УО «Белорусский государственный университет информатики и радиоэлектроники»

Факультет информационных технологий и управления

|  |  |
| --- | --- |
|  | «УТВЕРЖДАЮ»  Заведующий кафедрой ИТАС  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_НавроцкийА.А.  «05» марта\_2015 г. |

**З А Д А Н И Е**

на курсовой проект

Студенту гр. 420602.

1. Тема работы Сравнение итерационных методов уточнения корней: метод простой метод секущих и метод Вегстейна.

2. Дата сдачи студентом работы 24.05.2015

3. Исходные данные для работы Решить задачу .

4. Содержание расчетно-пояснительной записки

   Введение

1.Аналитический способ отделения корней

2.Численные методы уточнения корней

3.Програмная реализация

Заключение

Список используемой литературы

   Приложение (блок-схемы алгоритмов, текст программы).

5. Перечень графического материала Блок-схемы алгоритмов, используемых для решения задачи, графики используемых функций

6. Консультант по работе Шатилова О. О.

7. Дата выдачи задания 05.03.2015.г.

8. Календарный график выполнения курсовой работы

до 19.03.2015 г. сделать обзор существующих методов решения задачи;

до 16.04.2015 г. написать программу решения задачи;

до 28.05.2015 г. получить результаты, оформить пояснительную записку.

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ О. О. Шатилова

Заданне прыняў да выканання 05.03.2015 г.

ст. гр. 420602 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ФИО

ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 3](#_Toc420453762)

[1 АНАЛИТИЧЕСКИЙ СПОСОБ ОТДЕЛЕНИЯ КОРНЕЙ 4](#_Toc420453763)

[2 ЧИСЛЕННЫЕ МЕТОДЫ УТОЧНЕНИЯ КОРНЕЙ 8](#_Toc420453764)

[2.1. Метод секущих 8](#_Toc420453765)

[2.2. Метод Вегстейна 10](#_Toc420453766)

[3 ПРОГРАМНАЯ РЕАЛИЗАЦИЯ 12](#_Toc420453767)

[ЗАКЛЮЧЕНИЕ 14](#_Toc420453768)

[ЛИТЕРАТУРА 17](#_Toc420453769)

[Приложения А. Блок-схемы алгоритмов. 18](#_Toc420453770)

[Приложения Б. Код программы на С. 23](#_Toc420453771)

# ВВЕДЕНИЕ

В соответствии с вариантом задания нужно реализовать два итерационных метода нахождения корней: метода секущих и метод Вегстейна.

Реализация будет выполнена на высокоуровневом языке программирования С. Данный язык идеально подходит под задачу быстрого нахождения корней, так как в современном мире является одним из самых быстро-выполняемых языков программирования. Так же отличительной чертой будет являться полная реализация алгоритмов нахождения корней, без использования каких либо библиотечных готовых решений и модулей, что позволит однозначно сравнить количество итераций и время выполнения. Для высчитывания погрешности и построения графиков будем использовать систему компьютерной алгебры – Mathcad. Основные графики и вычисления будут произведены именно в этом ПО.

Сама задача нахождения корней уравнения очень распространена, и на данный момент имеется большое количество алгоритмов нахождения корней: В частности, среди них выделяют итерационные методы приближенного нахождения корней. Суть таких методов можно описать одним предложением: «Суть метода заключается в нахождении по приближённому значению величины следующего приближения (являющегося более точным)».

Среди самих итерационных методов выделяют метод деления отрезка пополам, метод хорд, метод простой итерации, метод Ньютона, метод секущих, метод Вегстейна, метод парабол и др. В данной работе будут рассматриваться и сравниваться два итерационных метода: метод простой итерации и метод Вегстейна.

При решении таких задач с помощью ЭВМ важно рассмотреть такие параметры алгоритмов как число итераций, время и точность найденного корня, после чего можно выбрать наиболее подходящий под конкретную задачу алгоритм.

Для реализации самого проекта будет использоваться инструмент разработки на языке С – Visual Studio 2012. В нем представлены все основные инструменты для разработки, в частности очень удобный и продуманный debug модуль, что существенно упростит написание курсового проекта.

После анализа литературы, мною было выделено три основные позиции, которые необходимо рассмотреть при работе с итерационными методами уточнения корней:

* Аналитический способ отделения корней.
* Численные методы уточнения корней.
* Программная реализация вычислительного процесса.

|  |  |  |
| --- | --- | --- |
|  |  |  |

1. АНАЛИТИЧЕСКИЙ СПОСОБ ОТДЕЛЕНИЯ КОРНЕЙ

В данном разделе будет рассмотрен способ отделения корней т.е способ выделения интервала с одним действительным корнем.

Входными данными для решения задачи нахождения корней будет являться алгебраические уравнения (в канонической форме):

*аn⋅x n + an-1⋅ x n-1 + ... + a1⋅x + a0 = 0*

Для примера рассмотрим упрощенный вариант алгебраического уравнения для некоторой функции *y*, заданной как:

*f(x) = cos(x)*

Таким образом нашей задачей будет являться нахождение точек пересечения графика функции *f(x)* с осью абсцисс, то есть нахождение нулей функции.



Рисунок 1.1 Нули функции

Однако для нахождения этих корней на ЭВМ, для начала необходимо выделить некоторые интервалы, на которых однозначно существует единственный корень. Необходимость выделения интервалов решает две проблемы. Первая – это существуют ли решение уравнения, то есть существуют ли действительные корни. Если не будет найдено ни одного интервала – можно сделать вывод, что корней нет. Вторая проблема возникает при анализе итерационных алгоритмов решения уравнения. Данные алгоритмы рассчитаны на нахождение первого и единственно корня, соответственно нам необходимо выделить интервал содержащий один корень, а затем применить итерационный алгоритм на этом интервале. Графическое преставления выделения интервалов представлено на рисунке 1.2:

Рисунок 1.2 Выделение корней

При выделении интервала мы получаем некоторые значения по оси абсцисс. В нашем случае это точки a и b. То есть функция будет иметь корень на интервале [a; b]. Но как выделить такой интервал?

Для программного выделения такого интервала заметим, что функция при пересечении оси абсцисс изменила свой знак. Таким образом достаточно проверить условие:



Если условие выполняется, то функция однозначно будет иметь корень на интервале [a; b]. Далее необходимо максимально сузить этот интервал, что бы было наиболее легко (меньше итераций) искать приближение корня на данном отрезке.

Для уточнения интервала вводится некоторая константа (*e*) – точность нахождения границ интервала. Так же выбирается некоторый шаг функции *h.* Как правило его берут равным одной десятой от точности *e.* Далее начиная со значения *a* мы идем по оси абсцисс с шагом *h*, и на каждом шаге вычисляем значение функции *f(x)*. Как только модуль значения функции становится меньшим заданной точности, мы запоминаем значение по оси абсцисс и продолжаем наращивать значение по *x* сшагом *h.* Как только модуль значения функции стал больше значения заданной точности *e*, мы повторно запоминаем значение по оси абсцисс. Запомненные значения и будут являться новым уточненным интервалом с точность *e.* Графическое представление данного метода отделения корней представлено на рисунке 1.3:



Рисунок 1.3 Метод отделения корней

На рисунке 1.3 показано движение по функции *f(x) = cos(x)* c шагом *h*. Первое запоминаемое значение меньшее *e* – start. Второе запоминаемое значении большее *e* – *end*. Таким образом интервал на котором однозначно будет хотя бы один корень – *[start; end]* (при условии, что *f(start)\*f(end) < 0*). После анализа всей функции мы получаем список интервалов, на которых однозначно будет корень. Далее применяя итерационные методы уточнения корней найдем (с определенной погрешностью) корень.

1. ЧИСЛЕННЫЕ МЕТОДЫ УТОЧНЕНИЯ КОРНЕЙ

В данном разделе будут рассмотрено математическое обоснование методов приближенного нахождения корней: метода секущих, метода Вегстейна.

* 1. Метод секущих

Как отмечалось выше, необходимо найти нуль функции *f(x)*. Также на этапе отделения интервалов мы получили две точки (границы интервала), обозначим их как *C1(x1; y1)* и *C2(x2;y2)* и проведем через них прямую. Она пересечет ось абсцисс в точке *(x3;0)*. Теперь найдем значение функции с абсциссой *x3*. Временно будем считать *x3* корнем на отрезке *[x1; x2]* . Пусть точка C3 имеет абсциссу X3 и лежит на графике. Теперь вместо точек *C1*  и *C2* мы возьмём точку *C3* и точку *C2*. Теперь с этими двумя точками проделаем ту же операцию и так далее, то есть будем получать две точки *Cn+1*  и *Cn*  и повторять операцию с ними. Отрезок, соединяющий последние две точки, пересекает ось абсцисс в точке, значение абсциссы которой можно приближённо считать корнем. Эти действия нужно повторять до тех пор, пока не получим значение корня с нужным приближением.

Графически метод секущих представлен на рисунке 2.1:

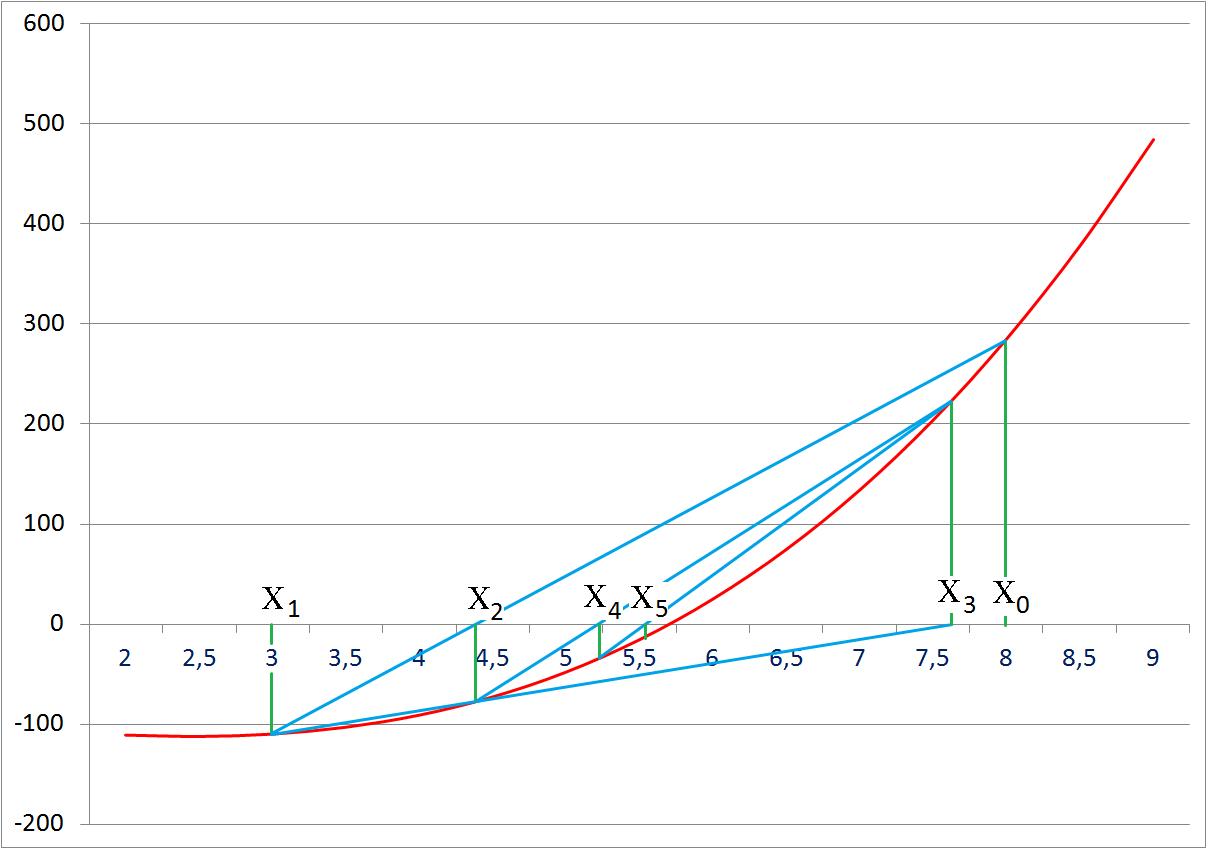


Рисунок 2.1 Графическое представление метода секущих

На рисунке 2.1 мы видим, как уточняется корень от значения X0 до X5 (значение функции приближается к нулю).

Рассмотрим математическое обоснование данного алгоритма.

Пусть *x1, x2* — абсциссы концов хорды, *y = kx+b -* уравнение секущей, содержащей хорду. Найдем коэффициенты *k* и *b* из системы уравнений:

*  
     \left\{  
     \begin{array}{rcl}  
      f(x_1) & = & kx_1+b,\\  
      f(x_2) & = & kx_2+b. \\  
     \end{array}   
     \right.  
  *.

Вычтем из первого уравнения второе:

f(x_1)-f(x_2)=k(x_1-x_2)

Затем найдем коэффициенты k и b:

k=\frac{f(x_2)-f(x_1)}{x_2-x_1}

Тогда:

b=f(x_1)-\frac{(f(x_2)-f(x_1))x_1}{x_2-x_1}.

Уравнение принимает вид:

y=\frac{f(x_2)-f(x_1)}{x_2-x_1}(x-x_1)+f(x_1)

Таким образом, теперь можем найти первое приближение к корню, полученное методом секущих:

x_3=x_1-\frac{(x_2-x_1)f(x_1)}{f(x_2)-f(x_1)}

Теперь возьмем координаты *x2* и *x3* и повторим все проделанные операции, найдя новое приближение к корню. Таким образом, итерационная формула метода секущих имеет вид:

 x_{i+1}=x_{i-1}-\dfrac{f(x_{i-1})\cdot (x_i-x_{i-1})}{f(x_i)-f(x_{i-1})}

Повторять операцию следует до тех пор, пока |*xi - xi-2*|  не станет меньше или равно заданному значению погрешности.

На основании выше сказанного можно составить алгоритм для нахождения корней уравнений:

* + - 1. Пользователь должен определить интервал, на котором он хочет найти корни уравнения.
      2. Данный интервал разбивается по методу отделения корней т.е на данном шаге определяется набор интервалов, на которых есть однозначно один корень. Если интервалов нет, значит уравнение не имеет действительных корней.
      3. На каждом интервале применяется метод секущих для нахождения корня:

1. Берем границы интервала как две точки *xi-1*и *xi*
2. Высчитываем третью точку *xi+1* по формуле:

 x_{i+1}=x_{i-1}-\dfrac{f(x_{i-1})\cdot (x_i-x_{i-1})}{f(x_i)-f(x_{i-1})}

1. Заменяем *xi-1 на xi; xi* на *xi+1*
2. Высчитываем разницу между двумя соседними точками, и если модуль разницы больше заданной точности *e* то возвращаемся к шагу b, если нет то выводим значение точки *xi+1*
   1. Метод Вегстейна

Метод Вегстейна эффективен для нахождения корней уравнений вида *x = g(x)*. Однако для нахождения корней уравнений вида *0 = f(x)* по своей сути является просто модификацией рассмотренного ранее метода секущих[1]. Кратко рассмотрим приведенное доказательство:

Дано уравнение:

*x = g(x) (1)*

Итерационная итерационный процесс метода Вегстейна имеет вид:

1. *x0*
2. *x1 = g(x0)*
3. *(2)*

Для случая когда решаем уравнение вида *f(x) =0 -* заменяя уравнение (1) эквивалентным ему уравнением *f(x) = g(x) - x*, и подставляя в (2) получаем:

*(3)*

Получившееся уравнение (3), является итерационным уравнением метода секущих. Все основные пункты для метода секущих мы рассмотрели ранее.

Так же метод Вегстейна иногда называют улучшенным методом простой итерации [2].

Алгоритм метода Вегстейна будет отличаться от метода секущих только формулой приближения:

1. Метод Вегстейна - это двухшаговый метод, и для начала нужно задать два приближения *х0* и *х1*
2. Вычислить новое приближение *x2* по формуле (2).
3. Присвоить *х0* значение *х1*; *х1* - значение *x2*
4. Вычислить значение в точке *x2*и вернуться на шаг 2, если модуль значения функции в точке *x2* больше заданной точности *e.*
5. Вывести корень
6. ПРОГРАМНАЯ РЕАЛИЗАЦИЯ

Для реализации поставленной задачи использовался высокоуровневый язык программирования С. Для запуска кода необходимо начать выполнения (запустить) функцию *main*. После запуска появляется меню, к котором можно выбрать одну из трех функций либо завершить программу. Затем вводится точность (*e*) и интервал, на котором должны быть найдены корни. После чего на экран выводится результаты работы алгоритмов. Консольный интерфейс представлен на рисунке 3.1.

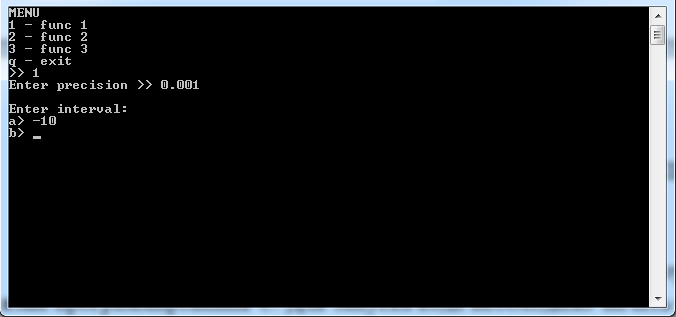


Рисунок 3.1 Консольный интерфейс

В данной программе для тестирования алгоритмов использовались три функции:.

После установления интервала и выбора точности, вызывается функция разбиение на подинтервалы - *get\_intervals*. В своей основе она использует метод отделения корней, описанный в пункте 1. Резульататом работы функции *get\_intervals* будет являться список структур, содержащих начало и конец подинтревала. Далее последовательно вызываются функции *secant\_method\_run* и *vegsteyn\_method\_run* которые проходятся по интервалам и на каждом из них вызывают соответственно функции *secant\_method* (в основе метод секущих) и *vegsteyn\_method* (в основе метод Вегстейна). Выводятся корни, значение функций при данных корнях, количество итерация для каждого из методов и затраченное время на вычисления. Результат работы программы представлен на рисунке 3.2.

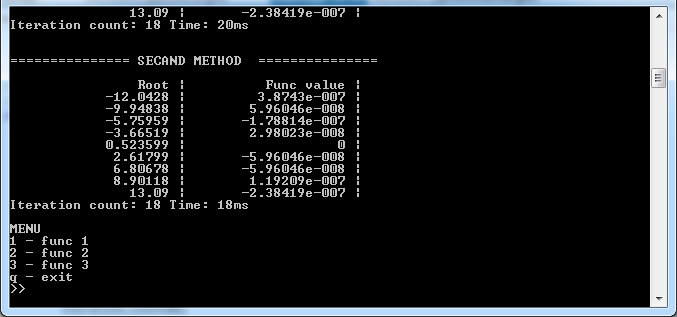


Рисунок 3.2 Результат работы программы

Полный код программы с некаторами комментариями представлен в приложении Б.

ЗАКЛЮЧЕНИЕ

В качестве вывода приведем результат работы нашей программы. Систематизируем полученные данных для трех функций, при точности e равной 0.001 и на интервале от -10 до +10. Рассмотрим функции и результаты работы программы на этих функциях:

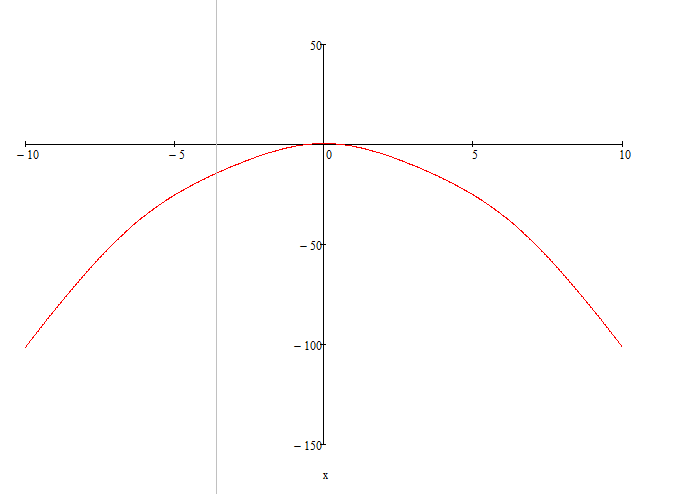


Рисунок 4.1 График функции 1.

Таблица 4.1 – Результаты вычислений

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Метод секущих | | | Метод Вегстейна | | |
| Корень | Итерации  /время | Погрешность для *f(x), 10-7* | Корень | Итерации  /время | Погрешность для *f(x), 10-7* |
| -0.58003 | 2/16ms | 4.768 | -0.58004 | 2/2ms | 4.769 |
| 0.58003 | 4.768 | 0.58004 | 4.769 |

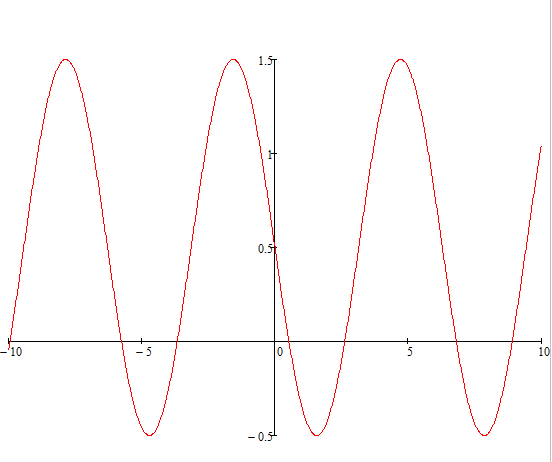


Рисунок 4.2 График функции 3

Таблица 4.3 – Результаты вычислений

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Метод секущих | | | Метод Вегстейна | | |
| Корень | Итерации  /время | Погрешность для *f(x), 10-7* | Корень | Итерации  /время | Погрешность для *f(x), 10-7* |
| -9.94838 | 14/63ms | 5.96046 | -9.94838 | 2/32ms | 5.96046 |
| -5.75959 | -1.7888 | -5.75959 | -1.7888 |
| -3.66519 | 2.98023 | -3.66519 | 2.98023 |
| 0.523599 | 0 | 0.523599 | 0 |
| 2.61799 | -0.59604 | 2.61799 | -0.59604 |
| 6.80678 | -5.96046 | 6.80678 | -5.96046 |
| 9.90118 | 1.19209 | 9.90118 | 1.19209 |

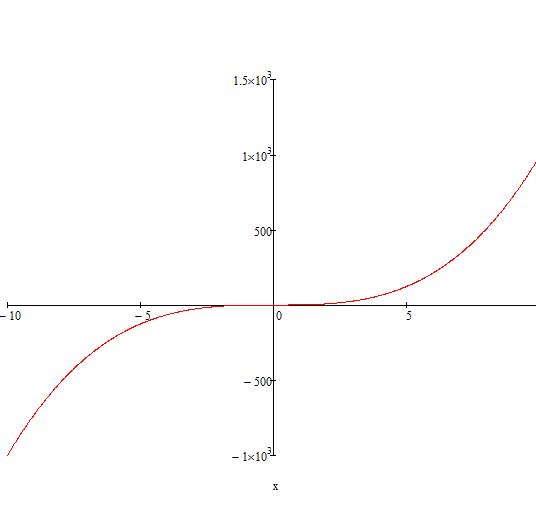


Рисунок 4.3 График функции 3

Таблица 4.3 – Результаты вычислений

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Метод секущих | | | Метод Вегстейна | | |
| Корень | Итерации  /время | Погрешность для *f(x), 10-7* | Корень | Итерации  /время | Погрешность для *f(x), 10-7* |
| -1.25992 | 1/16ms | -2.38419 | -1.25992 | 2/16ms | -2.38419 |

После анализа данных можно заметить, что данные два метода практически идентичны. Это объясняется тем, что разница между ними только в итерационной формуле. Однако метод Вегстейна при одинаковых количествах итераций показал более лучшее время.

Считаю, что поставленная задача по изучению и реализации методов итерационного приближения корней выполнена.

ЛИТЕРАТУРА

[1] Метод простых итераций [Электронный ресурс]. – Электронные

данные. –

Режим доступа : http://www.machinelearning.ru/wiki/index.php?title=

Метод\_простых\_итераций

[2] Метод Вегстейна как модификация метода секущих

[Электронный ресурс]. – Электронные данные. –

Режим доступа http://www.mathnet.ru/links/d674fbd2b3cabc25a623

ca8515f2eb19/zvmmf5969.pdf

[3] Н.Н.Калиткин. Численные методы./ Н.Н.Калиткин. - Москва 1978. –

176 c.

[4] Демидович Б.П. Основы вычислительной математики./ Демидович

Б.П., Марон И.А. — Наука, 1970. – 34 c.

[5] Бахвалов. Численные методы./ Жидков, Кобельков, Бахвалов —

Наука. – 35 с.

[6] Мышенков В.И. «Численные методы», учебное пособие/ Мышенков

Е.В, – Москва, 2001. – 12 с.

# Приложения А. Блок-схемы алгоритмов.



Рисунок 1 Блок-схема стартовых функций.

Рисунок 2 Блок-схема алгоритма отделения корней



Рисунок 3 Блок-схема прохождения итерационными методами по найденным интервалам



Рисунок 4 Блок-схема метода секущих



Рисунок 5 Блок-схема метода Вегстейна

# Приложения Б. Код программы на С.

#include <iomanip>

#include <iostream>

#include <math.h>

#include <time.h>

#include <windows.h>

using namespace std;

#define MAX\_ITER 10000 // max number of iterations

float e; // precision

float h; // step

struct Interval {

float start;

float end;

Interval\* next;

};

/////////////////////////////////////////////////

// FUNCTIONS

/////////////////////////////////////////////////

float func\_1(float x) {

//return cos(x);

return -1\*(0.5 + x\*x - cos(x));

}

float func\_2(float x) {

return (-sin(x) + 0.5);

}

float func\_3(float x) {

return x\*x\*x + 2;

}

float (\*func\_ptr)(float);

/////////////////////////////////////////////////

/////////////////////////////////////////////////

// INTERVALS SPLIT

/////////////////////////////////////////////////

Interval\* get\_intervals(float (\*func\_ptr)(float), float a, float b) {

bool find\_start\_value = false;

Interval\* first\_interval = NULL;

Interval\* last\_interval = NULL;

for(float i = a; i < b; i += h) {

if(fabs(func\_ptr(i)) < e && !find\_start\_value)

{

if(first\_interval == NULL)

{

first\_interval = new Interval;

last\_interval = first\_interval;

}

else

{

last\_interval->next = new Interval;

last\_interval = last\_interval->next;

}

last\_interval->next = NULL;

last\_interval->start = i;

find\_start\_value = true;

}

if(abs(func\_ptr(i)) > e && find\_start\_value)

{

last\_interval->end = i;

find\_start\_value = false;

//debug string

//cout << endl << "Start " << last\_interval->start << "/ End: " << last\_interval->end << endl;

}

}

// return NULL if only find start and dont find end of the interval

return find\_start\_value ? NULL : first\_interval;

}

/////////////////////////////////////////////////

/////////////////////////////////////////////////

// SECAND ALGORITHM

/////////////////////////////////////////////////

float secant\_method(float (\*func\_ptr)(float), float a, float b, int& iteration\_count) {

float x0 = a;

float x1 = b;

float new\_x0;

unsigned int cycle\_count = 0;

while(fabs(x1 - x0) > e && cycle\_count++ < MAX\_ITER)

{

new\_x0 = x1;

x1 = x1 - (x1 - x0) \* func\_ptr(x1)/(func\_ptr(x1) - func\_ptr(x0));

x0 = new\_x0;

iteration\_count++;

}

return x1;

}

void secant\_method\_run(float (\*func\_ptr)(float), Interval\* interval\_ptr) {

int iteration\_count = 0;

cout << endl << "=============== SECAND METHOD ===============" << endl << endl

<< setw(20) << "Root" << setw(2) << "|" << setw(20) << "Func value" << setw(2) << " |" << endl;

double start = GetTickCount();

while(interval\_ptr != NULL) {

if(func\_ptr(interval\_ptr->start) \* func\_ptr(interval\_ptr->end) >= 0)

{

interval\_ptr = interval\_ptr->next;

continue;

}

float root = secant\_method(func\_ptr, interval\_ptr->start, interval\_ptr->end, iteration\_count);

cout << setw(20) << root << setw(2) << "|" << setw(20) << func\_ptr(root) << " |" << endl;

interval\_ptr = interval\_ptr->next;

}

double end = GetTickCount();

cout << "Iteration count: " << iteration\_count << " Time: " << end - start << "ms" << endl << endl;

}

/////////////////////////////////////////////////

/////////////////////////////////////////////////

// VEGSTEYN ALGORITHM

/////////////////////////////////////////////////

float vegsteyn\_method(float (\*func\_ptr)(float), float a, float b, int& iteration\_count) {

float x0 = a;

float x1 = b;

float y0 = func\_ptr(a);

float y1 = func\_ptr(b);

float root, diff;

unsigned int cycle\_count = 0;

do {

root = x1 - y1\*(x1 - x0)/(y1 - y0);

diff = fabs(x1 - root);

x0 = x1;

x1 = root;

y0 = y1;

y1 = func\_ptr(root);

iteration\_count++;

cycle\_count++;

}while(diff > e && cycle\_count < MAX\_ITER);

return root;

}

void vegsteyn\_method\_run(float (\*func\_ptr)(float), Interval\* interval\_ptr) {

int iteration\_count = 0;

cout << endl << "=============== VEGSTEYN METHOD ===============" << endl << endl

<< setw(20) << "Root" << setw(2) << "|" << setw(20) << "Func value" << setw(2) << " |" << endl;

double start = GetTickCount();

while(interval\_ptr != NULL) {

if(func\_ptr(interval\_ptr->start) \* func\_ptr(interval\_ptr->end) >= 0)

{

interval\_ptr = interval\_ptr->next;

continue;

}

float root = vegsteyn\_method(func\_ptr, interval\_ptr->start, interval\_ptr->end, iteration\_count);

cout << setw(20) << root << setw(2) << "|" << setw(20) << func\_ptr(root) << " |" << endl;

interval\_ptr = interval\_ptr->next;

}

double end = GetTickCount();

cout << "Iteration count: " << iteration\_count << " Time: " << end - start << "ms" << endl << endl;

}

/////////////////////////////////////////////////

char get\_key() {

cout << "MENU" << endl

<< "1 - func 1" << endl

<< "2 - func 2" << endl

<< "3 - func 3" << endl

<< "q - exit" << endl

<< ">> ";

char c; cin >> c;

return c;

}

Interval enter\_interval() {

float a, b;

do{

cout << endl

<< "Enter interval:" << endl

<< "a> "; cin >> a;

cout << "b> "; cin >> b;

}while(a >= b);

Interval interval;

interval.start = a; interval.end = b;

return interval;

}

bool run() {

bool func\_chosen = false;

while(!func\_chosen) {

switch(get\_key()){

case '1': func\_ptr = func\_1; func\_chosen = true; break;

case '2': func\_ptr = func\_2; func\_chosen = true; break;

case '3': func\_ptr = func\_3; func\_chosen = true; break;

case 'q': return false;

}

}

cout << "Enter precision >> "; cin >> e;

h = e/10; // step

// enter base interval and func split to little intervals :)

Interval base\_interval = enter\_interval();

Interval\* intervals = get\_intervals(func\_ptr, base\_interval.start, base\_interval.end);

vegsteyn\_method\_run(func\_ptr, intervals);

secant\_method\_run(func\_ptr, intervals);

return true;

}

void main() {

while(run());

}