|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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 *Робототехники и комплексной автоматизации*

КАФЕДРА *Системы автоматизированного проектирования (РК-6)*

**ОТЧЕТ О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ**

по дисциплине: «Автоматизация технологического проектирования»

|  |  |  |
| --- | --- | --- |
| Студент |  | Косенков Александр Александрович |
| Группа |  | РК6-32М |
| Тип задания |  | Лабораторная работа №1 |
| Тема лабораторной работы |  | Декомпозиция Морса при помощи окружностей |

Студент **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_Косенков А.А.\_**

*подпись, дата фамилия, и.о.*

Преподаватель **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_Божко А.Н.\_**

*подпись, дата фамилия, и.о.*

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Москва, 2022 г.*

**Оглавление**

[Задание на лабораторную работу 3](#_Toc124419687)

[Цель выполнения лабораторной работы 5](#_Toc124419688)

[Выполненные задачи 5](#_Toc124419689)

[Разработка исходных кодов 6](#_Toc124419690)

[Результаты численного решения ОДУ 9](#_Toc124419691)

[Автоматизация тестирования с использованием Gitlab CI 13](#_Toc124419692)

[Ответы на вопросы базовой части 15](#_Toc124419693)

[Заключение 15](#_Toc124419694)

[Список использованных источников 16](#_Toc124419695)

# Задание на лабораторную работу

Разработать программу разбиения конфигурационного пространства методом декомпозиции Морса при помощи окружностей. Препятствия создаются с помощью программы-генератора. Требуется определить зависимость времени выполнения программы от количества препятствий.

# Цель выполнения лабораторной работы

**Цель выполнения лабораторной работы** – ознакомление и изучение методов точной декомпозиции для решения задач планирования перемещений и реализация алгоритма декомпозиции Морса при помощи окружностей.

# Выполненные задачи

1. Разработана программа на языке С++, реализующая заданный метод численного интегрирования.
2. Разработана программа для тестирования программы, реализующей численный метод.
3. Разработан конфигурационный скрипт GitLab CI .gitlab-ci.yml, осуществляющий автоматический запуск сборки и тестирования программного кода при совершении пользователем операции git push.
4. Получены графические изображения численно получаемых решений заданного уравнения при всех указанных начальных условиях.
5. Представлены ответы на вопросы базовой и продвинутой частей задания.

# Разработка исходных кодов

Для варианта 5 () были определены следующие параметры для выбора условий реализаций исходных кодов:

* Номер уравнения: ;
* Номер метода:
* Индексы параметров:

Таким образом, исходные коды должны представлять собой вычислительный метод для решения следующего уравнения:

методом Адамса-Башфорта-Моултона 4-го порядка со следующими параметрами: , , , , .

Метод Адамса-Башфорта-Моултона 4-го порядка представляет собой многошаговый метод интегрирования ОДУ, построенный по схеме предиктор-корректор. Данный метод строится за счет комбинации многошаговых методов Адамса-Башфорта и Адамса-Моултона одного порядка точности, представляющих собой предиктор и корректор соответственно. Данная комбинация позволяет повысить устойчивость численной схемы. Метод описывается выражениями (5.1 – 5.3) [2].

(5.2) описывает предиктор, представленный явным методом Адамса-Башфорта, который находит приближенное значение . (3.3) описывает корректор, представленный неявным методом Адамса-Моултона, использующий в правой части значение , вычисленное предиктором.

При этом метод требует задания начальных условий для первых 4-х точек, что возможно реализовать через предварительное вычисление данных значений другим численным методом. В рамках разработанной реализации вычисление первых 4-х значений осуществляется с помощью метода Рунге-Кутты 4-го порядка.

Метод Рунге-Кутты 4-го порядка основан на аппроксимации ряда Тэйлора и для рассматриваемой задачи имеет следующую формулировку:

где:

Программная реализация метода Адамса-Башфорта-Моултона 4-го порядка представлена в Листинге 1.

Листинг 1. Программная реализация метода Адамса-Башфорта-Моултона 4-го порядка.

#include <cmath>

#include "adamsBashfortMoultonSolver.hpp"

int solve\_abm(Solution &solution, std::map<double, double> &resultABM) {

resultABM.clear();

Params params = solution.getParams();

double timestamp = params.timeStart;

double vCurrent = params.y\_0;

double vPrev = params.y\_0;

double vPrevs[4] = { vCurrent };

resultABM.insert(std::pair<double, double>(timestamp, vCurrent));

// Computing first 4 values via Runge-Kutta method

for (int i = 1; i < 4; i++) {

double k1 = params.step \* solution.getDiffSolutionPerTime(timestamp, vPrev);

double k2 = params.step \* solution.getDiffSolutionPerTime(timestamp, vPrev + 1 / 2 \* k1);

double k3 = params.step \* solution.getDiffSolutionPerTime(timestamp, vPrev + 1 / 2 \* k2);

double k4 = params.step \* solution.getDiffSolutionPerTime(timestamp, vPrev + k3);

vCurrent = vPrev + (k1 + 2. \* k2 + 2. \* k3 + k4) / 6;

vPrev = vCurrent;

timestamp += params.step;

resultABM.insert(std::pair<double, double>(timestamp, vCurrent));

vPrevs[i] = vCurrent;

}

double vCurrentPredict = 0.0;

double vCurrentCorrect = 0.0;

for (int i = 4; i <= (int) (params.timeEnd / params.step); i++) {

vCurrentPredict = vPrevs[3] + params.step / 24. \* (

55 \* solution.getDiffSolutionPerTime(timestamp, vPrevs[3]) -

59 \* solution.getDiffSolutionPerTime(timestamp, vPrevs[2]) +

37 \* solution.getDiffSolutionPerTime(timestamp, vPrevs[1]) -

9 \* solution.getDiffSolutionPerTime(timestamp, vPrevs[0])

);

vCurrentCorrect = vPrevs[3] + params.step / 24. \* (

9 \* solution.getDiffSolutionPerTime(timestamp, vCurrentPredict) +

19 \* solution.getDiffSolutionPerTime(timestamp, vPrevs[3]) -

5 \* solution.getDiffSolutionPerTime(timestamp, vPrevs[2]) +

1 \* solution.getDiffSolutionPerTime(timestamp, vPrevs[1])

);

vPrevs[0] = vPrevs[1];

vPrevs[1] = vPrevs[2];

vPrevs[2] = vPrevs[3];

vPrevs[3] = vCurrentCorrect;

timestamp += params.step;

resultABM.insert(std::pair<double, double>(timestamp, vCurrentCorrect));

}

isinf

return 0;

}

# Результаты численного решения ОДУ

Результаты решения ОДУ (1) разработанным методом для значений представлены на рис. 1-11. В случае отрицательных значений метод работает некорректно, откуда .

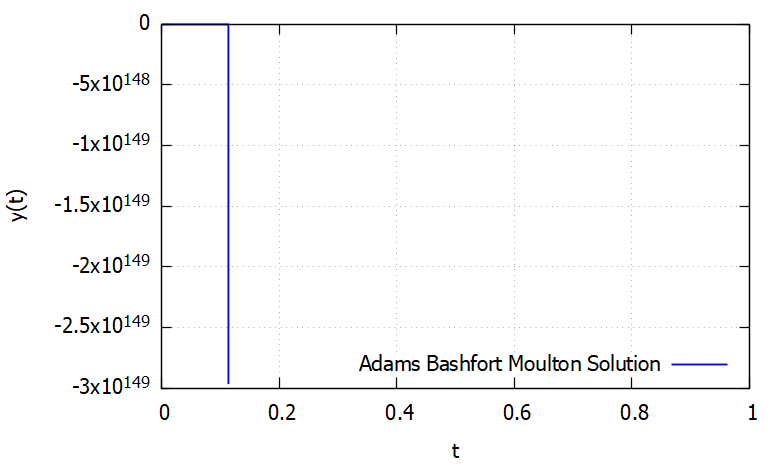


Рис. . Численное решение ОДУ (1) при .

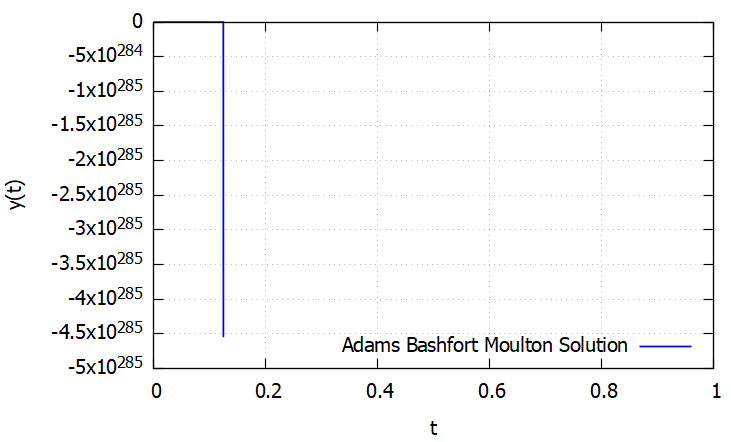


Рис. . Численное решение ОДУ (1) при .

Изображение выглядит как стол

Автоматически созданное описание

*Рис. 3.* Численное решение ОДУ (1) при .

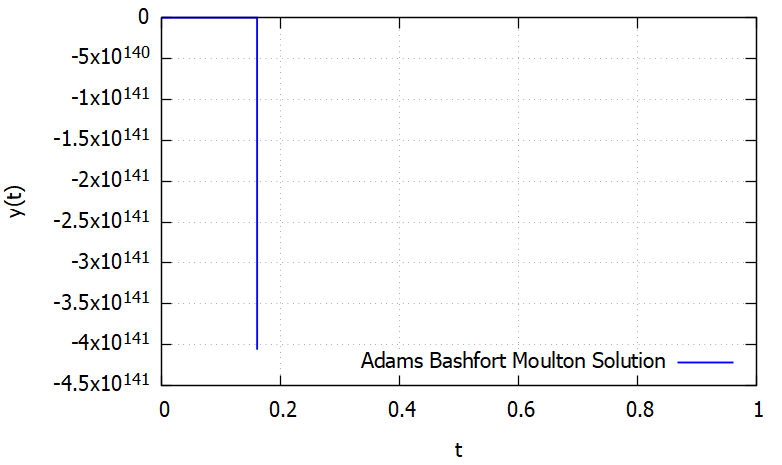


Рис. 4. Численное решение ОДУ (1) при .

Изображение выглядит как стол

Автоматически созданное описание

Рис.5 . Численное решение ОДУ (1) при .

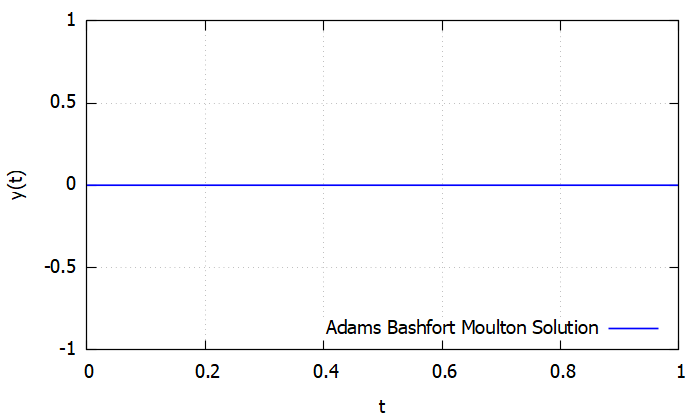


Рис. 6. Численное решение ОДУ (1) при .

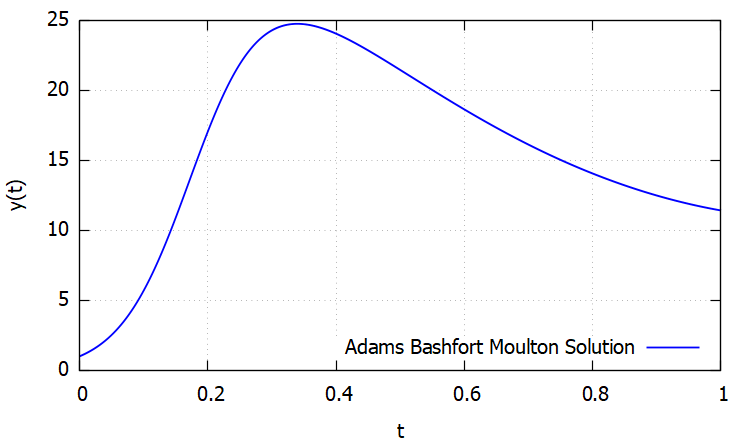


Рис. 7. Численное решение ОДУ (1) при .

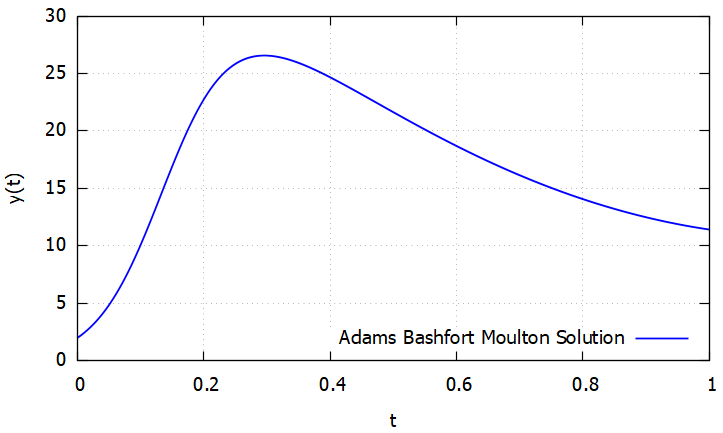


Рис. 8. Численное решение ОДУ (1) при .

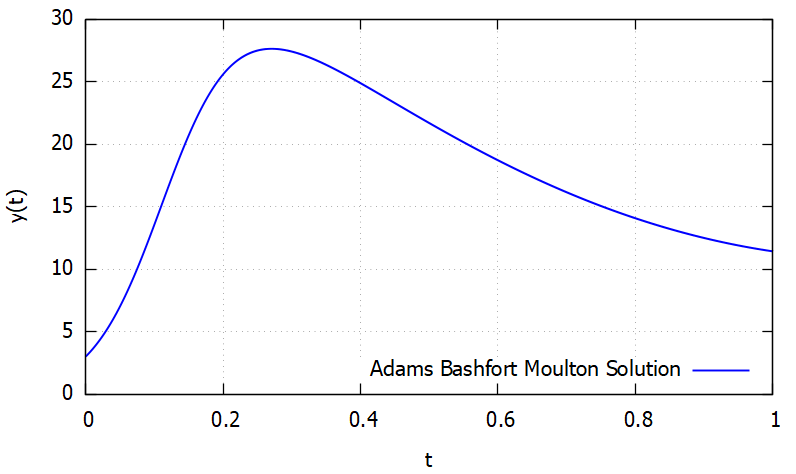


Рис. 9. Численное решение ОДУ (1) при .

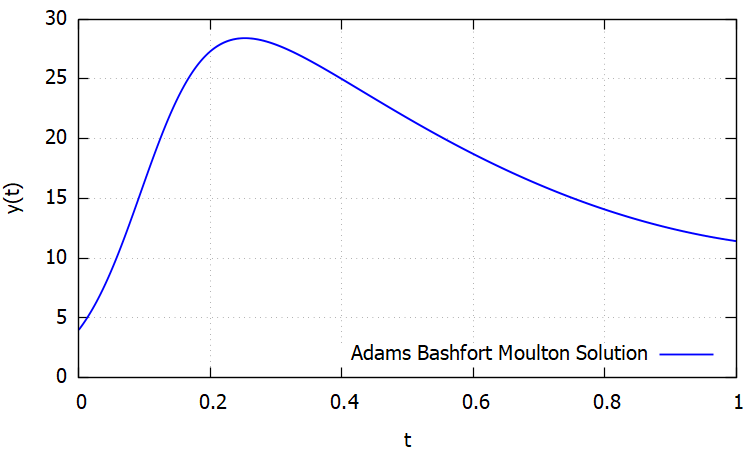


Рис. 10. Численное решение ОДУ (1) при .

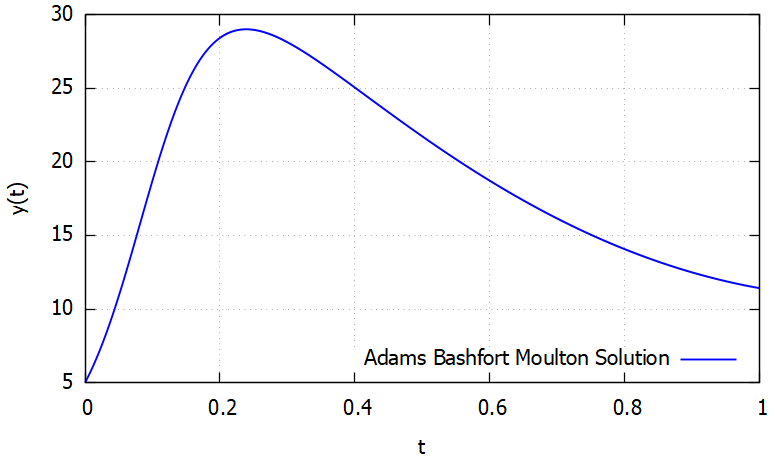


Рис. . Численное решение ОДУ (1) при .

# Автоматизация тестирования с использованием Gitlab CI

Тестирование программной реализации метода Адамса-Башфорта-Моултона было осуществлено на предмет того, не «сваливается» ли метод в значения и , для чего метод *solve\_abm*, представленный в Листинге 1 возвращает целочисленное значение, обозначающее конечный статус решения: 0 в случае успешного; -1, если в процессе вычислений появились значения *-inf*, *inf* и *nan*, проверка которых осуществляется с помощью методов стандартной библиотеки cmath *isnan* и *isinf*.

Сама реализация тестирования (Листинг 2) осуществлена в директории tests как отдельный cmake проект, использующий библиотеку google tests для генерации тестов для значений .

Листинг 2. Реализация тестирования с помощью google tests.

std::vector<double> create\_test\_cases() {

std::vector<double> y\_starts(11);

std::iota(y\_starts.begin(), y\_starts.end(), -5.);

return y\_starts;

}

class TestFixture: public testing::TestWithParam<double> {};

TEST\_P(TestFixture, TestErrorIsSmallEnough) {

double test\_case = GetParam();

std::cout << "Test for y\_0 = " << test\_case << std::endl;

Solution solution((Params(test\_case)));

std::map<double, double> abmResult;

int solveStatus = solve\_abm(solution, abmResult);

ASSERT\_TRUE(solveStatus == 0);

}

INSTANTIATE\_TEST\_CASE\_P(TestCorrectSolving, TestFixture, testing::ValuesIn(create\_test\_cases()));

int main(int argc, char \*argv[]) {

::testing::InitGoogleTest(&argc, argv);

return RUN\_ALL\_TESTS();

Для интеграции тестирования в систему Gitlab CI необходима предварительная разработка скриптовых файлов для сборки и запуска необходимых исполняемых файлов. Так, скрипт install.sh (Листинг 3) осуществляет предварительную установку библиотек в среду выполнения Gitlab CI, скрипт build.sh (Листинг 4) осуществляет сборку тестов, а test.sh (Листинг 5) непосредственно их запускает.

Листинг 3. Скрипт install.sh, осуществляющий установку в ОС необходимых библиотек для проведения тестирования.

#!/bin/bash

apt update **&&** apt **install** -y libgtest-dev cmake

cmake -S **/**usr**/**src**/**gtest -B install\_build **&&** cmake --build install\_build

**mv** install\_build**/**lib**/**lib**\***.a **/**usr**/**lib**/**

Листинг 4. Скрипт build.sh, осуществляющий компиляцию тестов.

#!/bin/bash

cmake -S lab3 -B lab3**/**build **&&** cmake --build lab3**/**build --target lab3\_tests

Листинг 5. Скрипт test.sh, осуществляющий запуск исполняемого файла для тестирования.

#!/bin/bash

.**/**lab3**/**build**/**tests**/**lab3\_tests

Автоматический запуск данных скриптов осуществляется системой Gitlab CI на основе конфигурационного файла .gitlab-ci.yml (Листинг 6), размещенного в корне репозитория. В данном скрипте описывается последовательность действий, которая будет выполняться при осуществлении публикации изменений в репозиторий (команда git push). Упомянутый кофигурационный файл позволяет указать этапы обработки и инструкции для данных этапов. Кроме того, возможно выполнение предварительных действий перед запуском этапов, которые описываются в секции before\_script, содержащей в рассматриваемом случае запуск скрипта install.sh.

Листинг 6. Конфигурационный файл .gitlab-ci.yml с описанием действий в рамках непрерывной интеграции и развертывания CI/CD.

image**:** gcc

before\_script**:**

- bash lab3/scripts/install.sh

stages**:**

- build

- test

build-job:

stage**:** build

script**:** bash lab3/scripts/build.sh

artifacts:

paths**:**

- lab3/build/tests/lab3\_tests

expire\_in**:** 5 mins

test-job:

stage**:** test

script**:** bash lab3/scripts/test.sh

Выполнение указанных в конфигурационном файле инструкций система GitLab осуществляет самостоятельно, предоставляя графический интерфейс для мониторинга результатов.

# Ответы на вопросы базовой части

1. Какие права должны быть предоставлены разработчику для выполнения созданных скриптов на уровне GitLab и операционной системы сервера? Учитывать, что при их предоставлении не должны быть сформированы уязвимости в ОС.

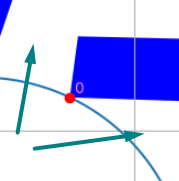
Для осуществления взаимодействия с Gitlab CI у разработчика должны быть роль Developer. Однако, в случае если программа gitlab-runner, осуществляющая анализ конфигурационного файла .gitlab-ci.yml установлена не в docker-контейнере, а напрямую на ОС сервера — может понадобится дополнительная настройка прав в самой ОС для возможности осуществления, например, выполнения команд apt-get (установка ПО в систему).

2. Какое ПО должно быть установлено в ОС сервера для работоспособности созданных скриптов?

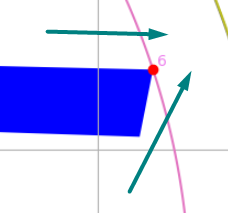
Основным требованием является наличие ПО gitlab-runner. Желательными являются наличие docker, cmake, libgtest-dev, поскольку первый позволяет выделять ресурсы для CI/CD безопасно и не влияя на ОС, а остальные требуются в большинстве случаев.

Крит точки:

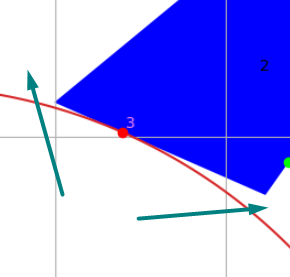
1. Вершина препятствия с наименьшим радиусом:



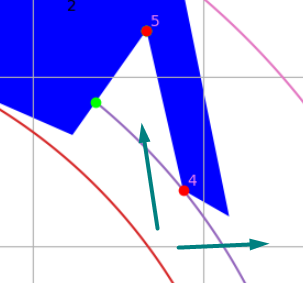
1. Вершина препятствия с наибольшим радиусом:



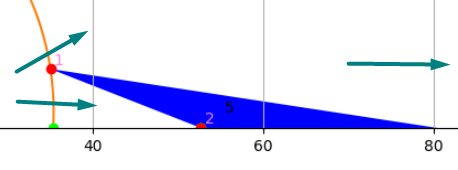
1. Наименьший радиус препятствия не в вершине, а на грани:



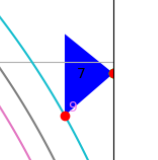
1. Внутренние точки выпуклого внутрь препятствия:



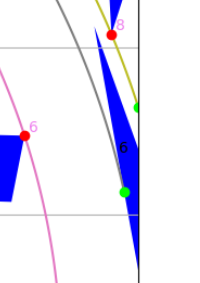
1. Критической точкой не является точка на границе поля, если они минимальная или максимальная:



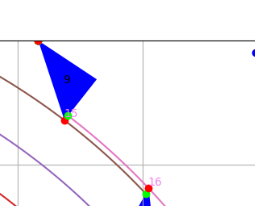
Исключение: препятствие касается границы поля только в одной точке:

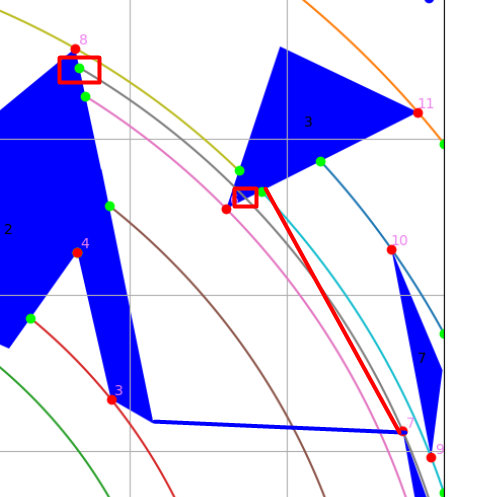


1. Критической точкой не является та, границы дуги которой совпадают с вершиной полигона



Точка позже финальной дуги





# Заключение

В рамках лабораторной работы 3 на примере численного решения заданного ОДУ (1) методом Адамса-Башфорта-Моултона был изучен и реализован процесс автоматизации тестирования программного кода с использованием системы непрерывной интеграции GitLab CI.

# Список использованных источников

1. **Косенков, А.А.** Комплект документации к лабораторной работе 3. URL: [https://sa2systems.ru:88/rk6\_student/edupmi/tree/2022\_rk6\_32m\_kosenkovaa/lab3](https://sa2systems.ru:88/rk6_student/edupmi/tree/2022_rk6_32m_kosenkovaa/lab31)
2. **Першин А.Ю.** *Лекции по вычислительной математике (черновик).* [Электронный ресурс] // Кафедра РК6 (Системы автоматизированного проектирования). МГТУ им. Н.Э. Баумана, Москва, 2020 г., 142 с.