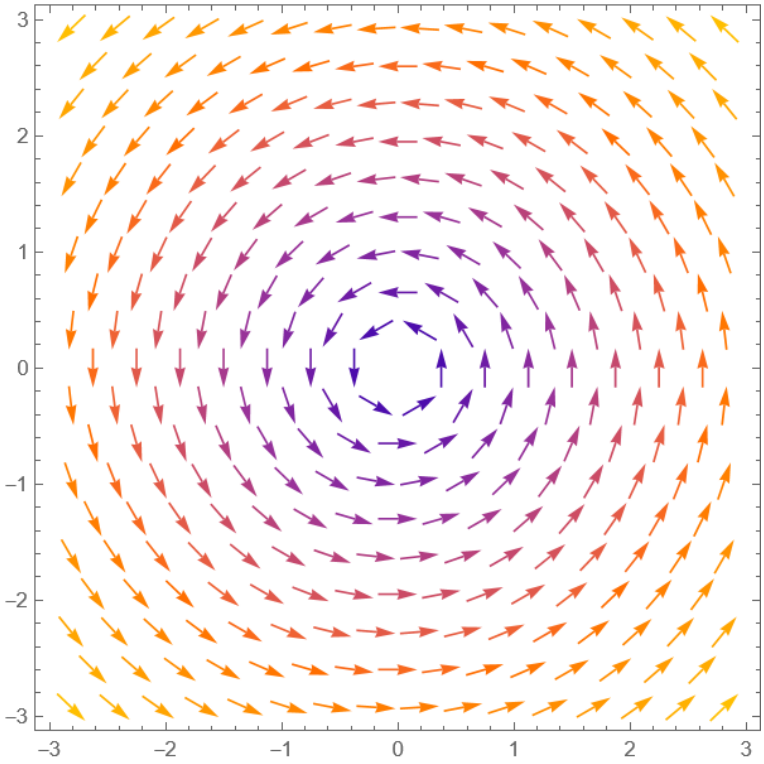


Рисунок 1.1 – Использование функции VectorPlot

На рисунке 1.2 изображено векторное поле, построенное с помощью функции SliceVectorPlot3D.

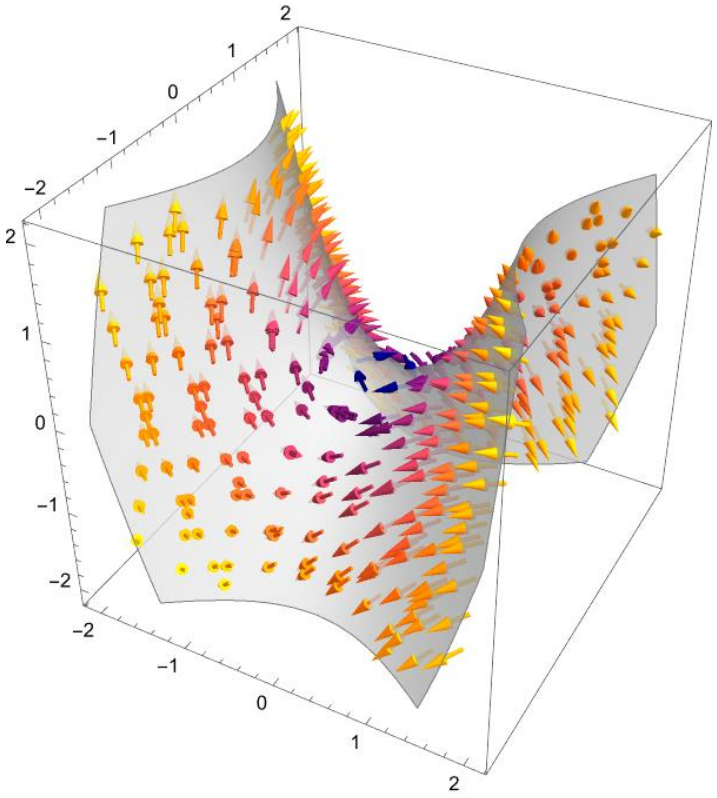


Рисунок 1.2 – Использование функции SliceVectorPlot3D

На рисунке 1.3 изображено векторное поле, построенное с помощью функции VectorDensityPlot.

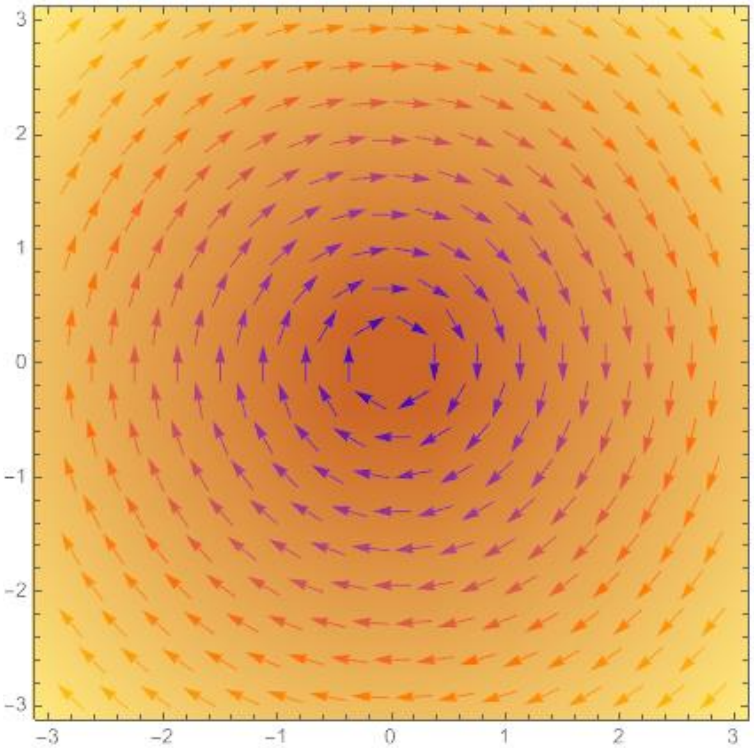


Рисунок 1.3 – Использование функции VectorDensityPlot

На рисунке 1.4 изображено векторное поле, построенное с помощью функции VectorDisplacementPlot.

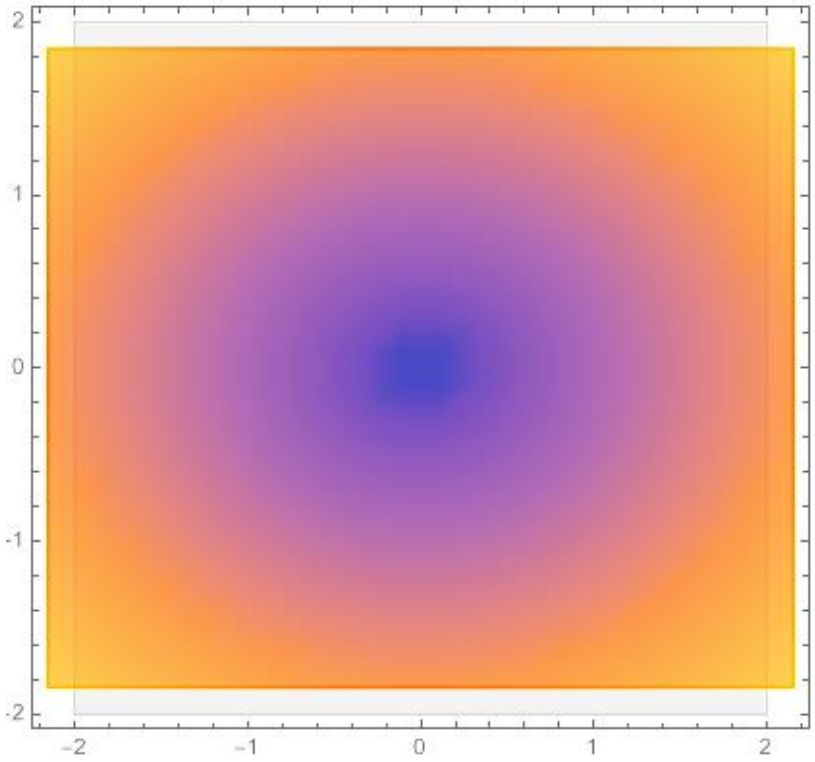


Рисунок 1.4 – Использование функции VectorDisplacementPlot

На рисунке 1.5 изображено векторное поле, построенное с помощью функции LineIntegralConvolutionPlot.

Рисунок 1.5 – Использование функции LineIntegralConvolutionPlot

1.3 Визуализация и интерполяция данных векторных полей в 2D и 3D

Возникают ситуации, когда необходимо работать с дискретными векторами. Работа с ними является достаточно сложной. В данном случае важно подобрать алгоритм для визуализации этого списка векторов.

Существуют функции, которые работают со списками векторов, принимая в качестве аргумента их координаты. Для визуализации этого списка используются методы, с помощью которых можно отобразить направление, величину векторов, характер их изменения в пространстве.

Функции, применяемые для отображения списка векторов:

- ListVectorPlot – создает двухмерный график векторного поля из списка векторов.
- ListVectorPlot3D – создает трёхмерный график векторного поля из списка векторов.
- ListStreamPlot – создает визуализацию векторного поля в виде обтекаемых линий.
- ListStreamPlot3D – создает трёхмерную визуализацию векторного поля с использованием обтекаемых поверхностей.
- ListVectorDensityPlot – строит плотностное векторное поле на основе списка данных
- ListStreamDensityPlot – строит потоковые линии на основе массива данных.
- ListLineIntegralConvolutionPlot – визуализирует векторное поле с помощью линейной интегральной свёртки.
- ListVectorDisplacementPlot – создает график вектора смещения на основе заданных начальных условий.
- ListVectorDisplacementPlot3D – создает трехмерное изображение вектора смещения.

На рисунке 1.6 изображено векторное поле, построенное с помощью функции ListVectorPlot3D.

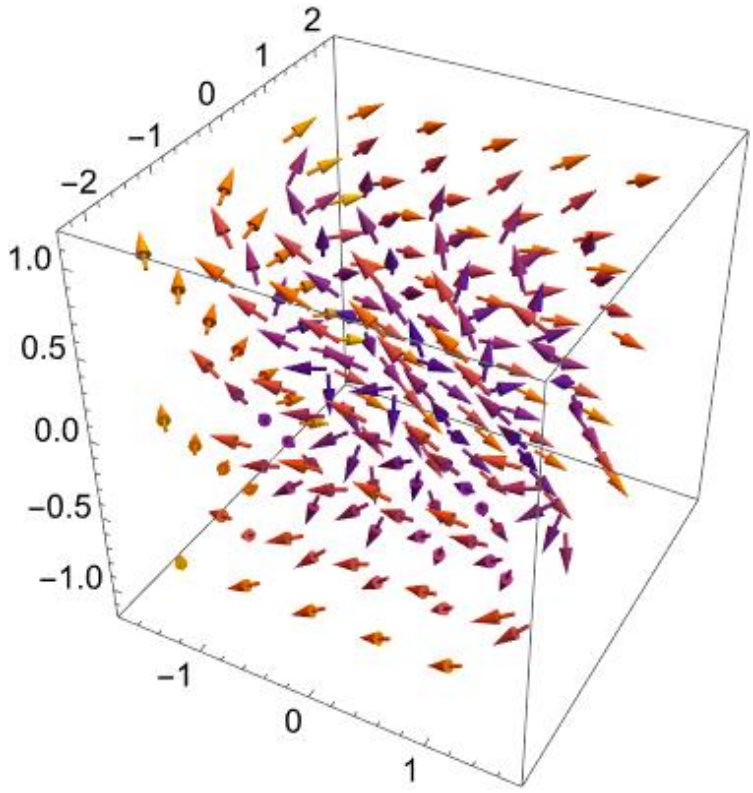


Рисунок 1.6 – Использование функции ListVectorPlot3D

На рисунке 1.7 изображено векторное поле, построенное с помощью функции ListStreamPlot.

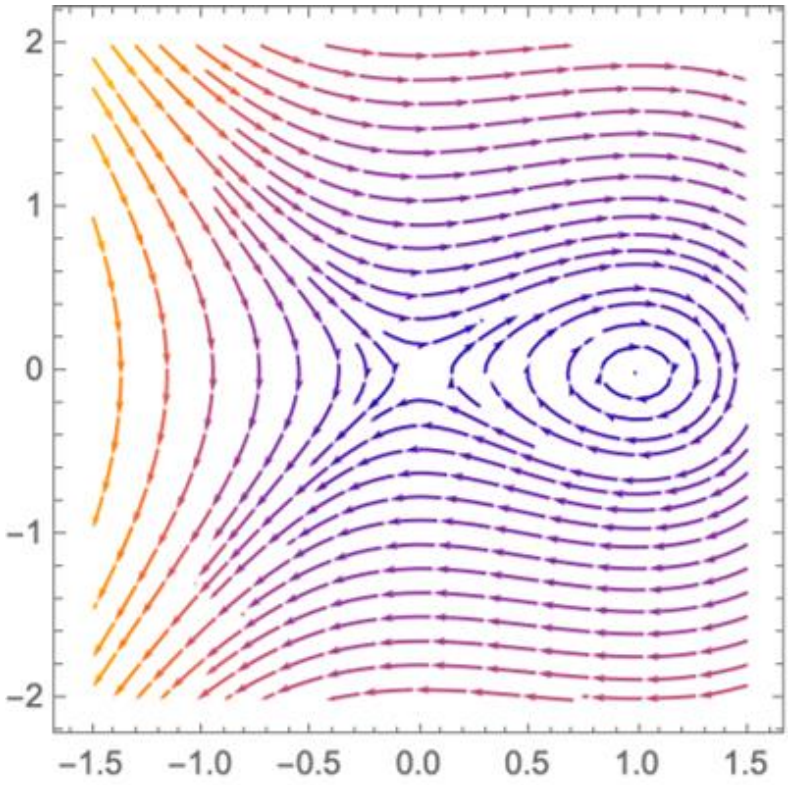


Рисунок 1.7 – Использование функции ListStreamPlot

На рисунке 1.8 изображено векторное поле, построенное с помощью функции ListVectorDisplacementPlot.

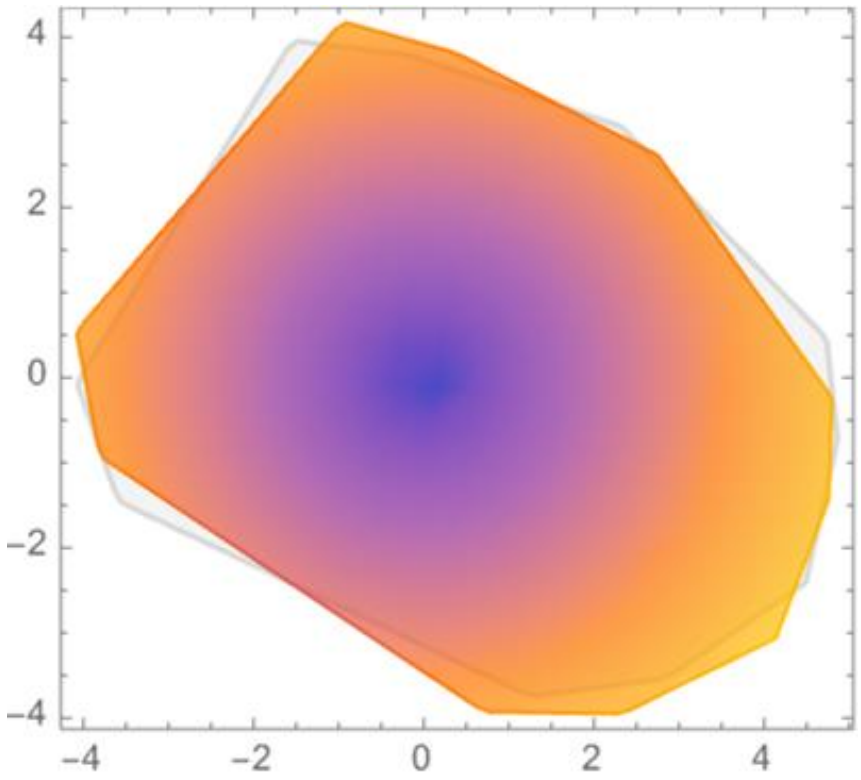


Рисунок 1.8 – Использование функции ListVectorDisplacementPlot

1.4 Линии тока на плоскости

Линия тока – в гидромеханике линия, направление касательной к которой в каждой точке совпадает с направлением скорости частицы жидкости в этой точке. Линия тока является частным случаем векторной линии, когда в качестве векторного поля выступает поле скоростей точек сплошной среды. Набор линий даёт представление о потоке жидкости или газа в текущий момент времени. В стационарном потоке частицы движутся вдоль линий тока. Однако, в случае неустановившегося движения линии тока не совпадают с траекториями.

В Wolfram Mathematica для визуализации линий тока на плоскости используется функция StreamPlot.

На рисунке 1.9 представлен график линий тока на плоскости, построенный с помощью функции StreamPlot.

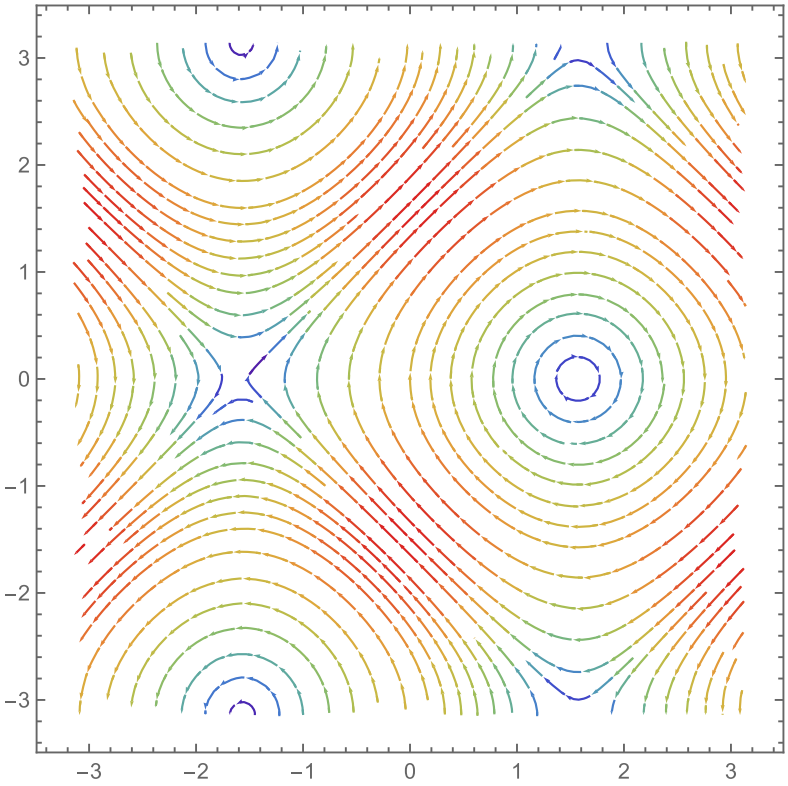


Рисунок 1.9 – Линии тока на плоскости

1.5 Линии тока на плоскости с фоном плотности функции

Распределение плотности с наложением линий тока, сетки, градиента (f – скалярная функция 2-х переменных, gradient – векторное поле).

StreamDensityPlot – линии тока на плоскости с фоном плотности функции (плотностная диаграмма потоков). StreamDensityPlot генерирует карту потока векторного поля $\{v_x, v_y\}$, как функцию x и y, с фоном карты плотности скалярного поля s.

StreamDensityPlot – метод визуализации, который объединяет в себе несколько: StreamPlot и DensityPlot. Тем самым, он является особенно полезен в задачах механики, физики, при изучении потенциала полей в математическом моделировании.

Глава 2. ФОРМИРОВАНИЕ И ВИЗУАЛИЗАЦИЯ ВЕКТОРНЫХ ПОЛЕЙ В СИСТЕМЕ WOLFRAM MATHEMATICA

2.1 Скалярное поле

Рассмотрим функцию (2.1).

$$f(x,y) = e^{-(x-9)^2-(y-2)^2} - \frac{2}{3}\, e^{-(x-7)^2-(3\,y-3)^2} + e^{-(x-6.5)^2-(y-3)^2} \qquad (2.1)$$

На рисунок 3.1 показана визуализация этой функции с помощью Plot3D.

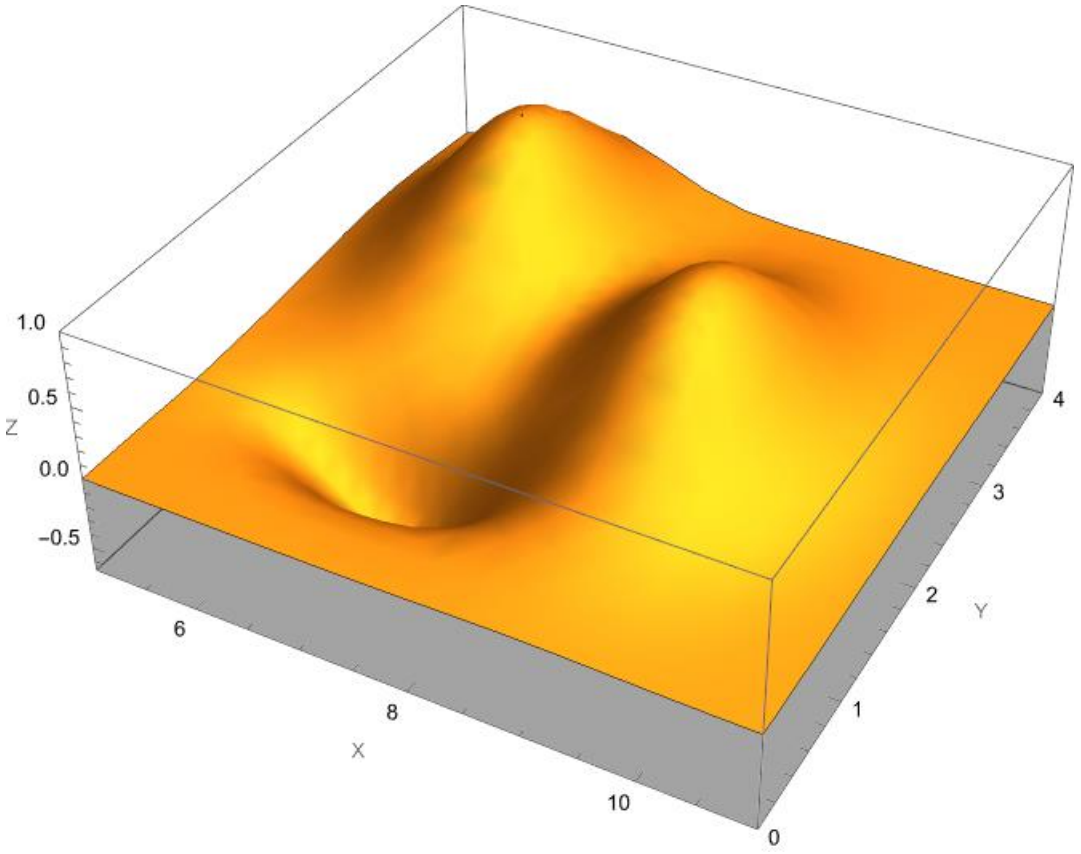


Рисунок 2.1 – Визуализация функции (2.1) с помощью Plot3D

На рисунке 2.2 показан пример применения цветовой схемы ColorFunction−> Hue.

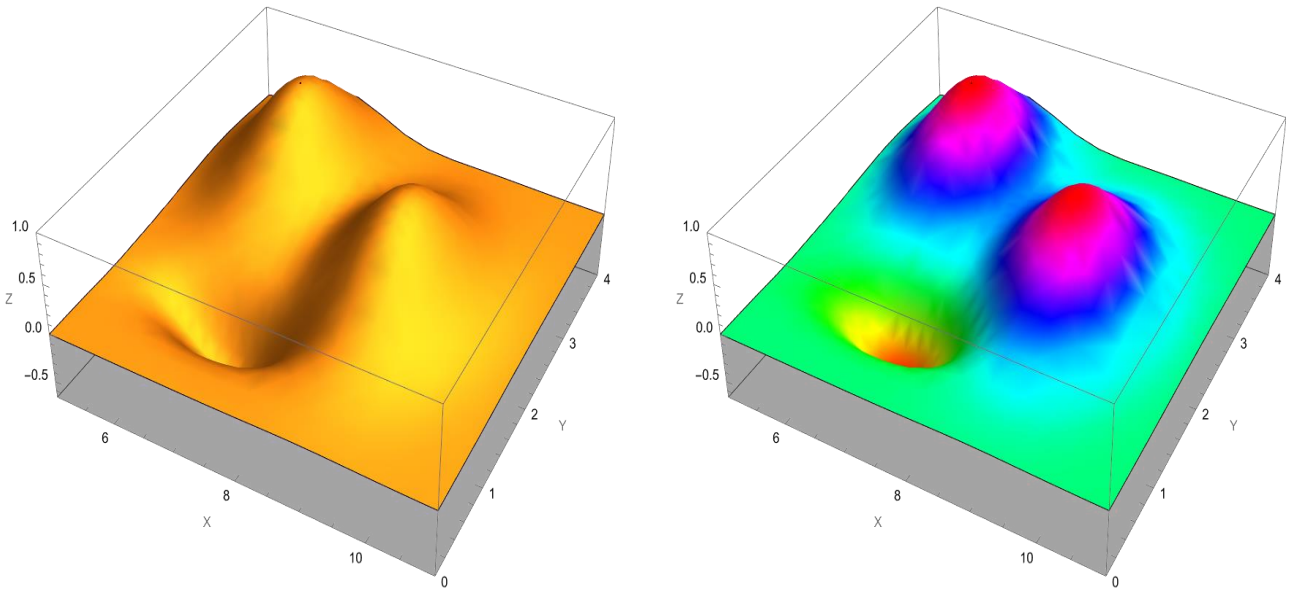


Рисунок 2.2 – Пример применения цветовой схемы

Увеличим количество начальных точек, для улучшения качества графика PlotPoints−> 100.

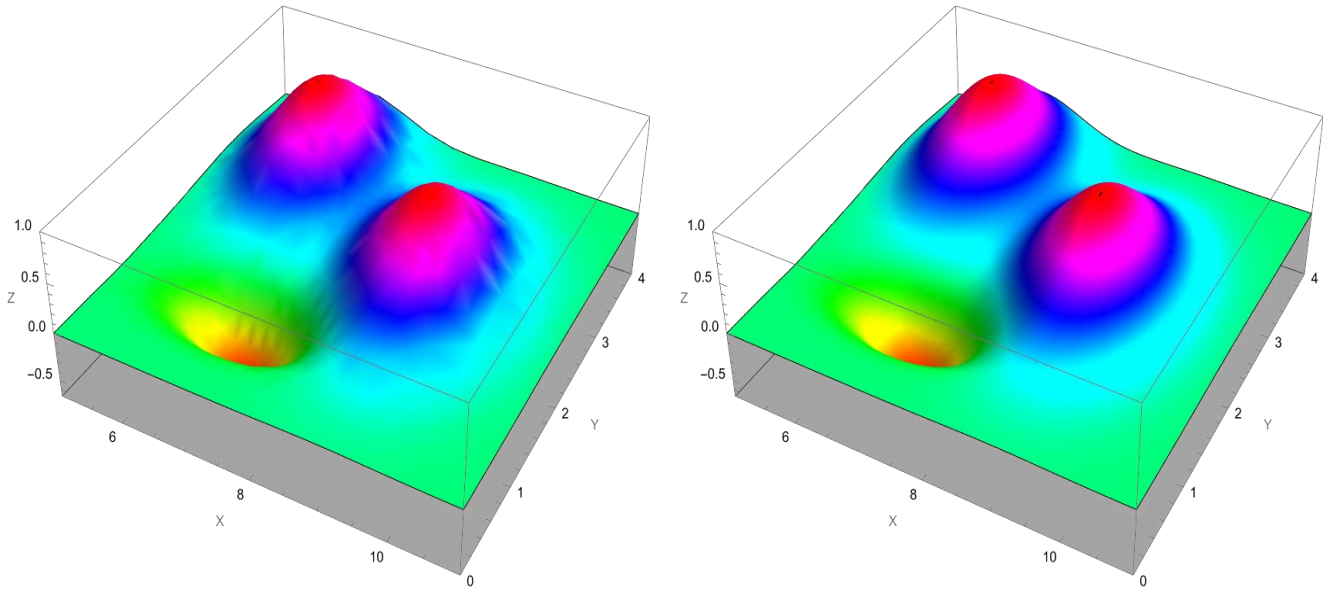


Рисунок 2.3 – Пример использования PlotPoints

2.2 Преобразования скалярного поля в векторное

Рассмотрим два способа получения из скалярного поля векторное:

1. Гамильтоново векторное поле функции, с компонентами $\partial f/\partial y$ и $-\partial f/\partial x$.
2. Векторное поле градиента функции, с компонентами $\partial f/\partial x$ и $\partial f/\partial y$.

На рисунке 2.4 показано построение Гамильтоново векторного поля $\text{hamf} = \{\text{gradf}[[2]], -\text{gradf}[[1]]\}$ и векторного поля градиента функции $\text{gradf} = D[f[x, y], \{\{x, y\}\}]$.

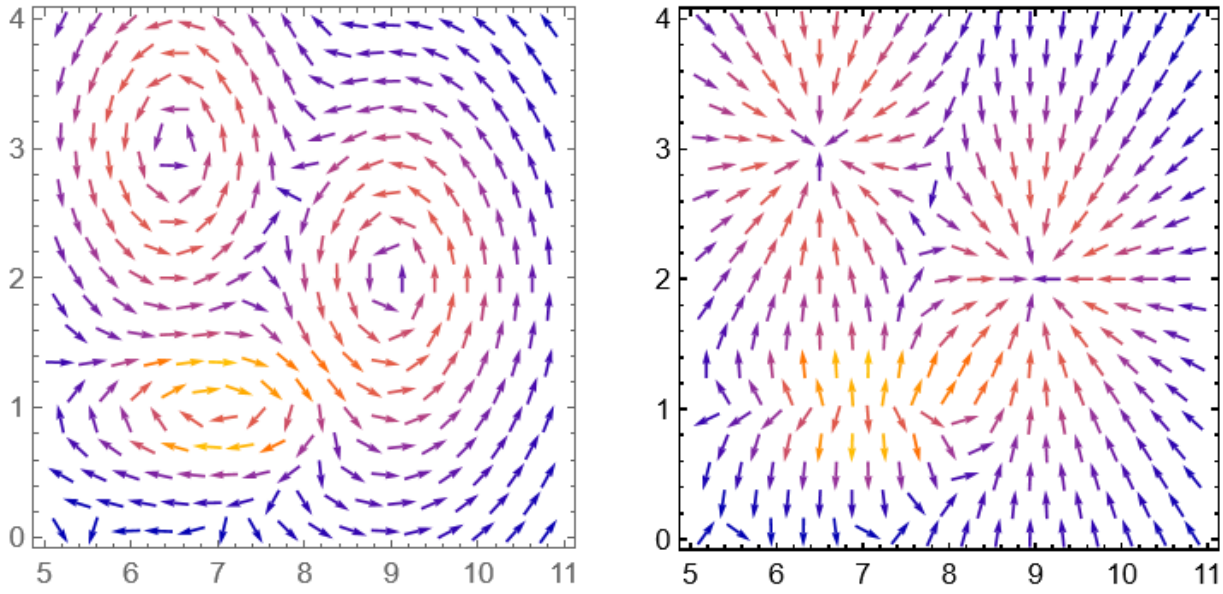


Рисунок 2.4 – Гамильтоново поле и поле градиента

Далее будем использовать векторное поле градиента функции.

На рисунке 2.5 представлено векторное поле градиента с изменением цветовой схемы.

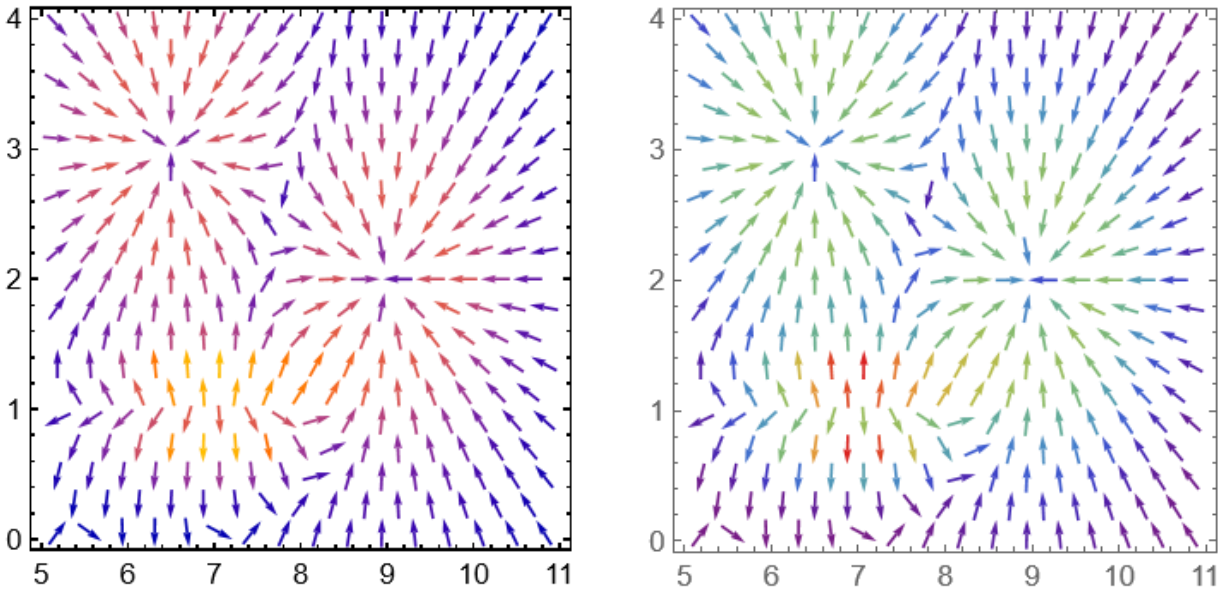


Рисунок 2.5 – Пример изменения цветовой схемы

На рисунке 2.6 показано применение маркера вектора типа капля VectorMarkers → "Drop".

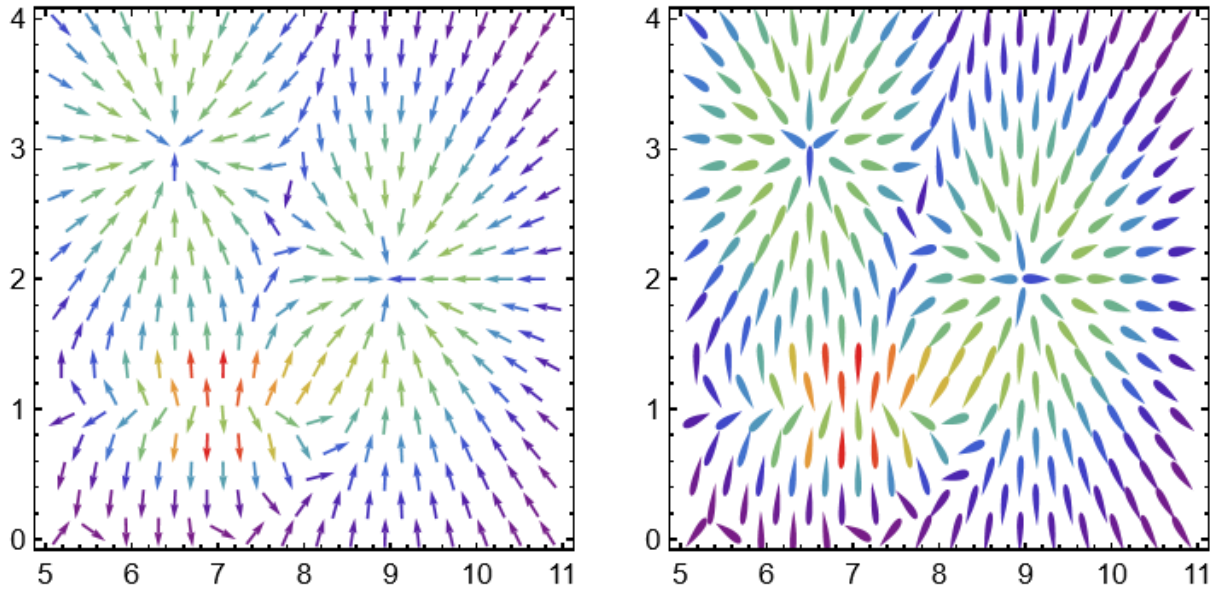


Рисунок 2.6 – Пример использования VectorMarkers

На рисунке 2.7 показано изменения размера векторов VectorSize−> 1.25.

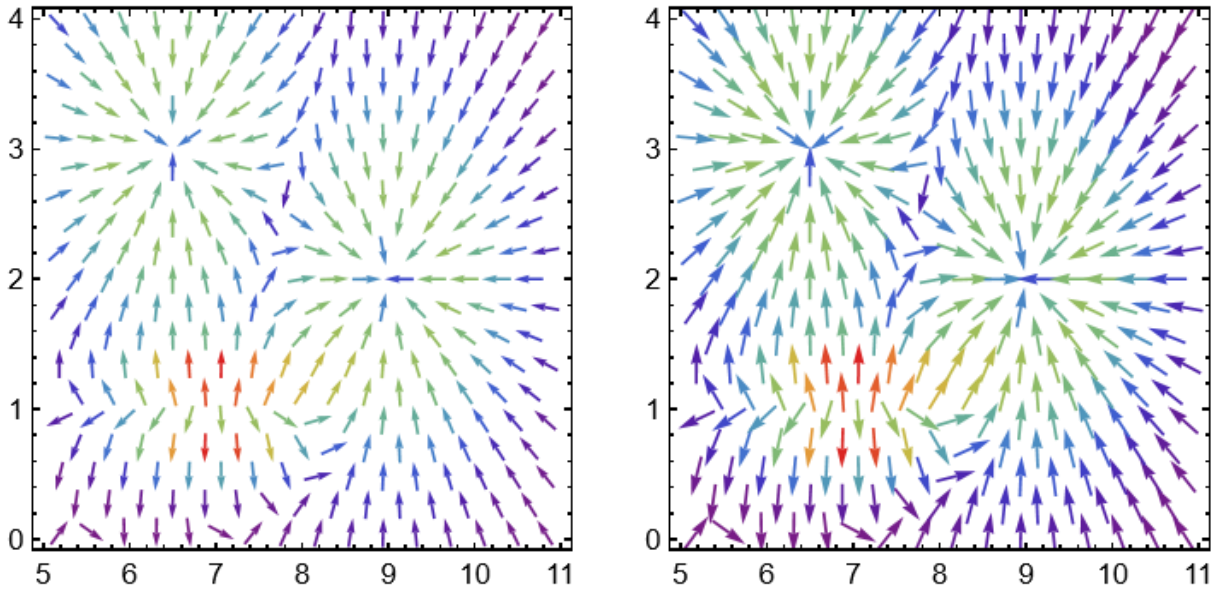


Рисунок 2.7 – Пример изменения размера векторов

2.3 Линии тока на плоскости. Линии тока на плоскости с фоном плотности функции

Приведём пример линий тока на плоскости, используя градиент функции (2.1), также изменим цвета векторов, StreamStyle−> Blue.

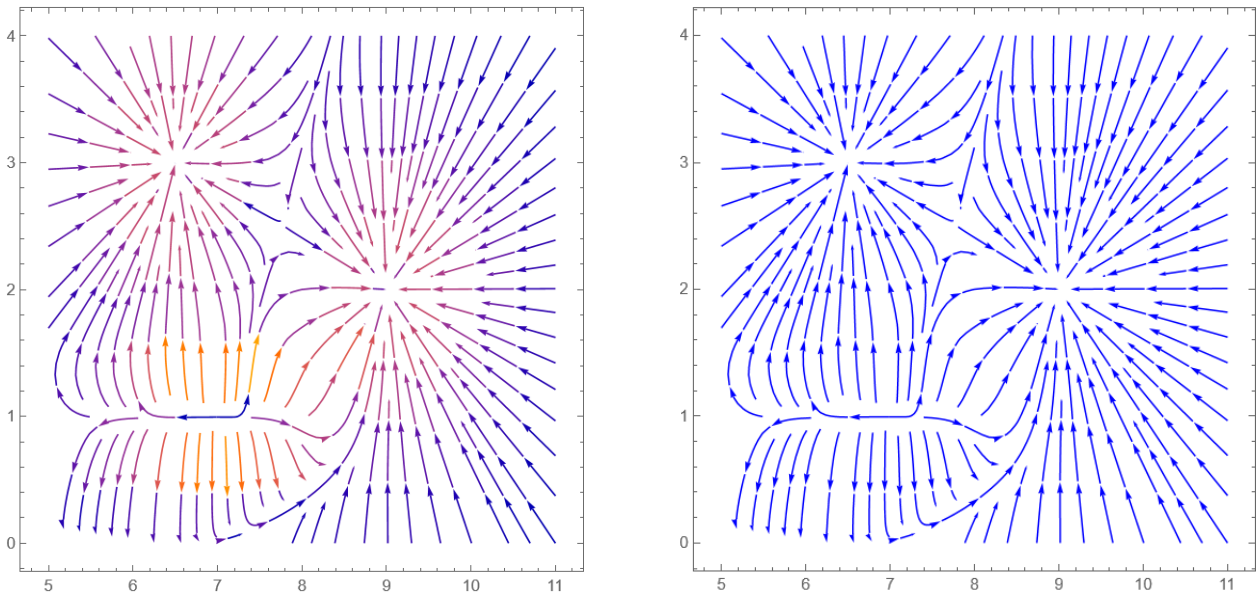


Рисунок 2.8 – Пример использования цветовой схемы для StreamPlot

На рисунке 2.9 приведены графики градиента функции (2.1), используя StreamPlot и StreamDensityPlot.

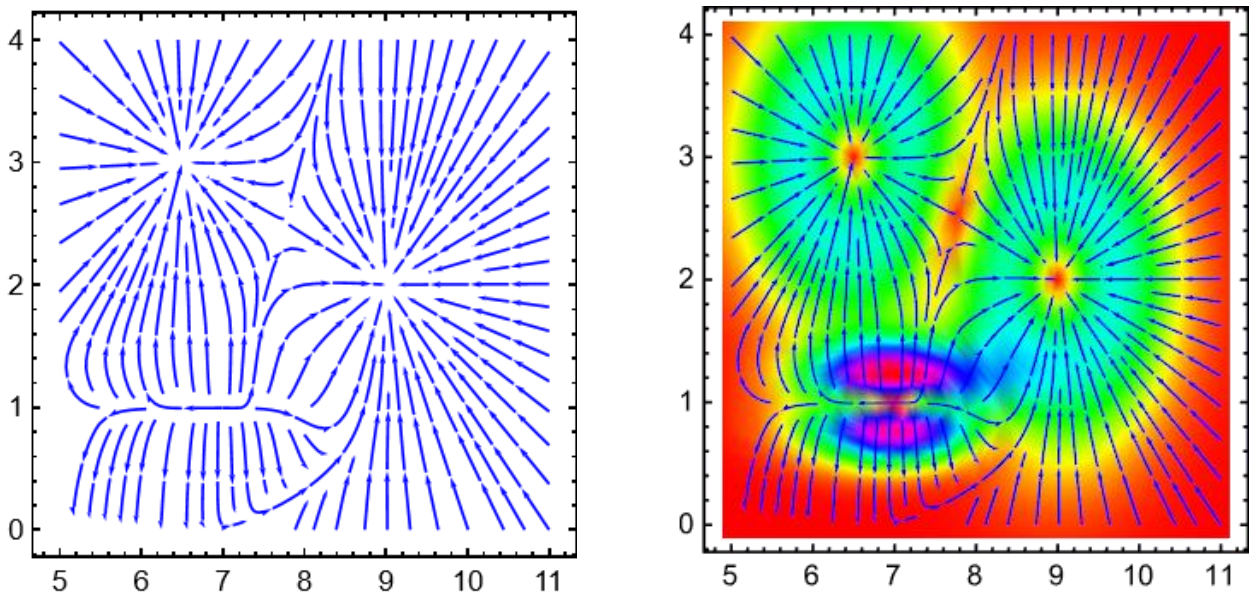


Рисунок 2.9 – Линии тока и плотность тока

Используем MaxRecursion для того, чтобы убрать дефекты, которые видны на рисунке 2.9

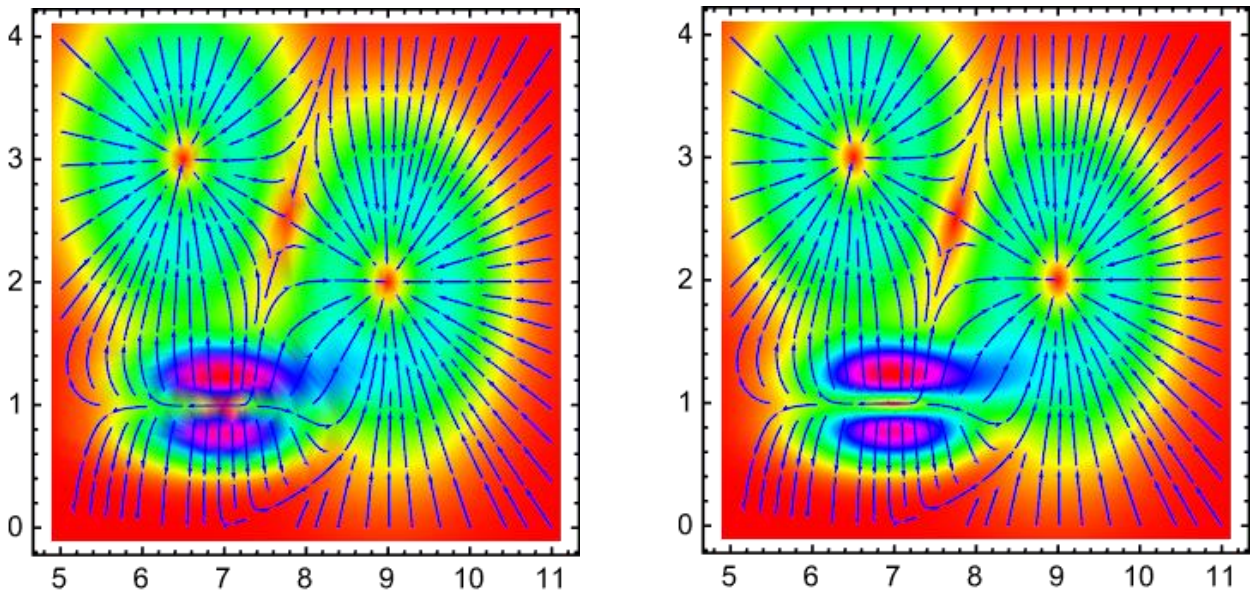


Рисунок 2.10 – Пример использования MaxRecursion

Попробуем разобраться какая функция находится в фоне на рисунке 2.10. Рассмотрим линии тока на плоскости с фоном плотности функции (2.1).

```
grSD1=StreamDensityPlot[gradf,{x,xMin,xMax},{y,yMin,yMax},StreamCo
lorFunction->None,StreamStyle->Blue,ColorFunction-
>Hue,MaxRecursion->5,ImageSize->200];

grSD2=StreamDensityPlot[{gradf,f[x,y]},{x,xMin,xMax},{y,yMin,yMax}
,StreamColorFunction->None,StreamStyle->Blue,ColorFunction-
>Hue,MaxRecursion->5,ImageSize->200];

grD1=DensityPlot[f[x,y],{x,xMin,xMax},{y,yMin,yMax},PlotPoints-
>50,ColorFunction->Hue,MaxRecursion->5,ImageSize->200];

Row[{grSD1,grSD2,grD1},Spacer[15]]
```

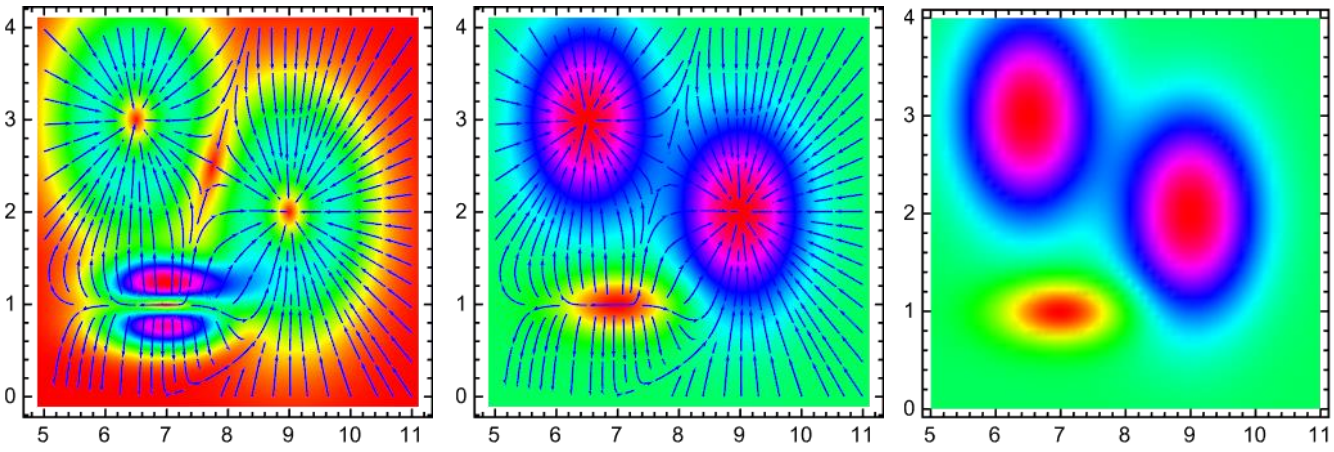


Рисунок 2.11

Сравнив графики на рисунке 2.11, можно прийти к выводу, что функция (2.1) не является фоном. Рассмотрим в качестве фона дивергенцию.

divf = D[gradient[[1]], x] + D[gradient[[2]], y]

```
grSD3=StreamDensityPlot[{gradf,divf},{x,xMin,xMax},{y,yMin,yMax},S
treamColorFunction->None,StreamStyle->Blue,ColorFunction-
>Hue,MaxRecursion->5,ImageSize->200];

grD2=DensityPlot[divf,{x,xMin,xMax},{y,yMin,yMax},ColorFunction-
>Hue,PlotPoints->50,MaxRecursion->5,ImageSize->200];

Row[{grSD1,grSD3,grD2},Spacer[15]]
```

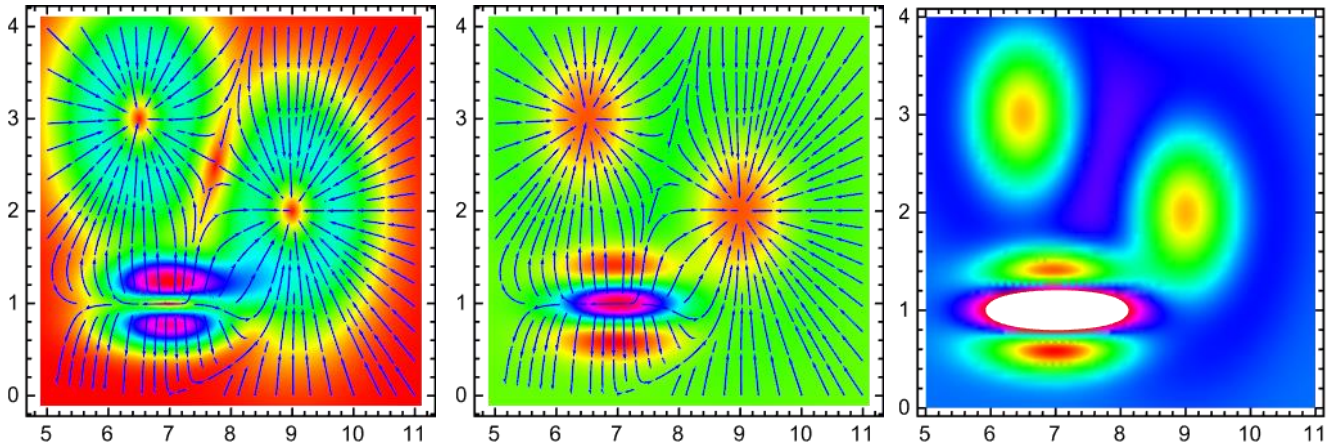


Рисунок 2.12

Из графиков на рисунке 2.12 видно, что дивергенция не является фоном. Рассмотрим норму градиента.

```
grSD4=StreamDensityPlot[{gradf,Norm[gradf]},{x,xMin,xMax},{y,yMin,
yMax},ColorFunction->Hue,ImageSize->200];

grD4=DensityPlot[Norm[gradf],{x,xMin,xMax},{y,yMin,yMax},ColorFunc
tion->Hue,ImageSize->200];

Row[{grSD1,grSD4,grD4},Spacer[15]]
```

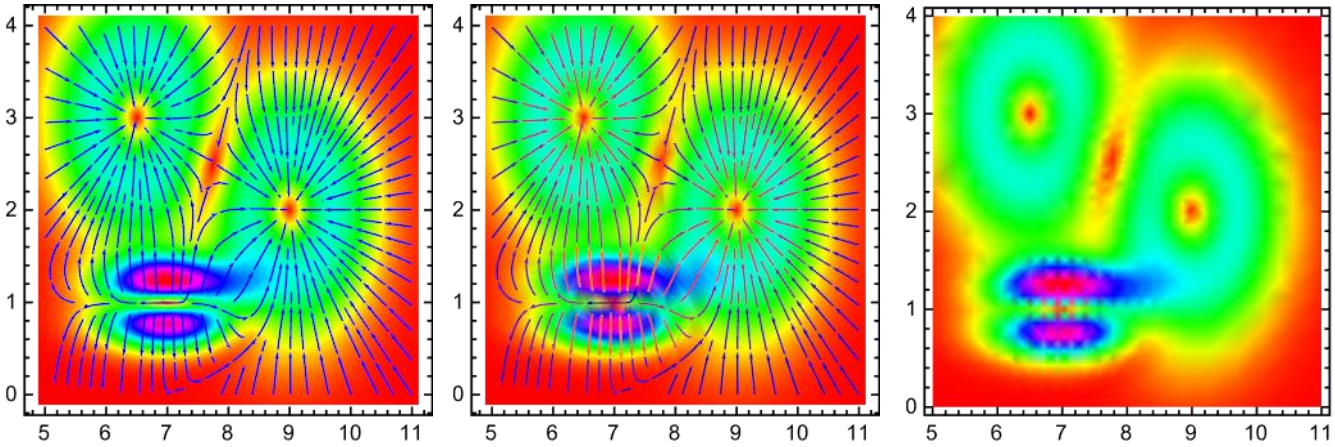


Рисунок 2.13

Исходя из рисунка 2.13, видно, что по умолчанию в качестве фона используется норма градиента.

Глава 3. ФОРМИРОВАНИЕ И ВИЗУАЛИЗАЦИЯ ВЕКТОРНЫХ ПОЛЕЙ НА JAVA

Для реализации алгоритмов формирования и визуализации векторных полей будем использовать библиотеку JavaFX [11]. JavaFX является мощным средством для разработки интерактивных графических пользовательских интерфейсов (GUI) на языке Java. По сравнению с другими платформами, такими как Swing, JavaFX является более современной и многофункциональной. При этом она сложнее, так как требует знаний продвинутых концепций, таких как графы сцен, стилизации с использованием CSS. Она предоставляет широкий набор инструментов по работе с мультимедиа (аудио, видео, изображения), двухмерной и трёхмерной графикой. JavaFX позволяет создавать приложения под различные операционные системы, такие как Windows, Linux, MacOS и для различных типов устройств: смартфон, планшет, компьютер, телевизор. JavaFX также обладает мощным API для работы с графикой, включая возможности для отрисовки векторных графиков, анимации и визуализации данных.

3.1 Компоненты JavaFX для визуализации

JavaFX предоставляет широкий набор инструментов, необходимых для визуализации векторных полей. Он использует следующие компоненты графического интерфейса:

1) Сцены и группы:

Прежде чем использовать контейнеры, необходимо использовать контейнер самого верхнего уровня, который называется сценой (Scene). Она является так называемым «каркасом», для остальных компонентов и необходима для того, чтобы на ней располагались эти самые компоненты. Group – это контейнер, который объединяет несколько графических элементов в один блок, который в последствии может быть добавлен на сцену.

2) Элементы рисования Canvas

Основой является компонент Canvas, который позволяет программно рисовать на поверхности, называемой холстом. Класс GraphicsContext описывает средства, которые позволяют рисовать на холсте:

- setFill(color) –устанавливает цвет заливки фигуры.
- setStroke(color) – устанавливает цвет обводки.
- strokeLine(x1, y1, x2, x2) – рисует прямую линию между двумя точками с координатами (x_1, y_1) и (x_2, y_2).

• fillRect(x, y, w, h) – рисует прямоугольник, в котором параметры x и y являются координатами левого верхнего угла прямоугольника, а w и h – ширина и высота соответственно.

3) 2D - графика

JavaFX поддерживает как двухмерную так и трёхмерную графику. Двухмерная графика содержит геометрические фигуры, которые применяются для рисования. Есть такие фигуры как: прямая линия, дуга, ломаная линия, прямоугольник, эллипс, круг, многоугольник. Каждый примитив обладает свойствами такими как: цвет заполнения, цвет контура, толщина линии, прозрачность, видимость, угол поворота, эффекты и другие.

4) Работа с цветом

Работа с цветом осуществляется с помощью цветовой палитры, в которой можно выбрать конкретный цвет из доступного диапазона или задать собственный цвет, указав комбинацию RGB, HSB или WEB.

5) Система координат

JavaFX использует стандартную декартову систему координат, где начало координат находится в верхнем левом углу окна, ось X направлена вправо, а ось Y направлена вниз.

6) Трансформация и эффекты

Трансформация – это смещение, вращение и масштабирование элементов холста. Это играет особую роль при анализе векторного поля. Помимо трансформации, к узлу можно применить различные эффекты, например, добавить тень, в том числе внешнюю и внутреннюю тень, сделать зеркальное отображение, применить различные виды размытей, добавить свечение, изменить цветовой тон, насыщенность, яркость, контраст и многое другое.

3.2 Формирование векторного поля

Рассмотрим функцию $f(x,y)$, заданную на множестве A, где $A = \{(x,y): 5 \le x \le 11, 0 \le y \le 4\}$.

$$f(x,y) = e^{-(x-9)^2-(y-2)^2} - \frac{2}{3}\, e^{-(x-7)^2-(3\,y-3)^2} + e^{-(x-6.5)^2-(y-3)^2} \qquad (3.1)$$

Визуализируем функцию (3.1). Код для визуализации можно увидеть в приложении А.:

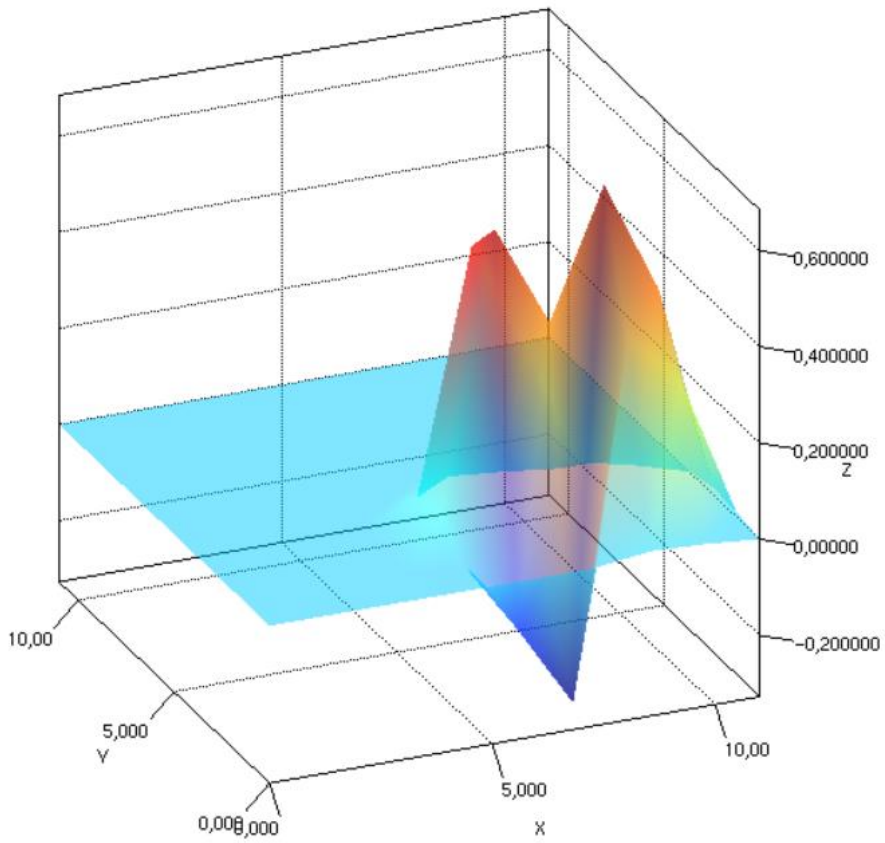


Рисунок 3.1 – Визуализация функции (3.1)

Увеличим количество начальных точек для того, чтобы сгладить функцию. Результат сглаживания представлен на рисунке 3.2.

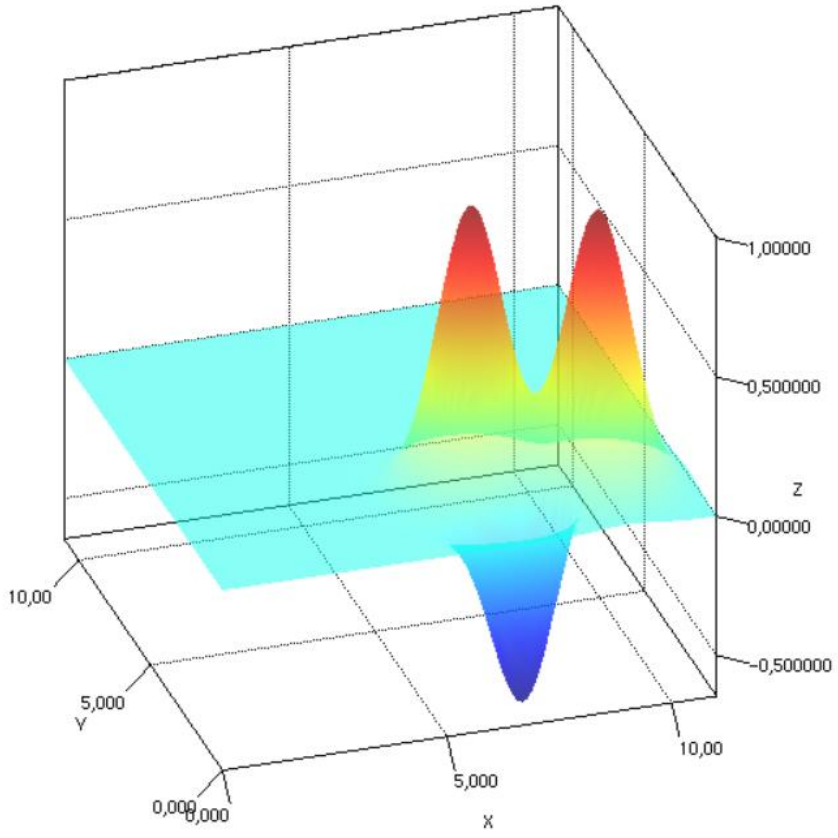


Рисунок 3.2 – Визуализация функции (3.1) с увеличенным числом начальных точек

Векторное поле будем строить как градиент функции (3.1), то есть

$$\text{gradf} = D\left[f[x,y],\{\{x,y\}\}\right] = \{\partial f/\partial x, \partial f/\partial y\} \quad (3.2)$$

Используя функционал JavaFX, зададим оси координат, деления на них. В цикле каждый раз будем вычислять градиент функции (3.2) и рисовать вектор. Код можно увидеть в приложении Б. Таким образом получается векторное поле:

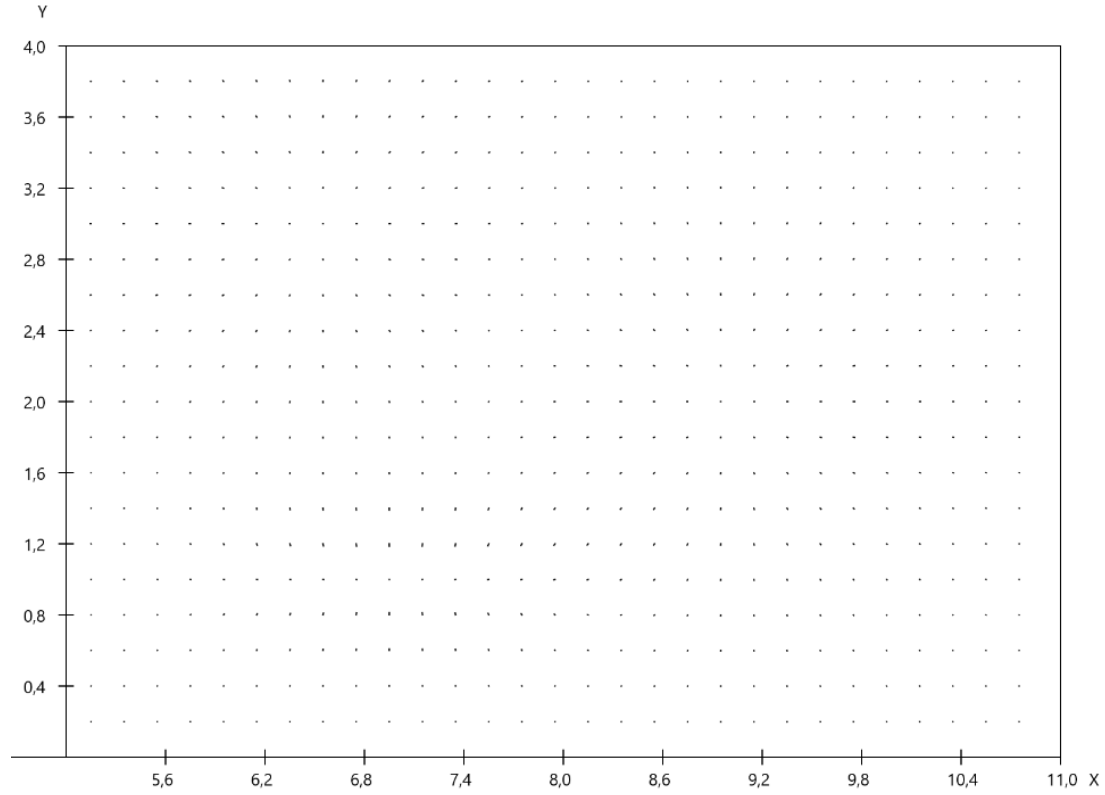


Рисунок 3.3 – Векторное поле функции (4.1)

Увеличим размер векторов, умножив градиент функции (3.2) на константу.

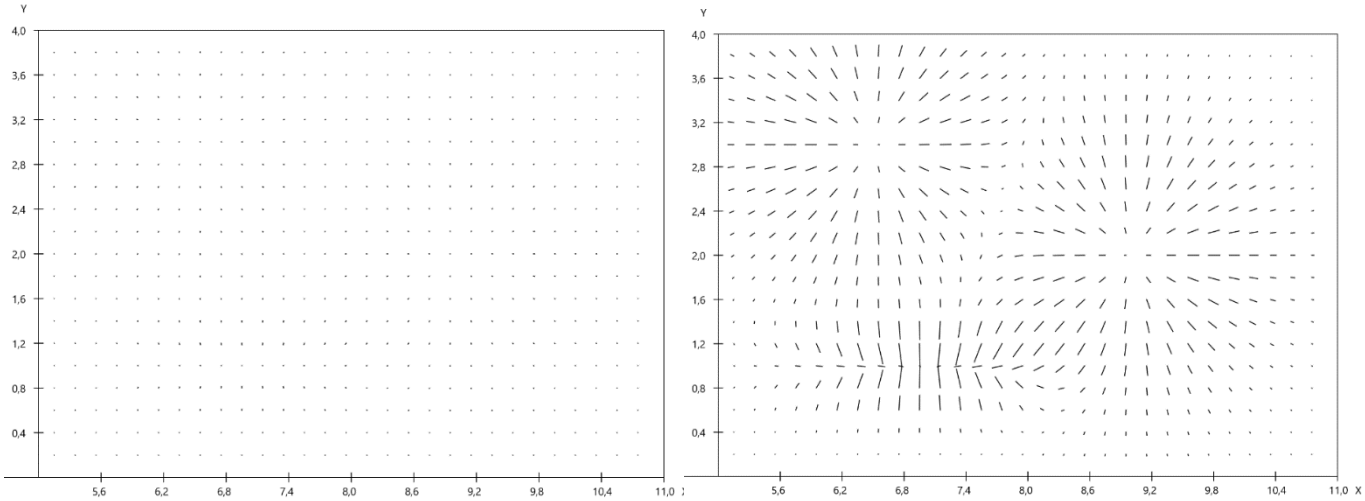


Рисунок 3.4 – Изменение размера векторов

3.3 Изменение цвета и добавление наконечника

Для того чтобы вектора имели одинаковый размер, нормируем их:

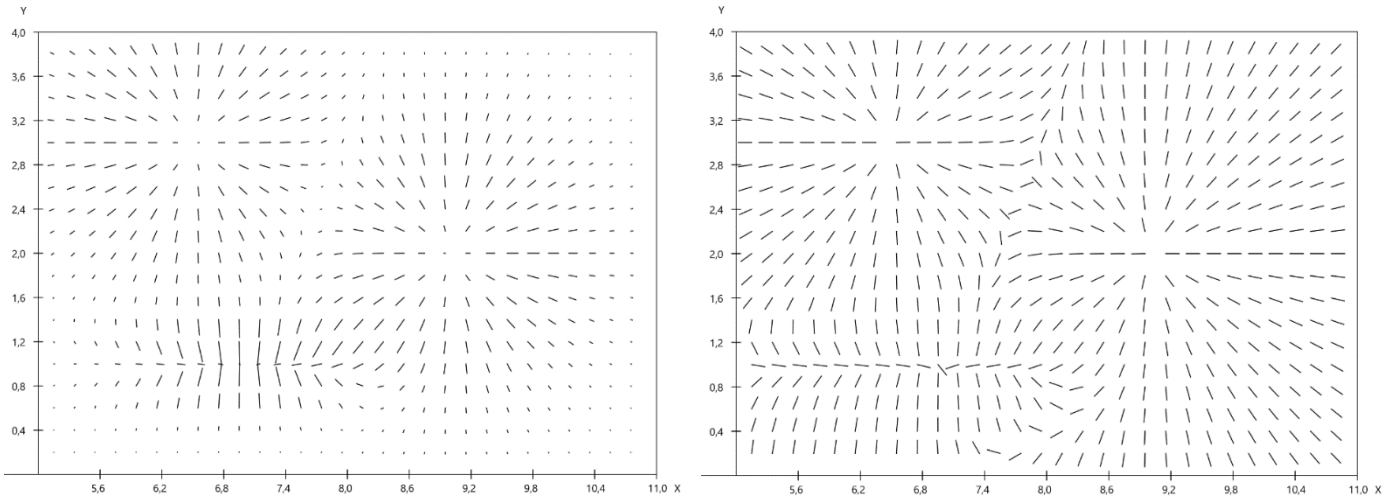


Рисунок 3.5 – Изменение размера векторов

Изменим цвет векторов с помощью функции getStroke().

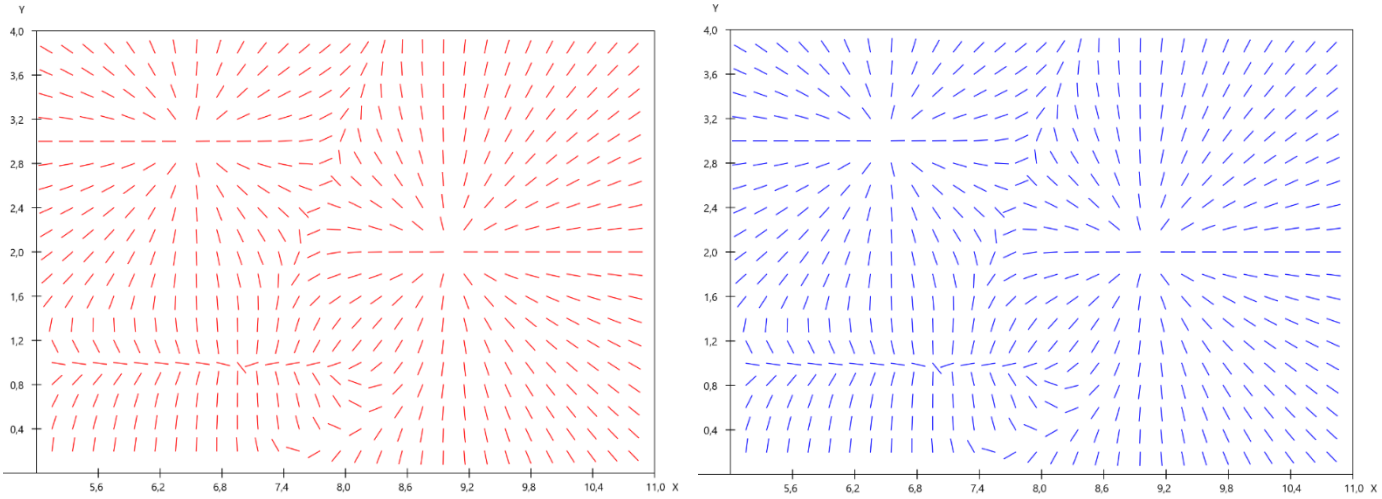


Рисунок 3.6 – Пример изменения цвета векторов

Для того, чтобы добавить наконечник на вектора, применим метод drawArrowhead(). Он определяет угол между начальной и конечной точками вектора, используя функцию вычисления арктангенса по формуле $arctan(y_2 - y_1, x_2 - x_1)$.

На основе найденного угла и координат начальной точки рассчитаем координаты вершин наконечника. В результате получается визуализация векторного поля, изображённая на рисунке 3.7.

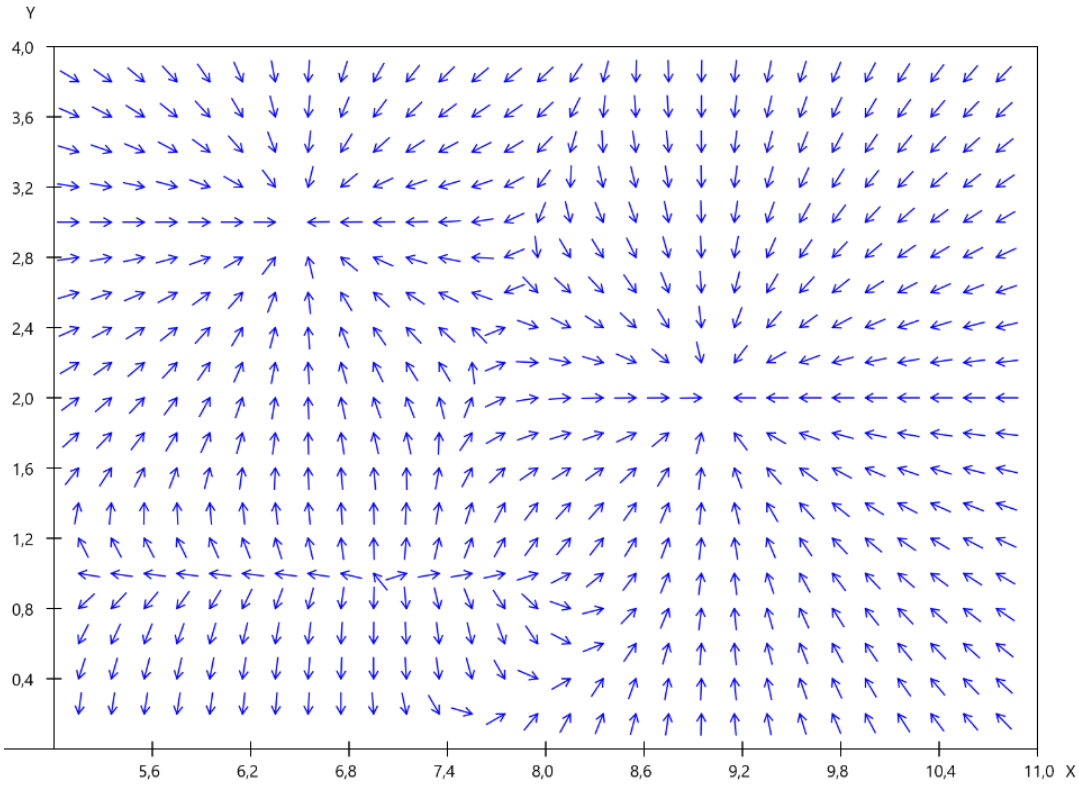


Рисунок 3.7 – Векторное поле функции (3.1)

Глава 4. ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ

4.1 Определение параллельных вычислений

Параллельные вычисления представляют собой метод обработки данных, при котором несколько операций выполняются одновременно. В отличие от последовательного способа, где действия следуют одно за другим, в параллельном подходе задача разделяется на части, которые обрабатываются параллельно, либо на разных процессорах, либо на нескольких ядрах одного процессора.

Главное преимущество такого подхода заключается в заметном увеличении производительности. Такой подход позволяет значительно ускорить работу с большими объёмами данных и сложными расчётами. Особенно это актуально в задачах вроде численного моделирования или анализа больших массивов информации, где прирост скорости может достигать в разы по сравнению с последовательными методами

Ещё одно важное преимущество — возможность разбить сложную задачу на более простые и независимые фрагменты, что называется декомпозицией. Это упрощает как сам процесс расчётов, так и разработку алгоритмов: каждую подзадачу можно отдельно протестировать, отладить и при необходимости оптимизировать

Кроме того, параллельные вычисления отличаются масштабируемостью. Современные системы — например, кластеры на базе MPI или платформы вроде Apache Spark — позволяют легко наращивать вычислительные мощности, подключая дополнительные узлы или процессоры. Это особенно полезно, когда объёмы обрабатываемых данных быстро растут или задачи становятся сложнее

Нельзя не отметить и то, что параллельные вычисления помогают повысить точность итоговых результатов. Благодаря возможности обрабатывать большие объёмы информации и использовать более детализированные модели, такие алгоритмы дают более надёжные и точные данные. Это крайне важно в научных работах, инженерных расчётах и других областях, где от результата зависит многое.

4.2 Параллельные вычисления в Wolfram Mathematica

В среде Wolfram Mathematica [12] реализована поддержка параллельных вычислений, которая позволяет задействовать несколько процессоров или ядер для выполнения сложных задач. Mathematica может работать в разных программных средах, предоставляя возможности для параллельного выполнения задач с применением функционального подхода. Архитектура

системы предусматривает автоматическое распределение вычислительных задач между доступными ядрами, управление памятью, а также использование виртуальной разделяемой памяти с поддержкой синхронизации и блокировок.

Дополнительно Mathematica умеет автоматически перераспределять задачи в случае возникновения ошибок на отдельных ядрах, что позволяет продолжать вычисления без потерь. Система минимизирует задержки и обеспечивает обмен как данными, так и программными конструкциями и символическими выражениями между локальными и удалёнными вычислительными узлами.

Основные функции, которые используются для работы с параллельными вычислениями в Wolfram Mathematica:

- Parallelize – распределяет выполнение вычислений между ядрами, для ускорения работы программы.
- ParallelTry – запускает несколько вариантов вычисления параллельно, выдавая первый получившийся результат.
- ParallelEvaluate – вычисляет выражение на доступных ядрах и возвращает список результатов полученных результатов.
- ParallelMap – применяет заданную функцию к каждому элемента списка параллельно.
- ParallelTable – создаёт список параллельно из заданного количества элементов и вычисляет каждый элемент независимо от других.
- ParallelSum – параллельно вычисляет сумму.
- $KernelCount – показывает количество запущенных ядер в данный момент.
- LaunchKernels – запускает заданное количество ядер.
- CloseKernels – закрывает все параллельные ядра.
- $ConfiguredKernels – возвращает список ядер, которые настроены по умолчанию для удаленных или параллельных вычислений.

Система компьютерной алгебры Wolfram Mathematica позволяет значительно ускорить выполнение операций за счёт эффективного распределения нагрузки между ядрами.

Для измерения времени выполнения кода в Wolfram Mathematica используются функции Timing и AbsoluteTiming, каждая из которых имеет свои особенности. Timing измеряет время, затраченное процессором на выполнение выражения. Эта функция учитывает только активное использование процессора, игнорируя задержки, связанные с операциями ввода-вывода. Функция AbsoluteTiming измеряет общее время, прошедшее с момента начала выполнения выражения до его завершения. Этот показатель включает не только активное использование процессора, но и внешние задержки, такие как ввод-вывод, синхронизацию потоков и другие факторы. В дальнейшем будем использовать именно AbsoluteTiming.

Проведём последовательное и параллельное вычисление суммы $\sum_{i=1}^{1\,000\,000} i^5$ в системе Wolfram Mathematica. Результаты измерения времени вычисления приведены на рисунках 4.1 и 4.2.

```
In[1]:= Sum[i^5, {i, 1, 1000000}] // AbsoluteTiming
Out[1]= {8.38381, 166667166667083333333333250000000000}
```

Рисунок 4.1 – Функция Sum

```
In[1]:= ParallelSum[i^5, {i, 1, 1000000}] // AbsoluteTiming
Out[1]= {6.42353, 166667166667083333333333250000000000}
```

Рисунок 4.2 – Функция ParallelSum

Можно заметить, что оба результата вычисления суммы получились идентичными, однако время, затраченное на вычисления, оказалось разным. При использовании последовательного алгоритма потребовалось 8.38381 секунд, а при вычислении с помощью параллельного алгоритма — 6.42353 секунды.

Для оценки эффективности параллельных вычислений используется показатель ускорения S, который определяется как отношение времени, затраченного на выполнение задачи, используя последовательный алгоритм, к времени, затраченному на выполнение задачи, используя параллельный алгоритм.

$$S = \frac{T_{послед}}{T_{паралл}} \tag{4.1}$$

В данной задаче, показатель ускорения $S \approx 1.30517$, и это означает, что при использовании параллельного алгоритма, задача решается почти в 1.30517 раз быстрее, чем при использовании последовательного.

Процессоры играют очень важную роль в производительности систем. При увеличении количества процессоров, работающих параллельно, время выполнения задач может сократиться. Однако, сокращение времени не всегда линейно. Это связано с накладными расходами на синхронизацию данных и управление задачами. Из-за необходимости координации между процессорами, при добавлении дополнительного процессора не всегда может приводить к пропорциональному уменьшению времени.

Рассмотрим пример с вычислением суммы от 1 до 1000000 i^5, используя разное количество процессоров, будет считать ускорение, полученное в результате вычисления суммы параллельным алгоритмом.

Количество процессоров	Время, сек	Ускорение
1	9.22868	0.69604
2	4.90769	1.30887
3	3.86666	1.66126
4	3.10436	2.06920
5	2.57562	2.49397
6	2.43441	2.63864

Чем на большее количество процессоров мы делим задачу, тем меньше времени уходит на вычисление суммы, а скорость работы растёт. Результаты эксперимента наглядно демонстрируют преимущества параллельного алгоритма при расчёте суммы. Важно понимать, если процессоров станет слишком много, прирост скорости замедлится, это возникнет из-за дополнительных затрат на их взаимодействие и ограничение самой системы. Лучшие результаты получатся тогда, когда число процессоров выбрано с умом.

4.3 Характеристики векторного поля

Рассмотрим векторное поле $v(x, y, z)$, заданное следующими компонентами:

$$v_x(x, y, z) = -y * e^{-(x^2+y^2+z^2)}, v_y(x, y, z) = x * e^{-(x^2+y^2+z^2)}, v_z(x, y, z) = z * e^{-(x^2+y^2+z^2)} \quad (4.2)$$

Это поле заставляет частицы вращаться вокруг оси z, причём чем дальше от центра, тем слабее становится это вращение.

Чтобы лучше понять его структуру и динамику, мы визуализируем его с помощью VectorPlot3D и StreamPlot3D. VectorPlot3D показывает направление и величину векторов в разных точках пространства, а StreamPlot3D рисует траектории движения частиц, помогая увидеть общую картину их поведения.

На рисунке 4.3 представлен график, полученный в результате использования функции VectorPlot3D.

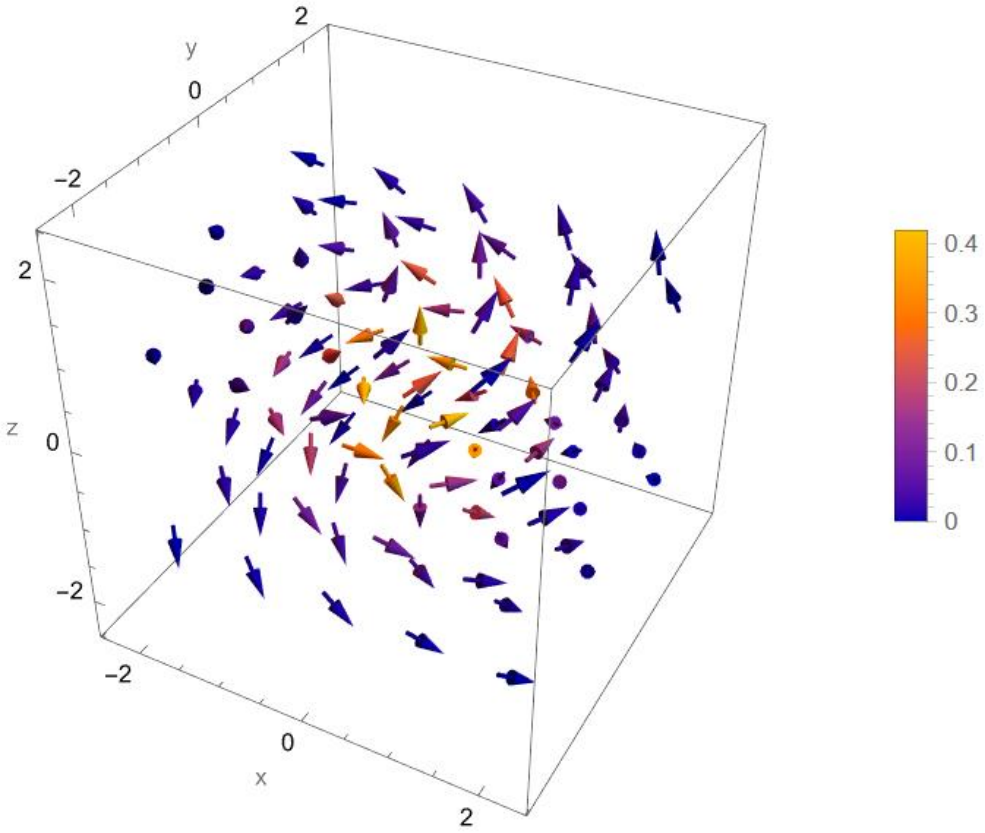


Рисунок 4.3 – Визуализация векторного поля с помощью VectorPlot3D

На рисунке 4.4 представлен график, полученный в результате использования функции StreamPlot3D.

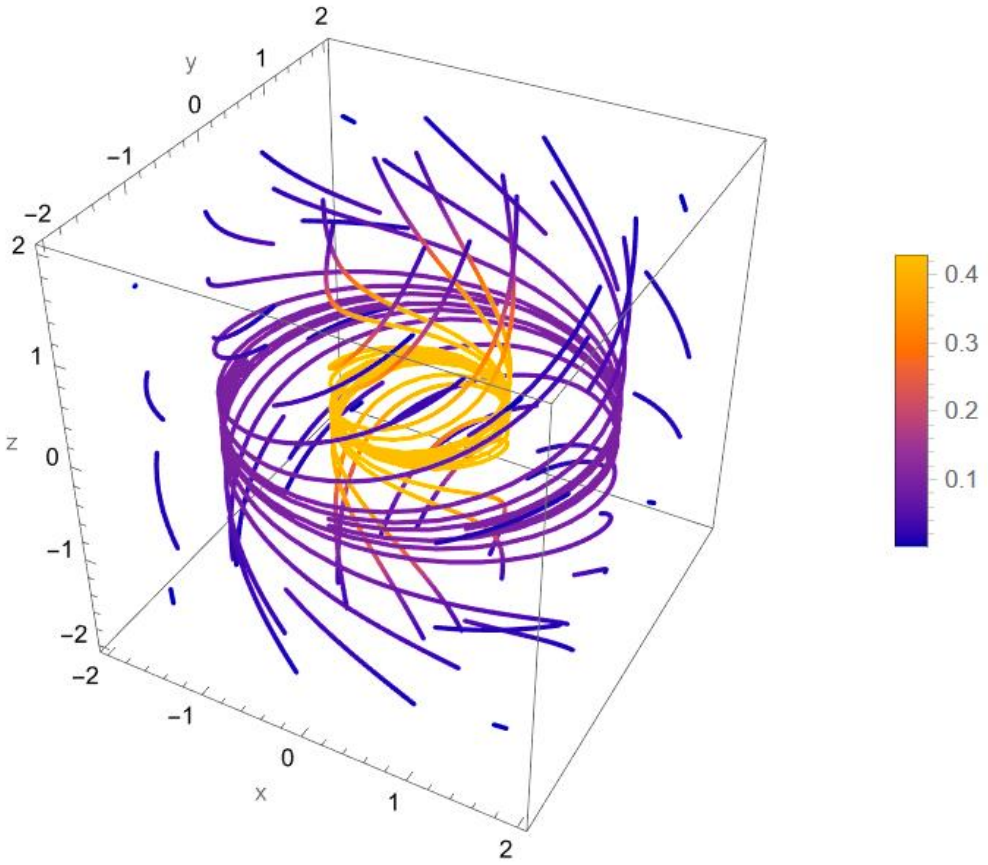


Рисунок 4.4 – Визуализация векторного поля с помощью StreamPlot3D

Рассмотрим основные свойства векторных полей, введём понятия дивергенции и ротора векторного поля $v(x, y, z) = \{v_x, v_y, v_z\}$.

• Дивергенция векторного поля v – это скалярная величина, показывающая, насколько сильно поле расходится из данной точки. Она измеряет плотность источников или стоков векторного поля. Математически дивергенция векторного поля выражается как:

$$div(v) = \frac{dv_x}{dx} + \frac{dv_y}{dy} + \frac{dv_z}{dz}, \tag{4.3}$$

где v_x, v_y, v_z – компоненты векторного поля v.

• Ротор векторного поля v – это векторная величина, показывающая направление и величину вращения векторного поля вокруг данной точки. Он определяет меру закрученности поля и направление, вокруг которого происходит это вращение. Математически ротор определяется следующим образом:

$$rot(v) = (\frac{dv_z}{dy} - \frac{dv_y}{dz})i + (\frac{dv_x}{dz} - \frac{dv_z}{dx})j + (\frac{dv_y}{dx} - \frac{dv_x}{dy})k, \tag{4.4}$$

где i, j, k – ортонормированный базис в пространстве.

Вычислим дивергенцию и ротор для векторного поля $v(x, y, z)$.

4.4 Параллельные вычисления векторного поля

Рассмотрим векторное поле $v(x, y, z)$, заданное компонентами (4.1). Будем вычислять дивергенцию и ротор на трёхмерной сетке.

Код для задания векторного поля, визуализации и реализации всех использованных алгоритмов, включая ParallelMap, ParallelTable и Parallelize, приведён в приложении В. Вся дальнейшая работа основывается на этом коде.

Для начала зададим трёхмерную сетку в области $x, y, z \in [-2, 2]$ с шагом $h = 0.1$.

В каждой точке заданной сетки найдём дивергенцию и ротор векторного поля $v(x, y, z)$, с применением формул (4.3) и (4.4). Для расчётов были выбраны три метода параллельных вычислений: ParallelMap, ParallelTable, Parallelize. Полученные результаты приведены в таблицах 4.1, 4.2 и 4.3.

Таблица 4.1 Результаты работы метода ParallelMap

Характеристика	Последовательный алгоритм, сек	Параллельный алгоритм, сек	Ускорение
Дивергенция	7.696	3.21194	2.39606
Ротор	82.1366	21.1377	3.88578

Применение метода ParallelMap показало довольно хорошее ускорение вычислений как для ротора, так и для дивергенции.

Таблица 4.2 Результаты работы метода ParallelTable

Характеристика	Последовательный алгоритм, сек	Параллельный алгоритм, сек	Ускорение
Дивергенция	0.0377481	2.45552	0.0153728
Ротор	77.8426	2.51524	30.9483

Применение метода ParallelTable также продемонстрировало хорошее ускорение при вычислении ротора. Но для дивергенции выигрыш в производительности гораздо менее заметен. Низкое ускорение для дивергенции может быть связано с накладными расходами при распараллеливании, что при малых объёмах данных является значительным.

Таблица 4.3 Результаты работы метода Parallelize

Характеристика	Последовательный алгоритм, сек	Параллельный алгоритм, сек	Ускорение
Дивергенция	0.0427962	3.16977	0.0135013
Ротор	83.0746	24.7574	3.35554

Метод Parallelize показал неплохое ускорение для ротора, но для дивергенции ускорение также, как и при использовании метода ParallelTable оказалось менее значительным.

Таким образом метод ParallelMap демонстрирует более стабильное и хорошее ускорение. Для задач, связанных с обработкой больших данных можно использовать методы ParallelMap, ParallelTable и Parallelize. При выполнении задач с малым объёмом вычислений следует учитывать накладные расходы и тщательно выбрать метод распараллеливания задачи.

Глава 5. МОДЕЛИРОВАНИЕ ВОЛН ЦУНАМИ

Цунами – это серия волн в море, вызванная из-за подводных землетрясений, извержений подводных вулканов, подводных оползней, взрывов в воде, падений в воду обломков скал и т.п. Цунами входит в том самых разрушительных природных явлений, представляющее серьёзную опасность не только для прибрежных зон, но и для территорий, расположенных вглубь суши. Проведение моделирования цунами имеет важное значение для оценки потенциального ущерба, разработки систем раннего оповещения и обеспечения безопасности населения. Такие исследования позволяют лучше понять природу возникновения и распространения волн, а также прогнозировать их дальнейшее развитие.

Одной из главных задач в изучении цунами остаётся точный прогноз их поведения и моделирование распространения волн с учётом особенностей подводного рельефа и текущих океанографических условий.

Процессы, связанные с цунами, достаточно сложны и описываются с использованием дифференциальных уравнений в частных производных. Для качественного моделирования подобных явлений требуются эффективные алгоритмы, способные с высокой точностью решать подобные уравнения и обеспечивать наглядное представление результатов в виде векторных и скалярных цифровых полей.

5.1 Методология исследования

Цель исследования – выбор и применение инструментов Wolfram Mathematica решения дифференциальных уравнений в частных производных, описывающих распространение волн цунами, реализация различных вариантов визуализации векторных и скалярных полей.

Уравнения мелкой воды запишем в следующем виде [12]:

$$\frac{1}{R\cos(\phi)}\left(\frac{\partial(u(\lambda,\phi,t)(h(\lambda,\phi,t)-b(\lambda,\phi)))}{\partial\lambda}+\frac{\partial(\cos(\phi)v(\lambda,\phi,t)(h(\lambda,\phi,t)-b(\lambda,\phi)))}{\partial\phi}\right)+\frac{\partial h(\lambda,\phi,t)}{\partial t}=0 \quad (5.1)$$

,

$$\frac{1}{R\cos(\phi)}\left(g\frac{\partial h(\lambda,\phi,t)}{\partial\lambda}+u(\lambda,\phi,t)\frac{\partial u(\lambda,\phi,t)}{\partial\lambda}\right)+\frac{v(\lambda,\phi,t)}{R}\frac{\partial u(\lambda,\phi,t)}{\partial\phi}+\frac{\partial u(\lambda,\phi,t)}{\partial t}=fv(\lambda,\phi,t), \quad (5.2)$$

$$\frac{u(\lambda,\phi,t)}{R\cos(\phi)}\frac{\partial v(\lambda,\phi,t)}{\partial\lambda}+\frac{1}{R}\left(g\frac{\partial h(\lambda,\phi,t)}{\partial\phi}+v(\lambda,\phi,t)\frac{\partial v(\lambda,\phi,t)}{\partial\phi}\right)+\frac{\partial v(\lambda,\phi,t)}{\partial t}=-fu(\lambda,\phi,t), \quad (5.3)$$

где $h(\lambda,\phi,t)$ – высота воды в точке (λ,ϕ) и времени t, $u(\lambda,\phi,t)$ – горизонтальная скорость воды в направлении долготы в точке (λ,ϕ) и времени t, $v(\lambda,\phi,t)$ –

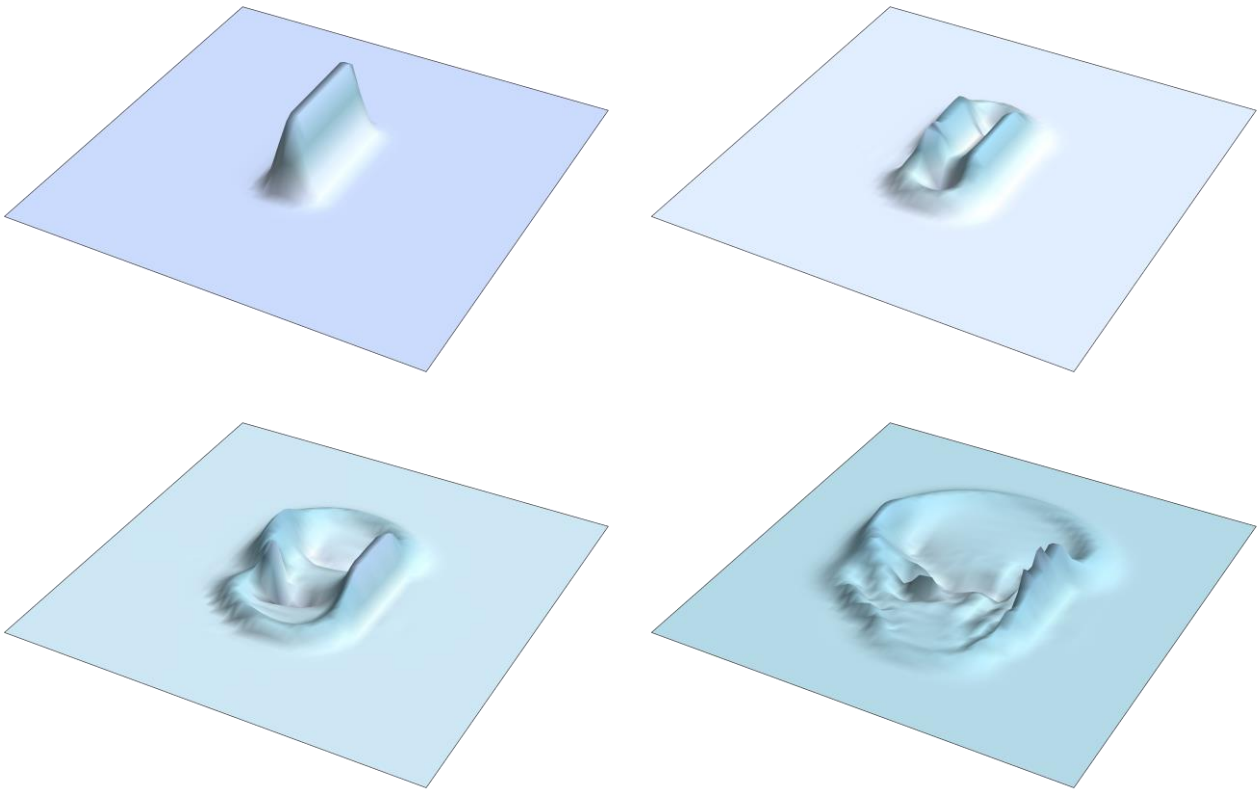
горизонтальная скорость воды в направлении широты в точке (λ,ϕ) и времени t, $b(\lambda,\phi)$ – высота рельефа дна, R – радиус Земли, g – ускорение свободного падения, f – некоторый параметр.

Для решения дифференциальных уравнений (5.1) – (5.3) будем использовать функцию DSolve, настроенную на метод линий, позволяющую представить исходные уравнения в виде системы обыкновенных дифференциальных уравнений, для упрощения решения задачи и повышения точности [7, 14]. Так же будем использовать метод Адамса для численного интегрирования, обеспечивая требуемую точность при решении обыкновенных дифференциальных уравнений.

Визуализацию полученного решения будем осуществлять с использованием инструментов Wolfram Mathematica, что позволит наглядно представить результаты моделирования.

5.2 Результаты

Для представления динамики поверхности воды в ходе распространения волн цунами была использована анимация на основе функции Plot3D [8, 9]. Данный метод позволил визуализировать изменение формы волны во времени. На рисунке 5.1 представлены виды формы поверхности воды в различные моменты времени t, что демонстрирует эволюцию волны и её амплитудные характеристики.



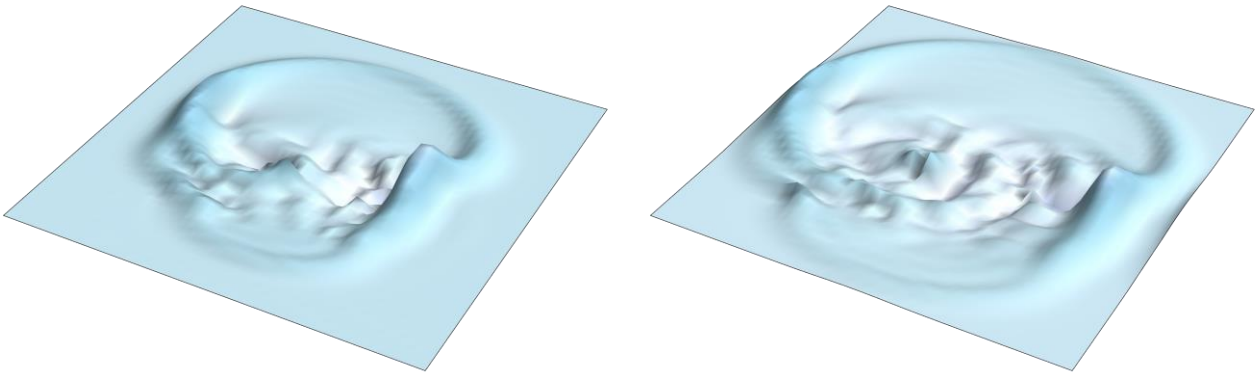


Рисунок 5.1 – 3D визуализация поверхности цунами в моменты времени t =1; 4; 7; 12; 16 и 20 с

Для анализа динамики волн воды было построено векторное поле, отражающее горизонтальные компоненты скорости в каждой точке пространства. На рисунке 5.2 представлено векторное поле скорости воды [15], которое демонстрирует направления движения и интенсивность потоков вдоль поверхности, то есть показывает горизонтальные проекциям полного вектора скорости.

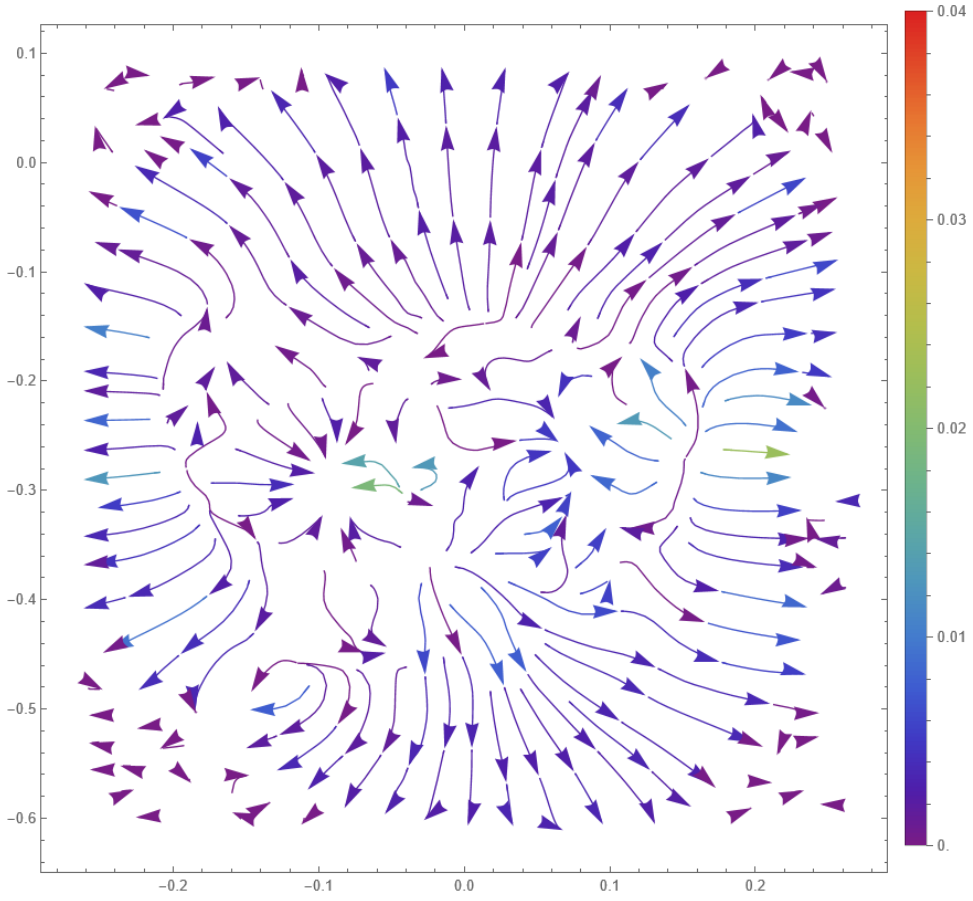
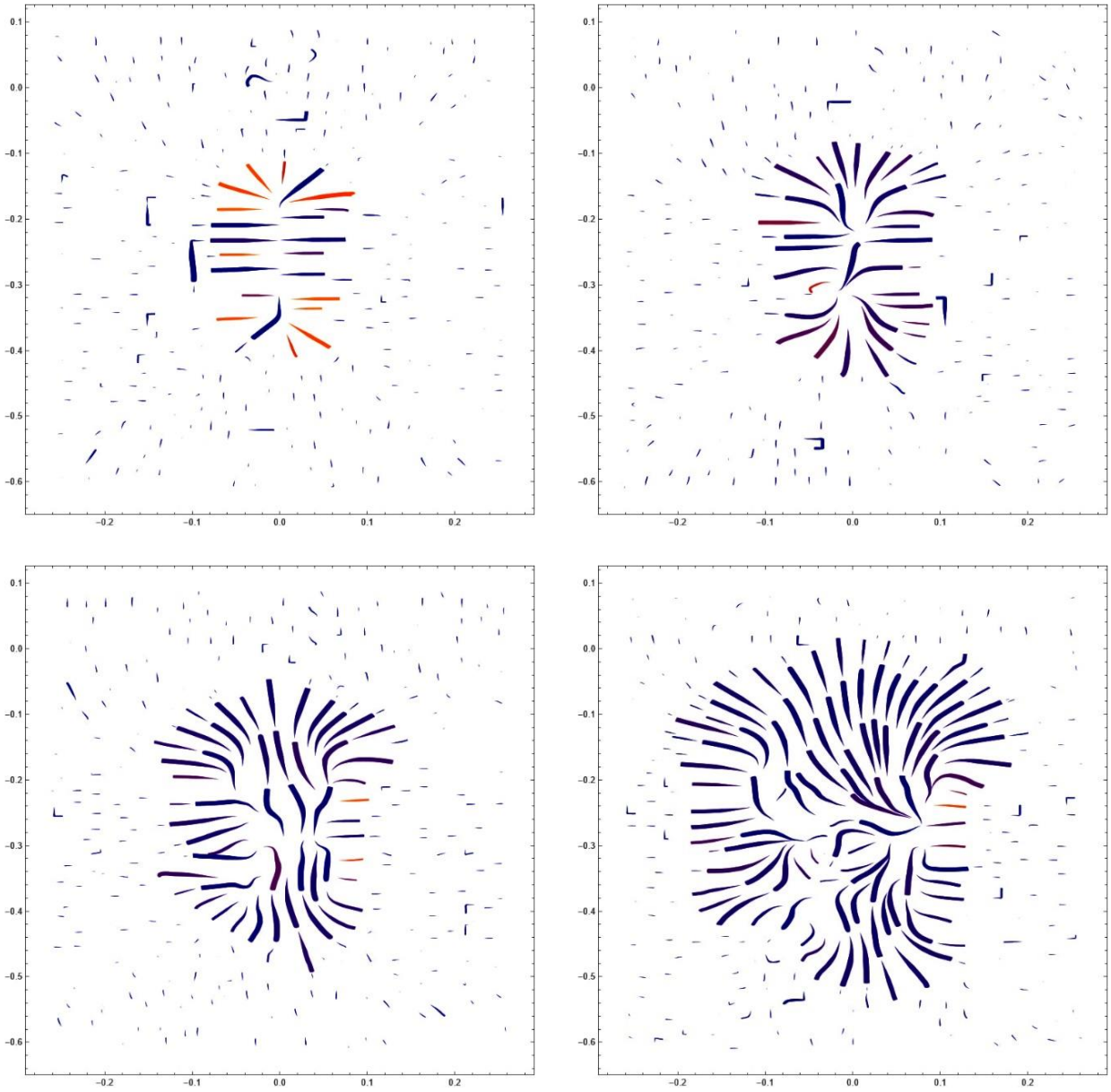


Рисунок 5.2 – Векторное поле касательных проекций скорости воды вдоль поверхности

На рисунке 5.3 представлены линии тока, иллюстрирующие распределение векторного поля скорости воды в различные моменты времени. График позволяет визуально оценить направление и интенсивность течений, возникающих в процессе моделирования цунами.



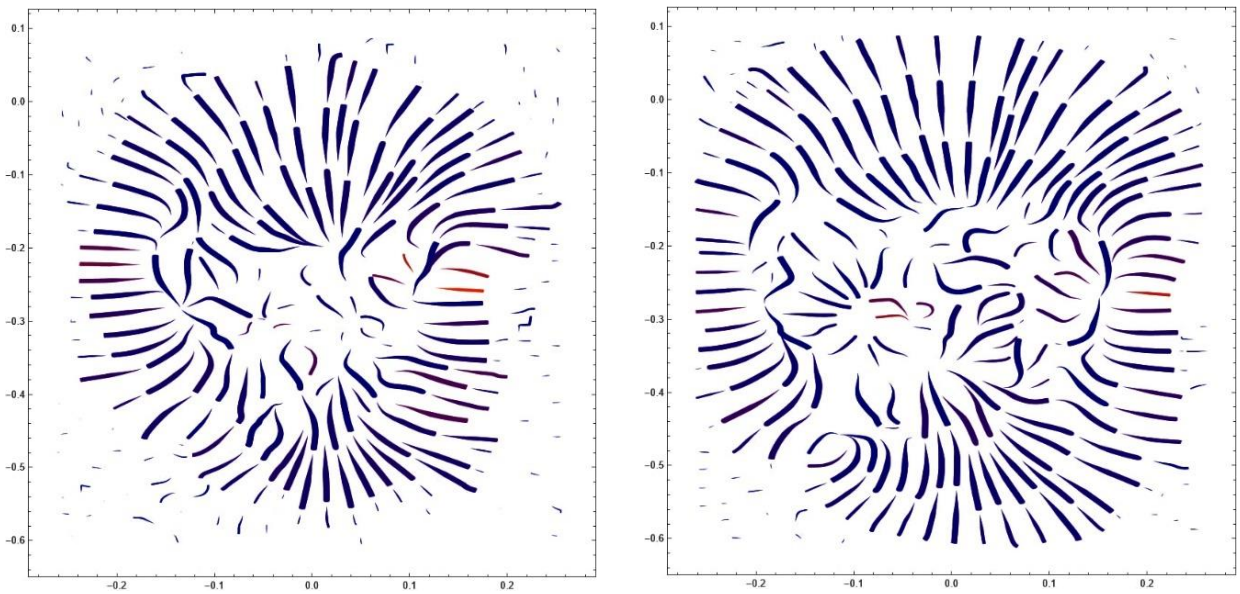
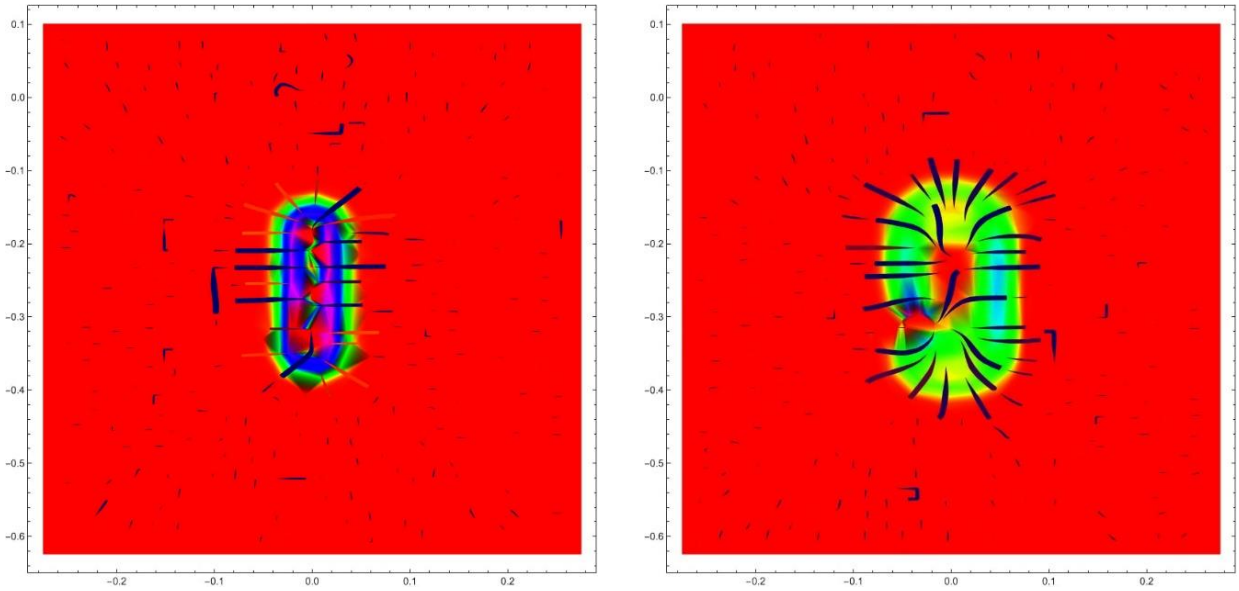


Рисунок 5.3 – Динамика потоковых линий скорости в моменты времени t =1; 4; 7; 12; 16 и 20 с

На рисунке 5.4 представлен потоковый плотностный график, который дополняет рисунок 5.3, отображая плотность линий потока в зависимости от скорости. Рисунок демонстрирует области с более интенсивным движением жидкости, помогая лучше понять динамику.



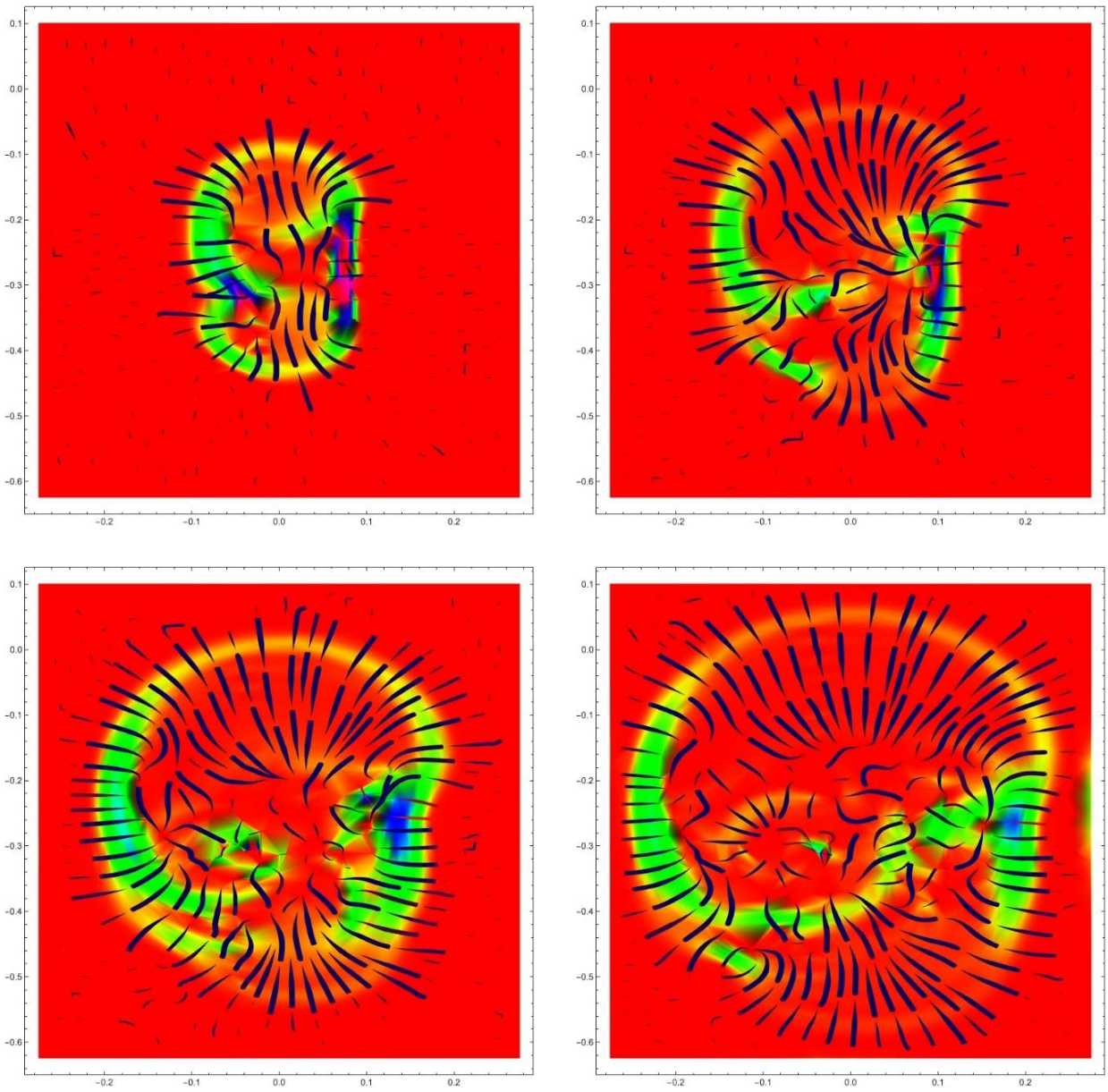


Рисунок 5.4 – Динамика потоковых линий скорости с плотностью потока в моменты времени t =1; 4; 7; 12; 16 и 20 с

Проведенное моделирование цунами в среде Wolfram Mathematica показало, что математические методы позволяют эффективно анализировать динамику распространения волны. Решение дифференциальных уравнений с помощью DSolve и последующая визуализация в виде потоковых графиков StreamPlot и StreamDensityPlot позволили выявить характер изменений поля скоростей в различные моменты времени.

Полученные результаты подтверждают сложность нелинейных процессов, сопровождающих распространение цунами. Данные модели и их визуализация могут быть полезны для прогнозирования последствий цунами и разработки стратегий минимизации ущерба.

Таким образом, использование математического моделирования играет ключевую роль в понимании природных катастроф и их возможных последствий.

ЗАКЛЮЧЕНИЕ

В ходе работы исследованы и протестированы основные функции для формирования и визуализации векторных полей в системе Wolfram Mathematica. Изучены инструменты для анализа векторных и цифровых полей.

Реализована визуализация векторных полей на языке программирования Java с использованием библиотеки JavaFX. Несмотря на большую трудоёмкость разработки по сравнению с Wolfram Mathematica, данный подход продемонстрировал неплохие результаты.

Рассмотрены основные алгоритмы параллельных вычислений в системе Wolfram Mathematica. Проведён сравнительный анализ этих алгоритмов, который показал, что использование параллельных алгоритмов может значительно ускорить время вычислений. Однако при выполнении задач с малым объёмом вычислений следует учитывать накладные расходы и тщательно выбирать метод распараллеливания задачи, в данном случае выбор последовательного алгоритма может быть наилучшим.

Смоделировано распространения волн цунами. Моделирование цунами является важным инструментом, позволяющим глубже понять волновые процессы и выявить их закономерности, что ценно для разработки мер предупреждения стихийных бедствий.

Результаты работы доложены на II Международной научно-практической конференции Трансформация механико-математического и IT-образования в условиях цифровизации (TMMIE-II), материалы доклада приняты в печать.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Каталог «WOLFRAM Demonstrations Project». [Электронный ресурс]. – Режим доступа: https://demonstrations.wolfram.com. – Дата доступа: 04.10.2023.
2. Параллельный алгоритм – Краткое руководство. [Электронный ресурс]. – Режим доступа: https://coderlessons.com/tutorials/akademicheskii/vyuchit-parallelnyi-algoritm/parallelnyi-algoritm-kratkoe-rukovodstvo. – Дата доступа 10.10.2024.
3. Пелиновский Е. Н. Гидродинамика воли цунами / ИПФ РАН. – Нижний Новгород, 1996. 275 с.
4. Таранчук, В.Б. Компьютерные модели подземной гидродинамики / В.Б. Таранчук. – Минск : БГУ, 2020. – 235 с.
5. Barovik, D. Tools for the analysis and visualisation of distributions and vector fields in surface forest fires modelling / D.V. Barovik, V.B Taranchuk // Journal of the Belarusian State University. Mathematics and Informatics. – 2. – 2022. – P. 82–93. (in Engl.) == /№ 2 (2022): Журнал Белорусского государственного университета. Математика. Информатика/
6. Department of Computer Science and Engineering [Электронный ресурс]. – Режим доступа: http://avida.cs.wright.edu/courses/CS399/CS399_5.pdf // Wright State University. – Дата доступа: 04.10.2023.
7. DSolve Overview [Электронный ресурс]. – Режим доступа: https://reference.wolfram.com/language/tutorial/DSolveOverview.html. – Дата доступа: 04.04.2025.
8. Dynamic Visualization [Электронный ресурс]. – Режим доступа: https://reference.wolfram.com/language/guide/DynamicVisualization.html. – Дата доступа: 04.04.2025.
9. Function Visualization [Электронный ресурс]. – Режим доступа: https://reference.wolfram.com/language/guide/FunctionVisualization.html. – Дата доступа: 04.04.2025.
10. Helgeland, A. Visualization of vector fields using Line Integral Convolution and volume rendering//A. Helgeland//University of Oslo: 2002
11. JavaFX Documentation. [Электронный ресурс]. – Режим доступа: https://openjfx.io. – Дата доступа: 20.04.2024.
12. Parallel Computing [Электронный ресурс]. – Режим доступа: https://reference.wolfram.com/language/guide/ParallelComputing.html. – Дата доступа 10.10.2024.
13. Strang G., Herman E. [Электронный ресурс]. – Режим доступа: https://math.libretexts.org/Bookshelves/Calculus/Calculus_%28OpenStax%29/16%3A_Vector_Calculus/16.01%3A_Vector_Fields. – Дата доступа: 04.10.2023.
14. Symbolic Solutions of PDEs [Электронный ресурс]. – Режим доступа: https://reference.wolfram.com/language/tutorial/SymbolicSolutionsOfPDEs.html. – Дата доступа: 04.04.2025.

15. Vector Visualization [Электронный ресурс]. – Режим доступа: https://reference.wolfram.com/language/guide/VectorVisualization.html. – Дата доступа: 04.04.2025.

16. Vojtěch, T. Visualization of a vector field. [Электронный ресурс]. – Режим доступа: https://medium.com/researchsummer/visualization-of-a-vector-field-9402615c780a. – Дата доступа: 04.10.2023.

ПРИЛОЖЕНИЕ А

```
public class visualisation3D extends Application {
    public static void main(String[] args) {
        Application.launch(args);
    }

    private Pane pane;
    private ImageView imageView;
    private AWTChart chart;
    private JavaFXChartFactory factory;
    private Scene scene;

    @Override
    public void start(Stage stage) {
        stage.setTitle(Visualisation3D.class.getSimpleName());

        factory = new JavaFXChartFactory();
        chart  = getDemoChart(factory, "offscreen");
        imageView = factory.bindImageView(chart);

        pane = new Pane();
        scene = new Scene(pane);
        stage.setScene(scene);
        stage.show();
        pane.getChildren().add(imageView);

        factory.addSceneSizeChangedListener(chart, scene);

        stage.setWidth(700);
        stage.setHeight(700);
    }

    private AWTChart getDemoChart(JavaFXChartFactory factory, String toolkit) {

        Mapper mapper = new Mapper() {
            @Override
            public double f(double x, double y) {
                return Math.exp((-(x - 9) * (x - 9) - (y - 2) * (y - 2))) -
                        2 / 3. * Math.exp((-(x - 7) * (x - 7) - (3 * y - 3) *
(3 * y - 3))) + Math.exp((-(x - 6.5) * (x - 6.5) - (y - 3) * (y - 3)));
            }
        };

        Range range = new Range(0, 11);
        int steps = 10;

        final Shape surface = Builder.buildOrthonormal(mapper, range, steps);
        surface.setColorMapper(new ColorMapper(new ColorMapRainbow(),
surface.getBounds().getZmin(), surface.getBounds().getZmax(), new Color(1, 1,
1, .5f)));
        surface.setFaceDisplayed(true);
        surface.setWireframeDisplayed(false);

        Quality quality = Quality.Advanced;

        AWTChart chart = (AWTChart) factory.newChart(quality, toolkit);
        chart.getScene().getGraph().add(surface);
        return chart;
    }
}
```

ПРИЛОЖЕНИЕ Б

```
public class vectorFieldPlot2D extends Application {

    private static final int WIDTH = 800;
    private static final int HEIGHT = 600;
    private static final int MARGIN = 50;

    private static final double X_MIN = 5;
    private static final double X_MAX = 11;
    private static final double Y_MIN = 0;
    private static final double Y_MAX = 4;

    private static final int NUM_TICKS = 10;

    @Override
    public void start(Stage primaryStage) {

        Canvas canvas = new Canvas(WIDTH, HEIGHT);
        GraphicsContext gc = canvas.getGraphicsContext2D();

        drawFrame(gc);

        drawVectorField(gc);

        BorderPane root = new BorderPane();
        root.setCenter(canvas);

        Scene scene = new Scene(new Group(root), WIDTH, HEIGHT);

        primaryStage.setTitle("2D Vector Field");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    private void drawFrame(GraphicsContext gc) {

        gc.setStroke(Color.BLACK);
        gc.strokeRect(MARGIN, MARGIN, WIDTH - 2 * MARGIN, HEIGHT - 2 * MARGIN);

        double zeroX = map(0, X_MIN, X_MAX, MARGIN, WIDTH - MARGIN);
        gc.strokeLine(zeroX, HEIGHT - MARGIN, WIDTH - MARGIN, HEIGHT - MARGIN);

        double zeroY = map(0, Y_MIN, Y_MAX, HEIGHT - MARGIN, MARGIN);
        gc.strokeLine(MARGIN, zeroY, MARGIN, HEIGHT - MARGIN);

        double xTickStep = (X_MAX - X_MIN) / NUM_TICKS;
        for (int i = 1; i <= NUM_TICKS; i++) {
            double xTick = X_MIN + i * xTickStep;
            double xPos = map(xTick, X_MIN, X_MAX, MARGIN, WIDTH - MARGIN);
            gc.strokeLine(xPos, HEIGHT - MARGIN - 5, xPos, HEIGHT - MARGIN +
5);
            gc.fillText(String.format("%.1f", xTick), xPos - 10, HEIGHT -
MARGIN + 20);
        }

        double yTickStep = (Y_MAX - Y_MIN) / NUM_TICKS;
        for (int i = 1; i <= NUM_TICKS; i++) {
            double yTick = Y_MIN + i * yTickStep;
            double yPos = map(yTick, Y_MIN, Y_MAX, HEIGHT - MARGIN, MARGIN);
            gc.strokeLine(MARGIN - 5, yPos, MARGIN + 5, yPos);
            gc.fillText(String.format("%.1f", yTick), MARGIN - 30, yPos + 5);
        }
```

```
        gc.fillText("Y", MARGIN - 20, MARGIN - 20);
        gc.fillText("X", WIDTH - MARGIN + 20, HEIGHT - MARGIN + 20);
    }

    private double gradX(double x, double y) {
        return -2 * Math.exp(-(-9 + x) * (-9 + x) - (-2 + y) * (-2 + y)) * (-9
+ x) +
                4 * Math.exp(-(-7 + x) * (-7 + x) - (-3 + 3 * y) * (-3 + 3 *
y)) * (-7 + x) / 3 -
                2 * Math.exp(-(-6.5 + x) * (-6.5 + x) - (-3 + y) * (-3 + y)) *
(-6.5 + x);
    }

    private double gradY(double x, double y) {
        return -2 * Math.exp(-(-6.5 + x) * (-6.5 + x) - (-3 + y) * (-3 + y)) *
(-3 + y) -
                2 * Math.exp(-(-9 + x) * (-9 + x) - (-2 + y) * (-2 + y)) * (-2
+ y) +
                4 * Math.exp(-(-7 + x) * (-7 + x) - (-3 + 3 * y) * (-3 + 3 *
y)) * (-3 + 3 * y);
    }

    private void drawVectorField(GraphicsContext gc) {

        double step = 0.2 ;

        for (double x = X_MIN + 0.15; x < X_MAX - step; x += step) {
            for (double y = Y_MIN + 0.2; y < Y_MAX; y += step) {

                double gradX = gradX(x, y);
                double gradY = gradY(x, y);

                double length = Math.sqrt(gradX * gradX + gradY * gradY);
                gradX /= length / 15;
                gradY /= length / 15;

                double xPos = map(x, X_MIN, X_MAX, MARGIN, WIDTH - MARGIN);
                double yPos = map(y, Y_MIN, Y_MAX, HEIGHT - MARGIN, MARGIN);

                double endX = xPos + gradX;
                double endY = yPos + gradY;

                gc.setStroke(Color.BLUE);
                gc.strokeLine(xPos, yPos, xPos + gradX, yPos + gradY);

                drawArrowhead(gc, xPos, yPos, endX, endY);
            }
        }
    }

    private void drawArrowhead(GraphicsContext gc, double x1, double y1, double
x2, double y2) {

        double arrowheadSize = 6;

        double angle = Math.atan2(y2 - y1, x2 - x1);

        double x3 = x1 + arrowheadSize * Math.cos(angle - Math.PI / 6);
        double y3 = y1 + arrowheadSize * Math.sin(angle - Math.PI / 6);
        double x4 = x1 + arrowheadSize * Math.cos(angle + Math.PI / 6);
        double y4 = y1 + arrowheadSize * Math.sin(angle + Math.PI / 6);

        gc.strokeLine(x1, y1, x3, y3);
```

```
        gc.strokeLine(x1, y1, x4, y4);
    }

    private double map(double value, double fromMin, double fromMax, double
toMin, double toMax) {
        return (value - fromMin) * (toMax - toMin) / (fromMax - fromMin) +
toMin;
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

ПРИЛОЖЕНИЕ В

```
LaunchKernels[6];
vx[x_,y_,z_]:=-y e^-(x^2+y^2+z^2);

vy[x_,y_,z_]:=x e^-(x^2+y^2+z^2);

vz[x_,y_,z_]:=z e^-(x^2+y^2+z^2);
v[x_,y_,z_]:={vx[x,y,z],vy[x,y,z],vz[x,y,z]};

VectorPlot3D[v[x,y,z],{x,-2,2},{y,-2,2},{z,-2,2},VectorSizes->Small,VectorPoints-
>5,PlotLegends->Automatic,AxesLabel->{"x","y","z"}]

StreamPlot3D[v[x,y,z],{x,-2,2},{y,-2,2},{z,-2,2},StreamPoints->Automatic,StreamStyle-
>"Arrow3D",Axes->True,PlotLegends->Automatic,AxesLabel->{"x","y","z"}]

grid=Flatten[Table[{x,y,z},{x,-2,2,0.1},{y,-2,2,0.1},{z,-2,2,0.1}],2];

Функция ParallelMap
seqTimeCurl=AbsoluteTiming[curlSeq=Map[Curl[v@@#,{x,y,z}]&,grid];][[1]];
seqTimeDiv=AbsoluteTiming[divSeq=Map[Div[v@@#,{x,y,z}]&,grid];][[1]];

parTimeCurl=AbsoluteTiming[curlPar=ParallelMap[Curl[v@@#,{x,y,z}]&,grid];][[1]];
parTimeDiv=AbsoluteTiming[divPar=ParallelMap[Div[v@@#,{x,y,z}]&,grid];][[1]];

{seqTimeDiv,parTimeDiv,seqTimeDiv/parTimeDiv}
{seqTimeCurl,parTimeCurl,seqTimeCurl/parTimeCurl}

Функция ParallelTable
seqTimeCurl=AbsoluteTiming[curlSeq=Table[Curl[v@@pt,{x,y,z}],{pt,grid}]][[1]];
seqTimeDiv=AbsoluteTiming[divSeq=Table[Div[v@@pt,{x,y,z}]&,{pt,grid}]][[1]];

parTimeCurl=AbsoluteTiming[curlPar=ParallelTable[Curl[v@@pt,{x,y,z}]&,{pt,grid}]][[1]];
parTimeDiv=AbsoluteTiming[divPar=ParallelTable[Div[v@@pt,{x,y,z}]&,{pt,grid}]][[1]];

{seqTimeDiv,parTimeDiv,seqTimeDiv/parTimeDiv}
{seqTimeCurl,parTimeCurl,seqTimeCurl/parTimeCurl}

Функция Parallelize
seqTimeCurl=AbsoluteTiming[curlSeq=Table[Curl[v@@pt,{x,y,z}],{pt,grid}]][[1]];
seqTimeDiv=AbsoluteTiming[divSeq=Table[Div[v@@pt,{x,y,z}]&,{pt,grid}]][[1]];

parTimeCurl=AbsoluteTiming[curlPar=Parallelize[Table[Curl[v@@pt,{x,y,z}],{pt,grid}]]
   ][[1]];
parTimeDiv=AbsoluteTiming[divPar=Parallelize[Table[Div[v@@pt,{x,y,z}],{pt,grid}]]
   ][[1]];

{seqTimeDiv,parTimeDiv,seqTimeDiv/parTimeDiv}
{seqTimeCurl,parTimeCurl,seqTimeCurl/parTimeCurl}
```