Министерство образования и науки Российской Федерации
Федеральное государственное
бюджетное образовательное учреждение
высшего профессионального образования
«Петрозаводский государственный университет»
математический факультет
кафедра прикладной математики и кибернетики

Отчёт по дисциплине «Верификация ПО»

ТЕСТИРОВАНИЕ ПРОГРАММЫ ДЛЯ ФАКТОРИЗАЦИИ МНОГОЧЛЕНОВ

Выполнил:
студент 6 курса группы 22609 Р. И. Кадиров

_____
подпись

Лектор:
к.ф-м.н., доцент К. А. Кулаков
Итоговая оценка: _____

_____
подпись

Петрозаводск

2015

Глава 1

Объект исследования

В ходе выполнения работы рассматривалось приложение Kronecker, предназначенное для разложение многочлена на неприводимые множители (факторизации) [1] согласено алгоритму Кронекера [2]. То есть программа находит корни многочлена $\xi_i, i = 1..k$ и возвращает ответ в виде произведения неприводимых множителей:

$$(x - \xi_1) \cdot (x - \xi_2) \cdot ... \cdot (x - \xi_k)$$

Программа реализована при помощи языка программирования Си++. В качестве вспомогательных библиотек использовались библиотеки MPIR и FLINT. Первая позволяет работать с длинной арифметикой, а вторая предоставляет структуры и базовые действия для работы с многочленами.

В программу на входе подаётся текстовый файл, в котором записан многочлен. Формат записи таков, что в первой строке написана степень полинома, а во второй строке через пробел перечислены коэффициенты полинома от младшей степени к старшей. Программа обрабатывает многочлен и записывает ответ в выходной файл см. рис. 1.1.

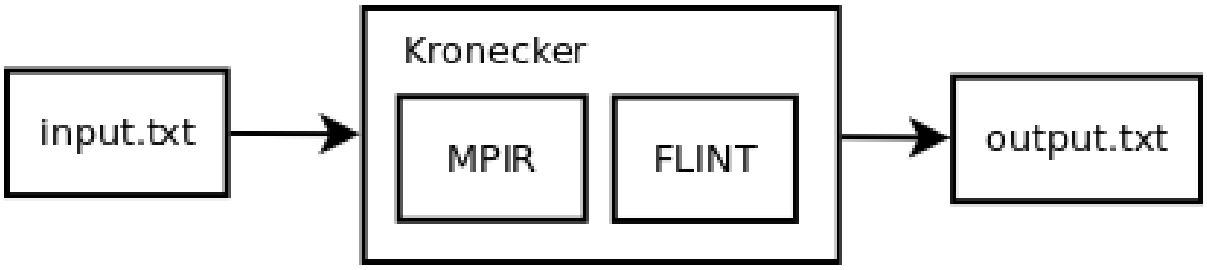

Рис. 1.1: Высокоуровневая архитектура проекта

Глава 2

План тестирования

2.1 Описание глобальных данных

В программе имеются глобальные структуры и переменные, которые будут затрагиваться в ходе тестирования программы. Они будут отмечаться в соответствующих тестах, как входные или выходные параметры, в зависимости от того читаем мы данные из них или записываем. В программе имеются следующие глобальные данные:

- mpz_t **U_list — матрица, хранящая числа для декартового произведения.
- mpz_t *Factors — массив, хранящий факторизацию числа в виде простых чисел.
- int *Powers — массив, хранящий степени простых чисел в факторизации числа.
- long long *sizerOfU — массив, хранящий количество элементов в декартовом произведении.
- int sizeOfCartProd — размер декартового произведения
- int degree — степень полинома, который подают на вход.

2.2 Описание функций

В программе будут тестироваться следующие функции, которые в случае успеха возвращают 0, а в случае ошибки возвращают -1 и выводят сообщение в поток ошибок:

- int main (int args_num, char **input_file) — главная функция программы. В ней происходит извлечение многочлена из файла и преобразование его в структуры библиотеки FLINT.
  - int args_num — количество аргументов, передающихся в программу.
  - char **input_file — имя входного файла.
- int kronecker (fmpz_poly_t polynom, fmpz_poly_t ans_poly) — функция, которая выполняет факторизацию многочлена.
  - fmpz_poly_t polynom — структура, хранящая входной полином.
  - fmpz_poly_t ans_poly — структура, хранящая неразложимый полином.
- mpz_t **mpz_cart_prod (int n) — декартово произведение списков. Структуры списков вынесены в глобальные переменные, но по смыслу являются входными параметрами.
  - int n — число элементов в списке для декартового произведения.
- int factor (int s, mpz_t n) — факторизация числа.
  - int s — индекс для записи в массив Factors.
  - mpz_t n — число для факторизации.
- int newthon_interpolation (fmpz_poly_t poly, mpz_t *ys, int size) — интерполяция по формуле Ньютона.
  - fmpz_poly_t poly — структура, для хранения интерполированного многочлена.
  - mpz_t *ys — набор значений функции, по которым интерполируем.
  - int size — количество точек интерполяции.

2.3 Стратегия тестирования

Для тестирования использовался фреймворк CUnit, который обладает простым и надёжным функционалом для работы с кодом на Си/Си++. CUnit не обладает встроенным графическим интерфейсом, поэтому необходимо работать с терминалом, что вполне удобно в системах UNIX-like.

Для организации тестирования необходимо создать тестовый модуль и набор функций-тестов для соответствующих тестируемых функций. Все тесты строятся на «утверждениях», которые обеспечивают проверку необходимых условий. В утверждения можно помещать различные выражения, значения, ожидаемый и реальный результат, а также многое другое, что позволяет назвать CUnit достаточно гибким фреймворком. Само тестирование будет организовано по принципу «белого ящика».

На рис. 2.1 представлен граф вызовов функций тестируемой программы. Интеграционное тестирование будет проводиться путём последовательного тестирования различных сочетаний данных функций. Все интеграционные тесты удобнее всего разбить на группы в соответствии с определённым сочетанием тестируемых функций:

1. kronecker + factor
2. kronecker + factor + mpz_cart_prod
3. kronecker + factor + mpz_cart_prod + newthon_interpolation
4. main + kronecker + factor + mpz_cart_prod + newthon_interpolation

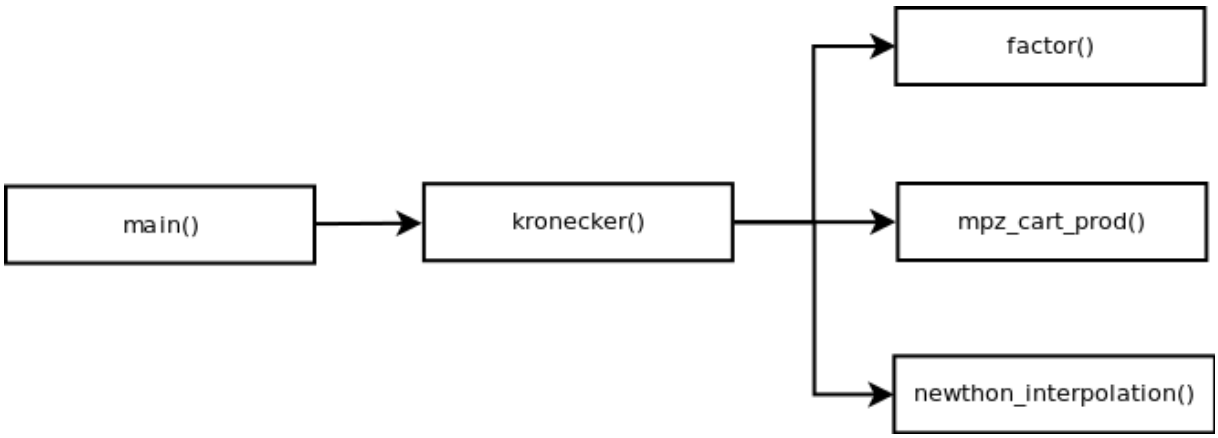

Рис. 2.1: Граф вызовов

Для проведения аттестационного тестирования программа должна выполнять факторизацию полинома, получаемого из файла, и записывать ответ в выходной файл. Если полином не разлагается на множители, то программа должна вернуть исходный полином в качестве ответа.

2.4 Подход к тестированию

Тестирование будет автоматизированным, благодаря фреймворку CUnit. Тестировщик лишь должен инициировать процесс тестирования и дождаться результата, где будет список пройденных и непройденных тестов. В случае ошибки необходимо составить отчёт об ошибке.

Критерий прохождения тестов: Тест будет считать успешно пройденным, если совпадают значения ожидаемых и фактических результатов.
Критерий приостановления тестирования: Тестирование можно считать не пройденным, если будет найдена хотя бы одна ошибка. Такой жёсткий критерий необходим, чтобы обеспечить надёжную работу приложения, которое в идеале может работать сутками (но реализованный алгоритм этого не позволяет). Стоит отметить, что сам проект небольшой, поэтому ошибок в принципе много не может быть.
Критерий возобновления работы: Тестирование возобновляется, если найденные ошибки были исправлены.

Глава 3

Тесты

Шаблон описания теста:

Тест: <номер>

Тип теста: <тип>

Описание: <описание>

Глобальные данные: <название структур и переменных>

Входные данные: <вход>

Ожидаемый результат: <выход>

Где:

<номер> – номер (идентификатор) теста.

<тип> – тип теста: простой, общий, краевой, негативный, специальный.

<описание> – краткое описание теста.

<вход> – входные данные в виде набора конкретных значений, ограничений или условий на данные.

<выход> – ожидаемый результат.

3.1 Блочные тесты

Метод int factor (int s, mpz_t n)

Входные параметры:

- int s — индекс для записи в массив Factors. Допустимые значения от 0 и до количество цифр в n в двоичной системе счисления, укладывающее в int.

- mpz_t n — число для факторизации. Ограничений на значение нет.

Тест: 1

Тип теста: Негативный

Описание: Попытаться записать полученную факторизацию числа в отрицательный индекс массива

Глобальные данные: Factors, Powers, U_list

Входные данные: s $<$ 0, любое n

Ожидаемый результат: Возвращаемое значение -1. Вывод сообщения в потоке ошибок об ошибке работы с памятью

Тест: 2

Тип теста: Краевой, простой

Описание: Попытаться записать полученную факторизацию числа в нулевой индекс массива

Глобальные данные: Factors, Powers, U_list

Входные данные: s = 0, любое n

Ожидаемый результат: Возвращаемое значение 0. Успешная запись факторизации числа

Тест: 3

Тип теста: Краевой, негативный

Описание: Попытаться записать полученную факторизацию числа в индекс массива, под который не выделена память

Глобальные данные: Factors, Powers, U_list

Входные данные: s $>$ количество цифр в n, n

Ожидаемый результат: Возвращаемое значение -1. Вывод сообщения в потоке ошибок об ошибке работы с памятью

Тест: 4

Тип теста: Простой

Описание: Попытаться факторизовать ноль

Глобальные данные: Factors, Powers, U_list

Входные данные: s = 0, n = 0

Ожидаемый результат: Возвращаемое значение 0. U_list[0][0] = 0, Factors[0] = 0, Power[0] = 0

Тест: 5

Тип теста: Простой

Описание: Попытаться факторизовать положительное число

Глобальные данные: Factors, Powers, U_list

Входные данные: s = 0, n > 0

Ожидаемый результат: Возвращаемое значение 0. В Factors записана правильная факторизация числа.

Тест: 6

Тип теста: Простой

Описание: Попытаться факторизовать отрицательное число

Глобальные данные: Factors, Powers, U_list

Входные данные: s = 0, n < 0

Ожидаемый результат: Возвращаемое значение 0. В Factors записана правильная факторизация числа.

Метод mpz_t **mpz_cart_prod (int size)

- int n — число элементов в списке для декартового произведения. Изменяется от 0 и до максимального значения int.

Тест: 7

Тип теста: Негативный

Описание: Попытаться найти произведение для списка отрицательной длины

Глобальные данные: U_list

Входные данные: size < 0

Ожидаемый результат: Вернуть пустой список. Сообщение об ошибке работы с памятью

Тест: 8

Тип теста: Краевой, негативный

Описание: Попытаться найти произведение для списка нулевой длины

Глобальные данные: U_list

Входные данные: size = 0

Ожидаемый результат: Вернуть пустой список. Сообщение об ошибке работы с памятью

Тест: 9

Тип теста: Простой

Описание: Попытаться найти произведение для списка положительной длины

Глобальные данные: U_list

Входные данные: size > 0

Ожидаемый результат: Вернуть список с декартовым произведением.

Метод int newthon_interpolation (fmpz_poly_t poly, mpz_t *ys, int size)

- fmpz_poly_t poly — структура, для хранения интерполированного многочлена.
- mpz_t *ys — набор значений функции, по которым интерполируем. Должен быть не пустой указатель.
- int size — количество точек интерполяции. Должно быть больше нуля.

Тест: 10

Тип теста: Негативный, специальный

Описание: Попытаться дать для записи входной полином, степень которого не совпадает с количеством точек интерполяции

Входные данные: Структура для хранения полинома poly со степенью != size, точки для интерполяции, количество точек

Ожидаемый результат: Вернуть -1. Сообщение о соответствующей ошибке в потоке ошибок.

Тест: 11

Тип теста: Негативный

Описание: Передать NULL-указатель в качестве списка с точками интерполяции

Входные данные: Структура для хранения полинома, NULL-список с точками интерполяции, количество точек

Ожидаемый результат: Вернуть -1. Сообщение об ошибке: «список с точками пуст» в потоке ошибок.

Тест: 12

Тип теста: Негативный

Описание: Передать отрицательное число количества точек интерполяции

Входные данные: Структура для хранения полинома poly, точки для интерполяции, size < 0

Ожидаемый результат: Вернуть -1. Сообщение об ошибке: «число точек меньше единицы» в потоке ошибок.

Тест: 13

Тип теста: Негативный

Описание: Передать число точек равное нулю

Входные данные: Структура для хранения полинома poly, список с точками интерполяции, size $= 0$

Ожидаемый результат: Вернуть -1. Сообщение об ошибке: «число точек меньше единицы» в потоке ошибок.

Метод int kronecker (fmpz_poly_t polynom, fmpz_poly_t ans_poly)

- fmpz_poly_t polynom — структура, хранящая входной полином.
- fmpz_poly_t ans_poly — структура, хранящая неразложимый полином.

Тест: 14

Тип теста: Простой

Описание: На вход подали нулевой полином

Входные данные: Нулевой входной полином, полином для ответа

Ожидаемый результат: Успешное завершение работы. В качестве ответа будет нулевой полином.

Тест: 15

Тип теста: Простой

Описание: На вход подали тривиальный полином

Входные данные: Тривиальный полином, полином для ответа

Ожидаемый результат: Успешное завершение работы. В качестве ответа будет тривиальный полином.

Тест: 16

Тип теста: Простой

Описание: На вход подали линейный полином

Входные данные: Линейный полином, полином для ответа

Ожидаемый результат: Линейным полином вернётся в качестве ответа

Тест: 17

Тип теста: Простой

Описание: На вход подаётся полином, который разлагается на N линейных множителей

Входные данные: Полином, разлагаемый на N линейных множителей, полином для ответа

Ожидаемый результат: Успешное завершение и возврат ответа в виде N линейных множителей

Тест: 18

Тип теста: Простой

Описание: На вход подаётся неразложимый полином

Входные данные: Неразложимый полином, полином для ответа

Ожидаемый результат: Вернётся неразложимый полином

Тест: 19

Тип теста: Специальный

Описание: Разложимый полином подаётся на вход, но при этом разлагается не имеет линейных или тривиальных множителей

Входные данные: Разложимый полином, полином для ответа

Ожидаемый результат: Полином должен вернуться в исходном виде, так как само разложение происходит при участии остальных функций, а вместо них заглушки.

Метод int main (int args_num, char **input_file)

- int args_num — количество аргументов, передающихся в программу.
- char **input_file — имя входного файла.

Тест: 20

Тип теста: Негативный

Описание: На вход подаём пустой файл

Входные данные: Пустой файл, количество аргументов равно единице

Ожидаемый результат: Вернуть -1. Ошибка в поток ошибок: «На вход подали пустой файл»

Тест: 21

Тип теста: Негативный

Описание: Подать на вход файл, в котором степень полинома меньше, чем количество коэффициентов

Входные данные: Файл с недостаточным количеством коэффициентов, количество аргументов равное единице

Ожидаемый результат: Вернуть -1. Ошибка в поток ошибок: «Степень полинома не соответствует количеству коэффициентов»

Тест: 22

Тип теста: Негативный

Описание: Подать на вход файл с испорченным форматом. Это могут быть буквы, специальные символы, либо вообще случайный набор символов

Входные данные: Файл с испорченным форматом, количество аргументов равное единице

Ожидаемый результат: Вернуть -1. Ошибка в поток ошибок: «Ошибка формата исходных данных»

Тест: 23

Тип теста: Негативный

Описание: Ввести количество аргументов не равное единице, то есть ноль или больше одного

Входные данные: Полином в файле, количество аргументов не равное единице

Ожидаемый результат: Вернуть -1. Ошибка в поток ошибок: «Слишком много/-мало аргументов»

Тест: 24

Тип теста: Простой

Описание: Подать на вход корректный полином и один аргумент

Входные данные: Полином в файле, один аргумент

Ожидаемый результат: Пустой выходной файл

3.2 Интеграционные тесты

Группа взаимодействия: kronecker + factor

Тест: 25

Тип теста: Простой

Описание: На вход подаётся полином. Убедиться, что функция factor факторизует необходимые числа

Входные данные: Входной полином, полином для ответа, список для хранения факторизованных чисел

Ожидаемый результат: Функция factor сохранит полученные факторизации чисел в списке, а функция kronecker вернёт с код ошибки -1

Тест: 26

Тип теста: Специальный

Описание: На вход необходимо подать полином, который является тривиальным или состоящим из N-го числа линейных множителей

Входные данные: Полином, состоящий из линейных множителей или тривиальный, полином для ответа

Ожидаемый результат: В полином для ответа будет сохранения факторизация

Группа взаимодействия: kronecker + factor + mpz_cart_prod

Тест: 27

Тип теста: Негативный

Описание: На вход необходимо подать любой полином, чтобы убедиться, что функция mpz_cart_prod не в состоянии работать без функции factor

Входные данные: Входной полином, выходной полином

Ожидаемый результат: Ошибка с кодом возврата -1

Группа взаимодействия: kronecker + factor + mpz_cart_prod + newthon_interpolation

Тест: 28

Тип теста: Общий

Описание: Убедиться, что происходит один шаг факторизации

Входные данные: Входной полином, выходной полином

Ожидаемый результат: Полином, у которого выделен один множитель

Тест: 29

Тип теста: Краевой

Описание: Убедиться, что случай с нулевым полиномом обрабатывается корректно

Входные данные: Нулевой полином, выходной полином

Ожидаемый результат: Нулевой полином

Группа взаимодействия: main + kronecker + factor + mpz_cart_prod + newthon_interpolation

Тест: 30

Тип теста: Краевой

Описание: Убедиться, что случай с нулевым полиномом нормально сохраняется в

файл

Входные данные: Файл с нулевым полиномом на входе, выходной файл

Ожидаемый результат: Ноль в выходном файле

3.3 Аттестационное тестирование

Тест: 1

Тип теста: Специальный

Описание: На вход подаём нулевой полином

Входные данные: Файл с полиномом 0, выходной файл

Ожидаемый результат: В выходном файле ответ 0

Тест: 2

Тип теста: Специальный

Описание: На вход подаём тривиальный полином

Входные данные: Файл с полиномом x, выходной файл

Ожидаемый результат: В выходном файле ответ x

Тест: 3

Тип теста: Специальный

Описание: На вход подаём произведение двух линейных множителей

Входные данные: Файл с полиномом $x^2 - 1$, выходной файл

Ожидаемый результат: В выходном файле ответ $(x + 1) \cdot (x - 1)$

Тест: 4

Тип теста: Специальный

Описание: На вход подаём полином вида $x^n + const$

Входные данные: Файл с полиномом $x^7 + 1$, выходной файл

Ожидаемый результат: В выходном файле ответ $(x+1)(x^6-x^5+x^4-x^3+x^2-x+1)$

Тест: 5

Тип теста: Специальный

Описание: На вход подаём квадрат двух неразложимых полиномов

Входные данные: Файл с полиномом $4225x^4 + 1560x^2 + 144$, выходной файл

Ожидаемый результат: В выходном файле ответ $(12 + 65 * x^2)^2$

Тест: 6

Тип теста: Специальный

Описание: На вход подаём полином 10-й степени, который является теоретическим ограничением для алгоритма Кронекера

Входные данные: Файл с полиномом $400x^10 - 40x^7 + 160x^6 + 320x^5 + x^4 - 8x^3 + 64x + 64$, выходной файл

Ожидаемый результат: В выходном файле ответ $(20x^5 - x^2 + 4x + 8)^2$

Тест: 7

Тип теста: Общий

Описание: На вход подаём любой полином

Входные данные: Файл с любым полиномом, выходной файл

Ожидаемый результат: В выходном файле должен появиться какой-нибудь ответ. Явных ошибок быть не должно.

Глава 4

Покрытие кода

Расчёт тестового покрытия относительно исполняемого кода ПО проводился по формуле:

$$T_{cover} = \frac{L_{tc}}{L_{code}} \cdot 100\%,$$

где

T_{cover} – тестовое покрытие в процентах,

L_{tc} – количество строк кода, покрытых тестами,

L_{code} – общее количество строк кода.

В случае с данным ПО получились следующие значения:

$T_{cover} \approx 86\%$

$L_{tc} = 458$

$L_{code} = 532$

Глава 5

Примеры реализации тестов

В трёх тестах в листинге 5.1 показан пример реализации трёх блочных тестов, которые проверяют функцию factor на корректность работы с различного рода числами.

Listing 5.1: Пример CUnit тестов

```
void test1(void)
{
    mpz_t num;
    mpz_init(num);

    mpz_set_ui(num, -9999999);
    CU_ASSERT(factor(0, num) == 0);
}
void test2(void)
{
    mpz_t num;
    mpz_init(num);

    mpz_set_ui(num, 9999999);
    CU_ASSERT(factor(0, num) == 0);
}
void test3(void)
{
    mpz_t num;
```

```
    mpz_init(num);

    mpz_set_ui(num, 0);
    CU_ASSERT(factor(0, num) == 0);
}
```

Глава 6

Отчёт о выполнении тестирования

Таблица 6.1: Результаты блочного тестирования

Функция	**Кол-во тестов**	**Кол-во ошибок**	**Дата проведения**
factor	6	1 (отчёт об ошибке №1)	02.01.2015
mpz_cart_prod	3	0	02.01.2015
newthon_interpolation	4	0	02.01.2015
kronecker	6	0	02.01.2015
main	5	0	02.01.2015

Таблица 6.2: Результаты интеграционного тестирования

Группа взаимодействия	**Кол-во тестов**	**Кол-во ошибок**	**Дата проведения**
kronecker + factor	2	0	02.01.2015
kronecker + factor + mpz_cart_prod	1	0	02.01.2015
kronecker + factor + mpz_cart_prod + newthon_interpolation	2	0	02.01.2015
kronecker + factor + mpz_cart_prod + newthon_interpolation	1	0	02.01.2015

Таблица 6.3: Результаты аттестационного тестирования

Номер теста	**Результат**	**Дата проведения**
1	Пройден	02.01.2015
2	Пройден	02.01.2015
3	Пройден	02.01.2015
4	Пройден	02.01.2015
5	Пройден	02.01.2015
6	Пройден	02.01.2015

Глава 7

Отчёт об ошибке №1

Не работает функция factor.

Тесты Тест 1: попытка записать факторизованные числа в список по отрицательному индексу.

Условия Версия библиотеки FLINT 2.4.4, версия библиотеки MPIR 2.6.0, ОС Linux Mint 17 с использованием QTCreator, CUnit 2.1.3

Приоритет Критический

Алгоритм воспроизведения ошибки

1. Создать список для хранения факторизованных чисел.

2. Запустить функцию factor с отрицательным параметром индекса

Ожидаемый результат: Получить сообщение об ошибке работы с памятью

Фактический результат: Ошибка сегментирования и экстренное завершение программы

Повторяемость: Всегда

Дата: 02.01.2015

Глава 8

Результаты

В ходе тестирования была обнаружена единственная ошибка, которая была найдена при помощи блочного тестирования:

- Отсутствие обработки в функции factor при попытке записать факторизованные числа в список по отрицательному индексу.

При интеграционном и аттестационном тестировании ошибок выявлено не было. Найденная ошибка в блочном тестировании функции factor является критической и требует немедленного устранения. В целом же можно с уверенностью сказать, что программа работает достаточно корректно и способна обрабатывать даже самые неожиданные ситуации.

Библиографический список использованной литературы

[1] А. Акритас, “Основы компьютерной алгебры с приложениями,” *Москва «Мир»*, 1994, pp. 383–384.

[2] Е.В. Панкратьев,“Элементы компьютерной алгебры,” *Интернет-университет информационных технологий; БИНОМ. Лаборатория знаний*, 2007, стр. 148–149.