Министерство образования и науки РФ

Санкт-Петербургский Политехнический университет Петра Великого

Институт компьютерных наук и кибербезопасности

Высшая школа программной инженерии

Работа №1

по дисциплине

«**Разработка программного обеспечения для моделирования физических процессов**»

Вариант CP3

**Выполнил**

студент гр. 5130904/10101

Абраамян А. М.

**Преподаватель**

Воскобойников С.П.

Санкт-Петербург

2024

Оглавление

[Постановка задачи 3](#__RefHeading___Toc5397_3458772843)

[Аппроксимация 3](#__RefHeading___Toc5399_3458772843)

[Разностная схема 5](#__RefHeading___Toc5401_3458772843)

[Метод прогонки 5](#__RefHeading___Toc5403_3458772843)

[Вычисление погрешности 7](#__RefHeading___Toc5405_3458772843)

[Оценка работы алгоритма на различных примерах 9](#__RefHeading___Toc5407_3458772843)

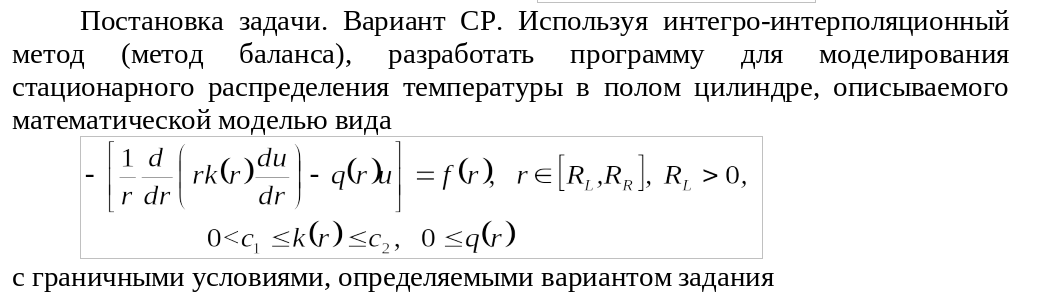
[Пример 1 9](#__RefHeading___Toc5409_3458772843)

[Пример 2 10](#__RefHeading___Toc5411_3458772843)

[Пример 3 10](#__RefHeading___Toc5413_3458772843)

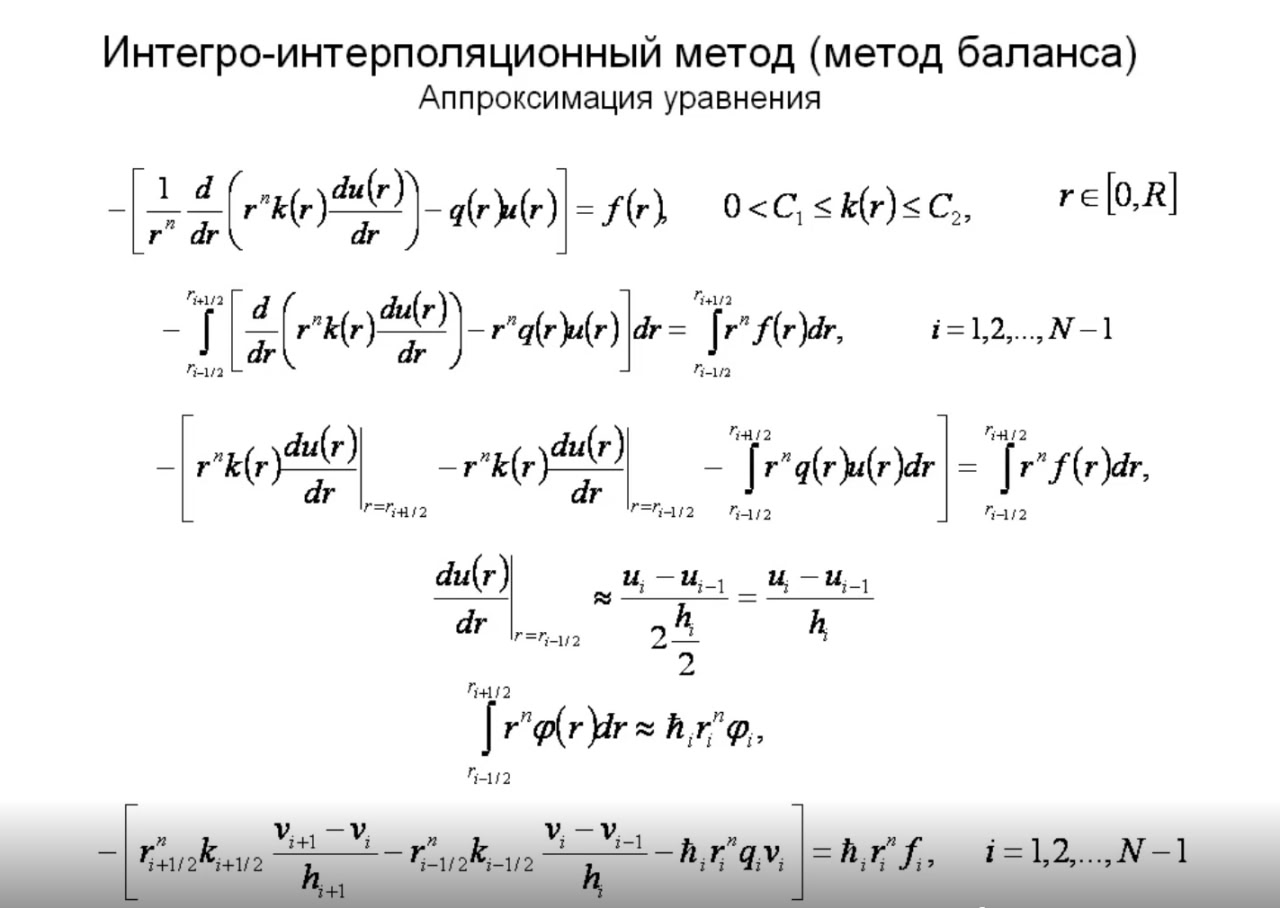
[Код 11](#__RefHeading___Toc5415_3458772843)

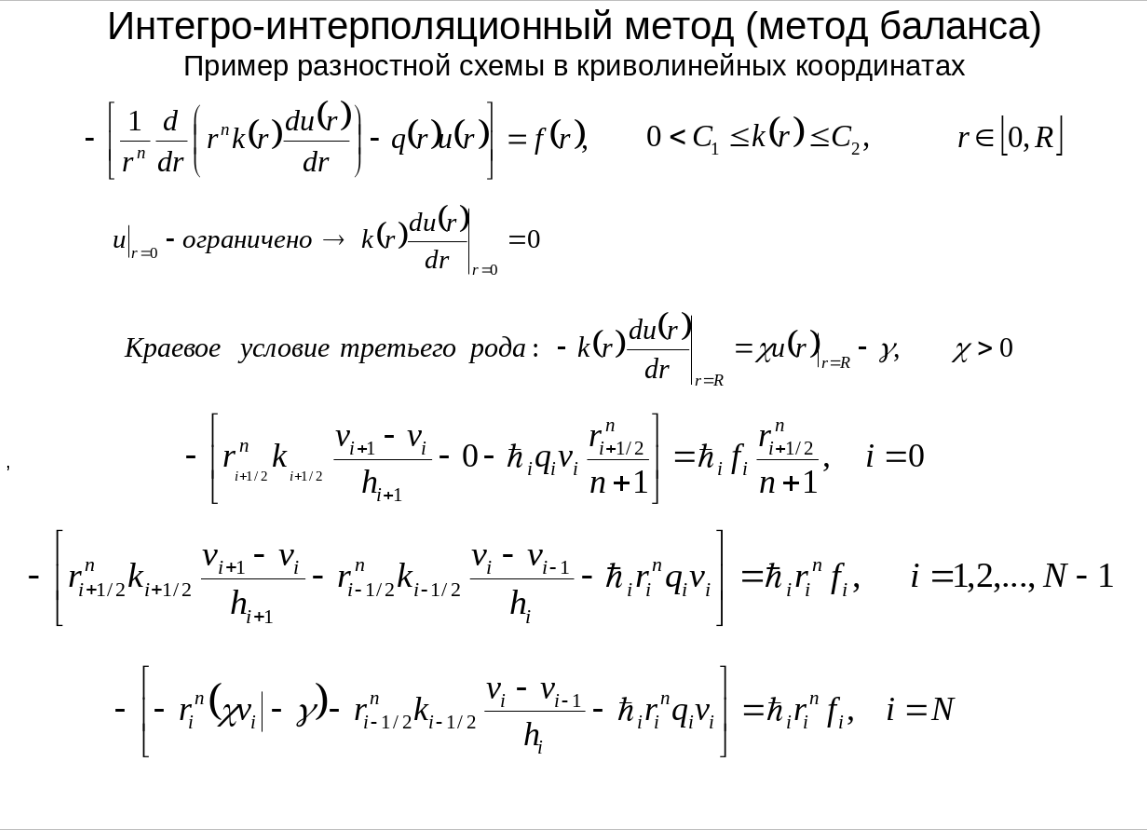
# Постановка задачи



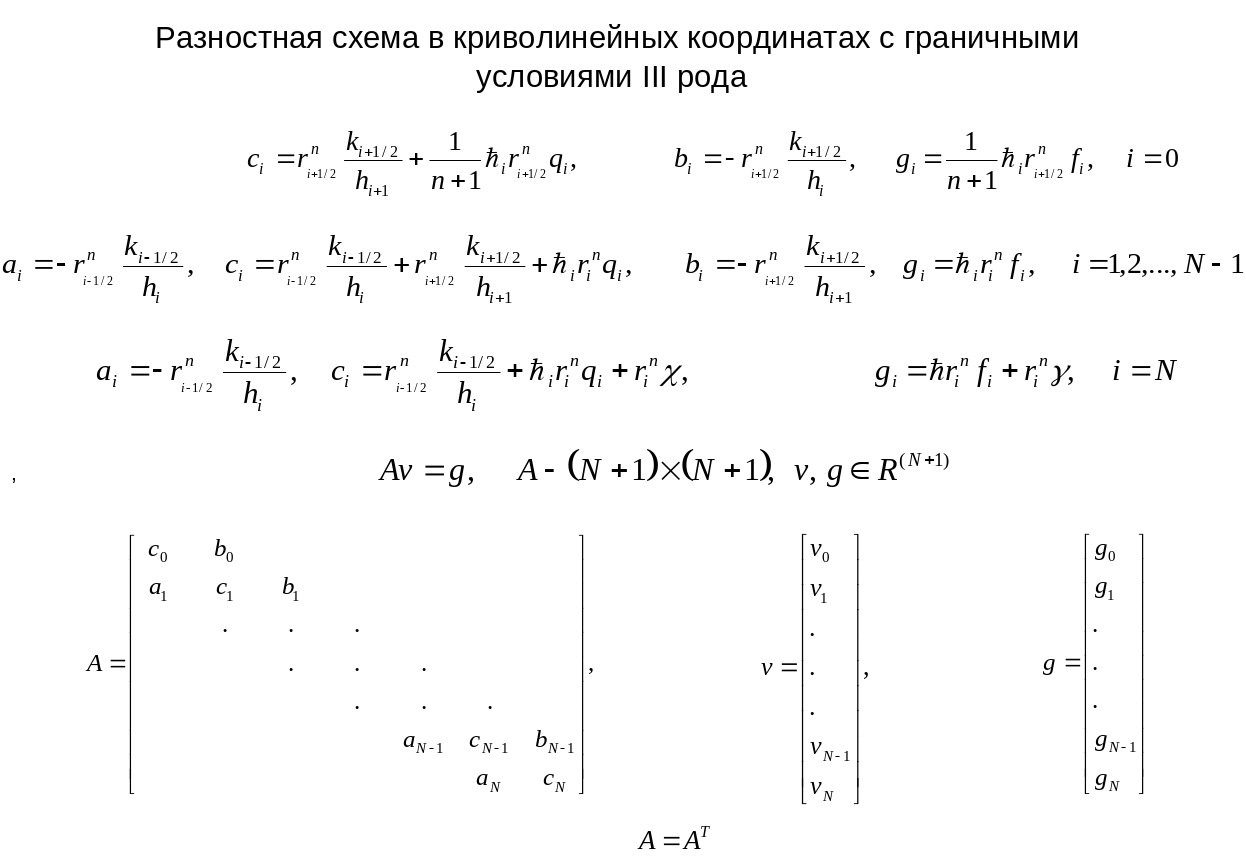
# Аппроксимация

Для третьего рода подойдут следующие формулы:



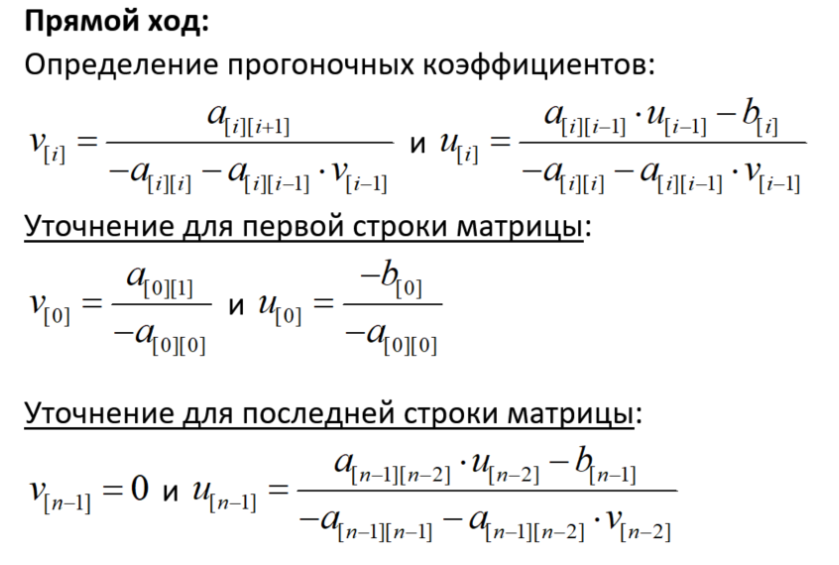


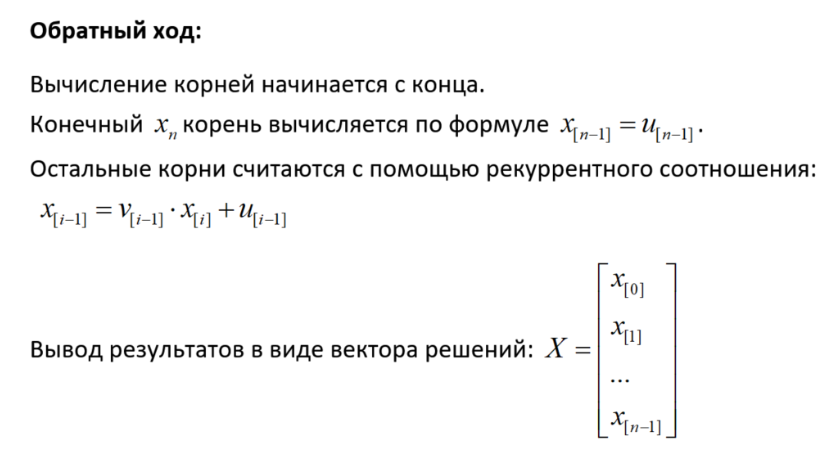
# Разностная схема



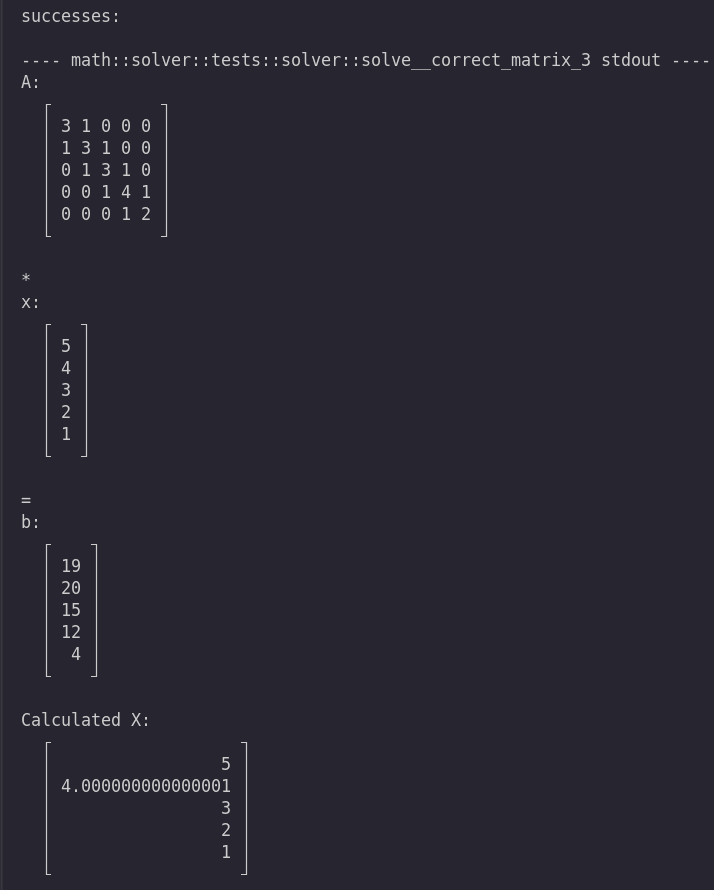
Из приведенной выше схемы для моего варианта задания подойдёт всё кроме **i=0.** Для i=0 подойдут следующие коэффициенты – b = 0, c = 1.

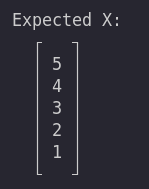
# Метод прогонки





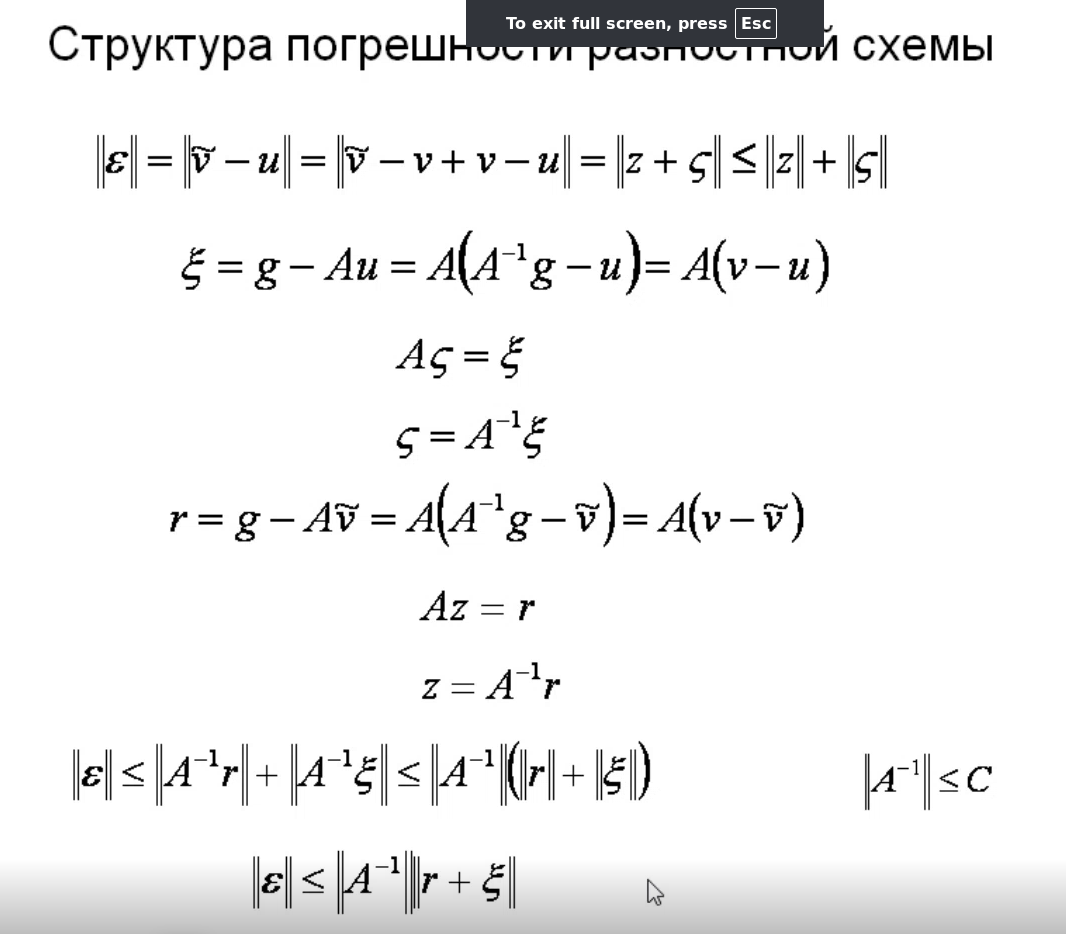
Проверить работоспособность метода помогут тесты. Ниже приведен один из многочисленных тестов доказывающих правильность работы программы и алгоритма в целом

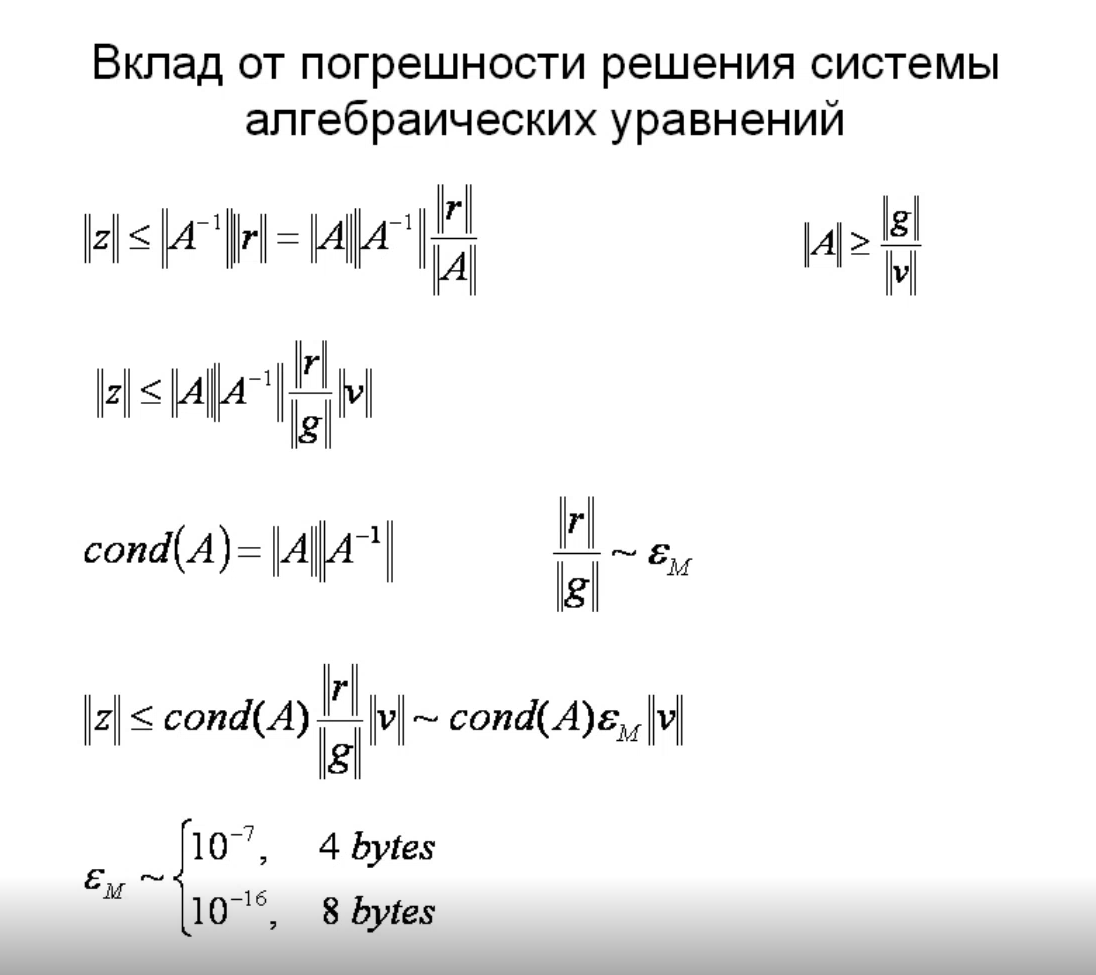


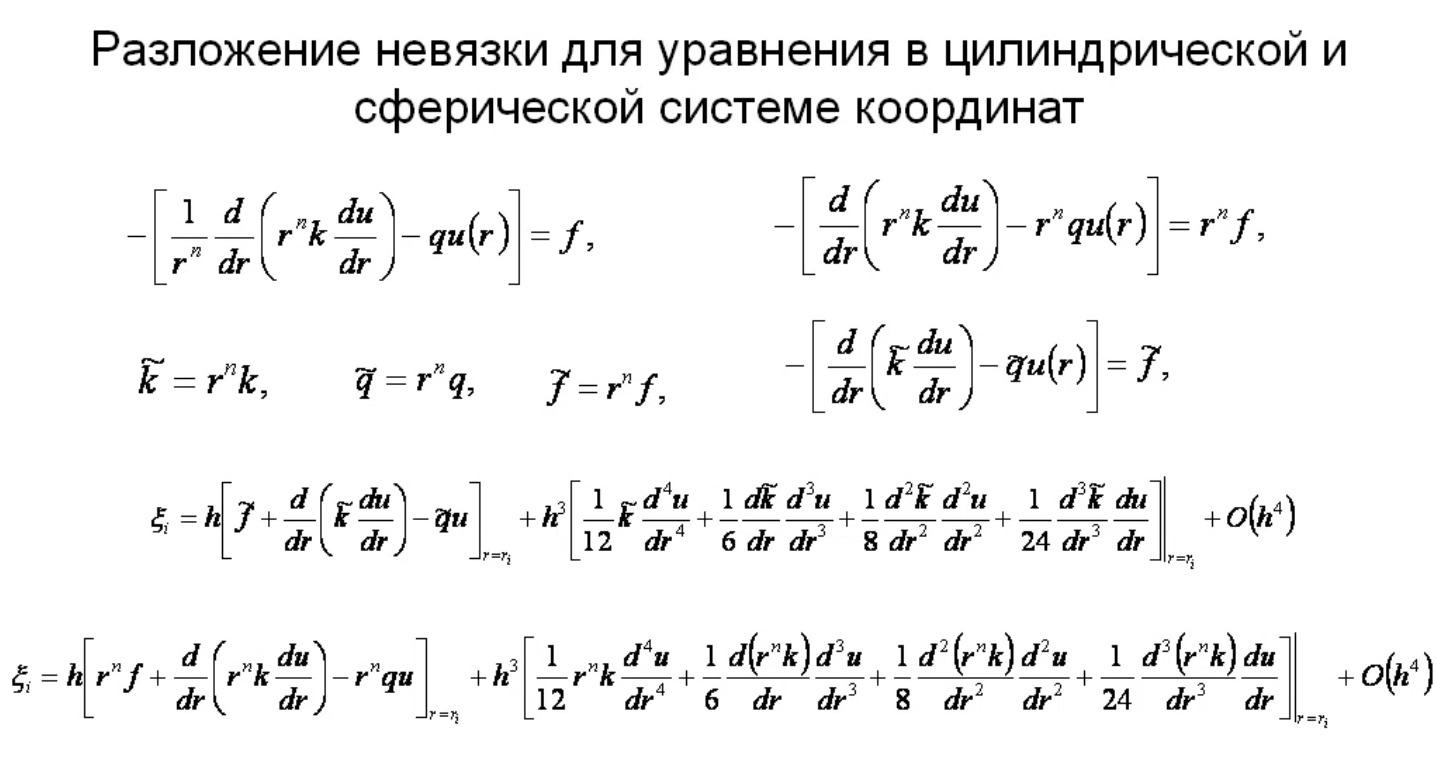


# Вычисление погрешности

Погрешность можно оценить с помощью следующих формул



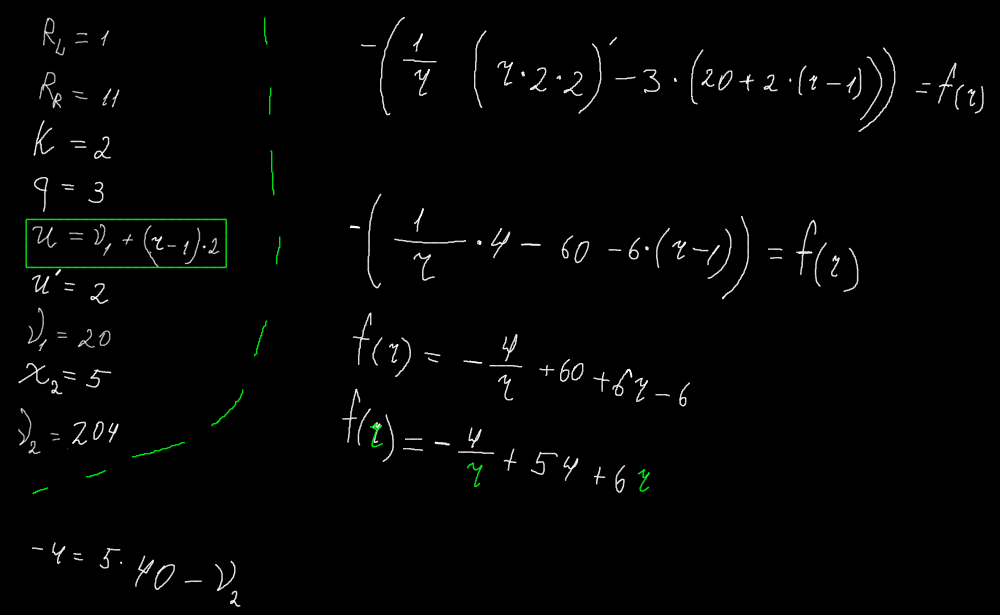
 Взглянув на разложение невязки, можно понять откуда берется погршеность и почему ее нету(она близка к нулевой) для линейных уравнений.



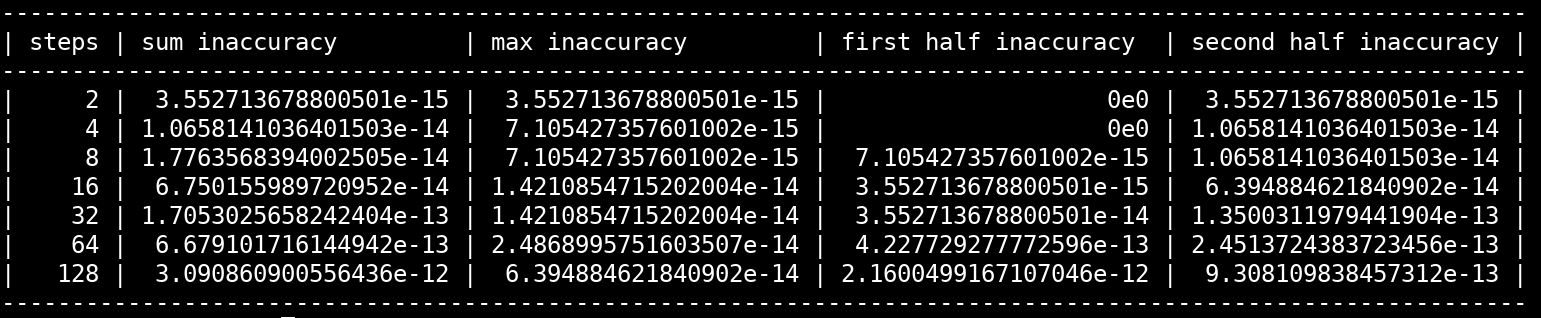
# Оценка работы алгоритма на различных примерах

Для оценки работы составим различные функции и сравним теоретические и экспериментальные данные.

## Пример 1



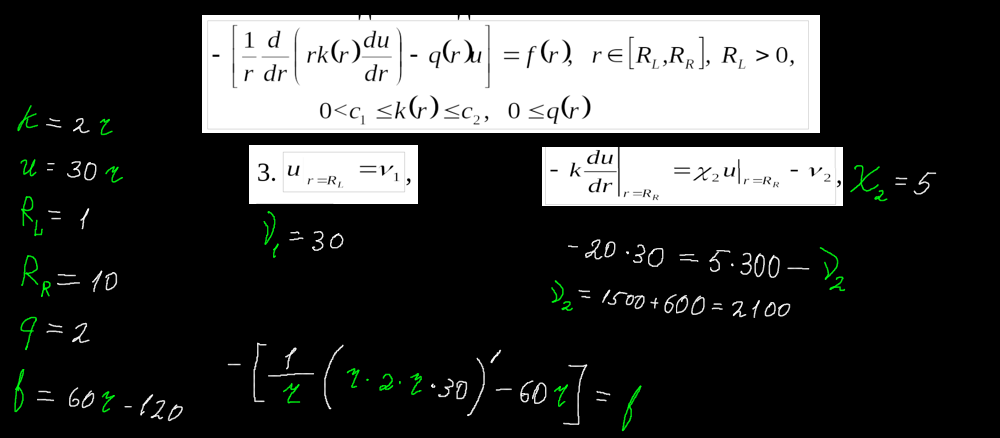
Данный пример предполагает k и q константные для простоты. Введем данные в программу и посмотрим на результат.

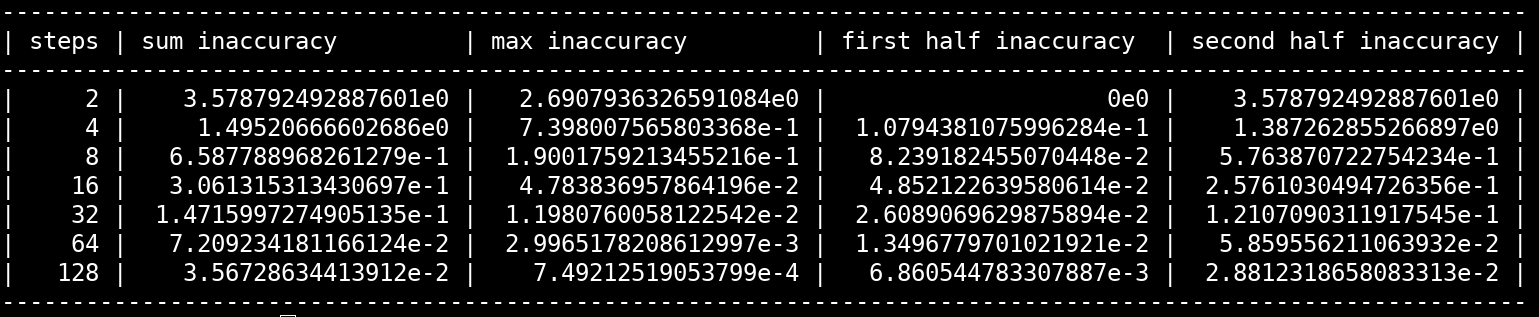


Получаем таблицу показывающую нулевые погрешности. Первый столбец это сумма погрешностей каждого x\_i. Второй столбец – максимальная погрешность между i элементами. Третий столбец – сумма погрешностей первой половины найденного вектора. Четвертый(последний) столбец – сумма погрешностей второй половины найденного вектора.

Нулевая погрешность была ожидаемой исходя из формул приведенных в разделе погрешностей.

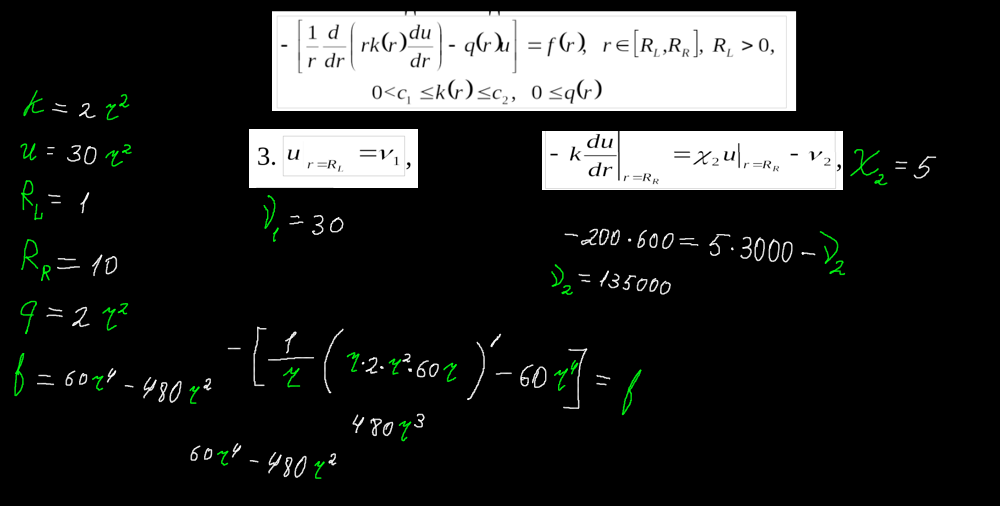
## Пример 2

 Второй пример уже немного интереснее – в качестве входных данных были взяты все константы, кроме k. k стала линейной функцией. Результаты работы программы следующие:

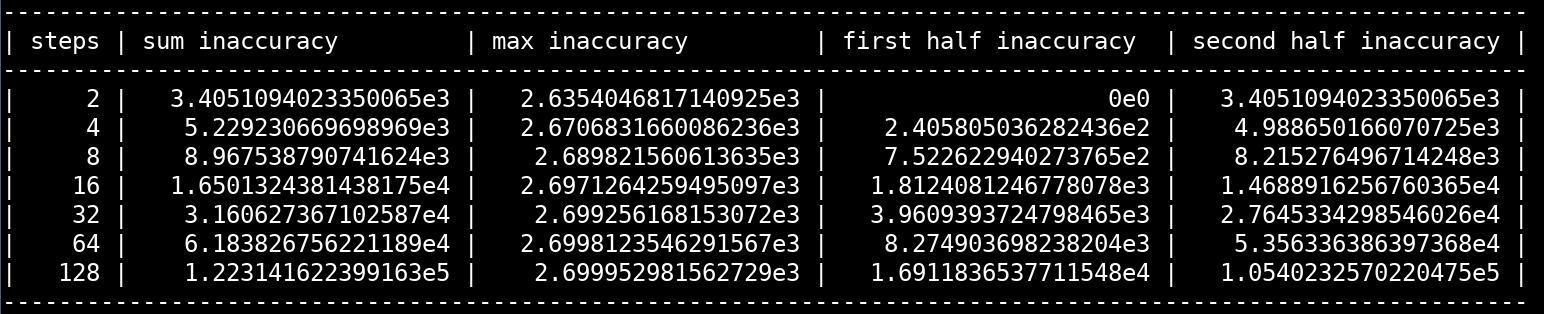


В целом погрешность тоже стремится к нулю, но уже заметно больше чем в предыдущем примере. Опять же взглянув на рассчет погрешности выше можно понять почему так происходит.

## Пример 3



Теперь приведём уже взрослый пример и посмотрим насколько успешно наш алгоритм справится с этой задачей. Теперь все функции взяты хотя бы второй степени, что возможно окажется непосильной задачей для нашей программы.



Как ожидалось – погрешность взлетела до необъятных масштабов. Дело в том что в самом большом коэффициенте невязки у нас стоят производные второй степени. Поскольку этот член не обнулился – погрешность стала возрастать непозволительно быстро.

# Код

Код программы находится на гитхабе по ссылке <https://github.com/Hryapusek/rust-tridiagonal-matrix-vector.git> в ветке СР3.