**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ**

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

**«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)**

Кафедра автоматизированных систем управления (АСУ)

**Решение уравнений с одной переменной**

Отчёт о лабораторной работе №1

по дисциплине «Вычислительная математика»

Выполнили

Студенты гр. 438-3

Разгуляева Т.С

Гурачевский Д.С

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2020 г.

Проверила

ассистент каф. АСУ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.Е. Косова

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г.

**Томск** **2020**

## **Задание**

Реализовать ряд методов решения уравнений: три обязательных метода (дихотомии, хорд и Ньютона) и, по желанию, три дополнительных (комбинированный метод, метод итераций и золотого сечения).

|  |  |
| --- | --- |
| N | – номер метода т.е. 1 – дихотомии, 2 – хорд и т.д.; |
| f(x) | – исследуемая функция в аналитическом виде; |
| a b | – границы отрезка; |
| Ε | – требуемая точность решения. |

Входные данные:

Выходные данные:

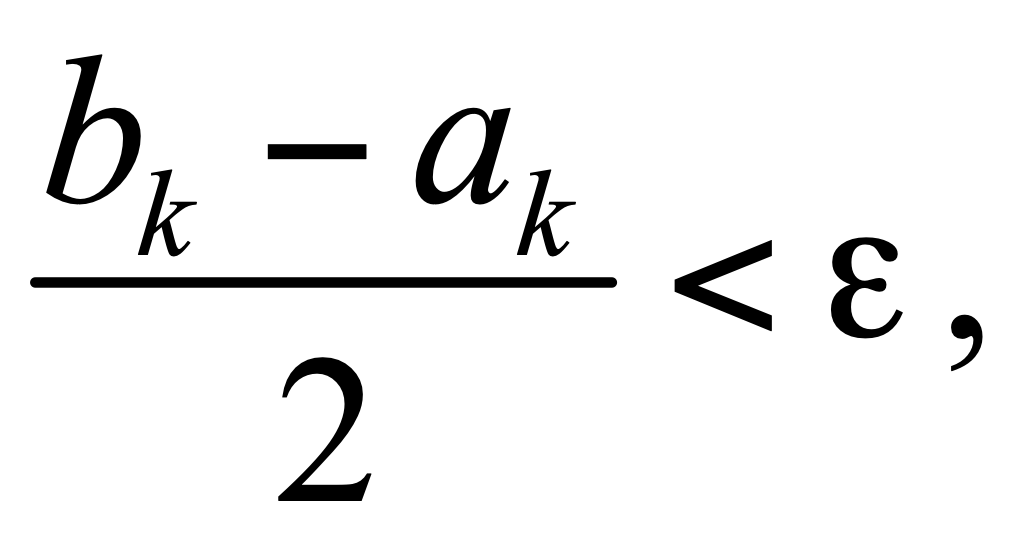
|  |  |
| --- | --- |
| x\* | – решение уравнения; |
| f(x\*) | – значение функции в найденной точке x\*; |
| ε\* | – погрешность полученного решения. |

## **Теория**

## **Интервальные методы**

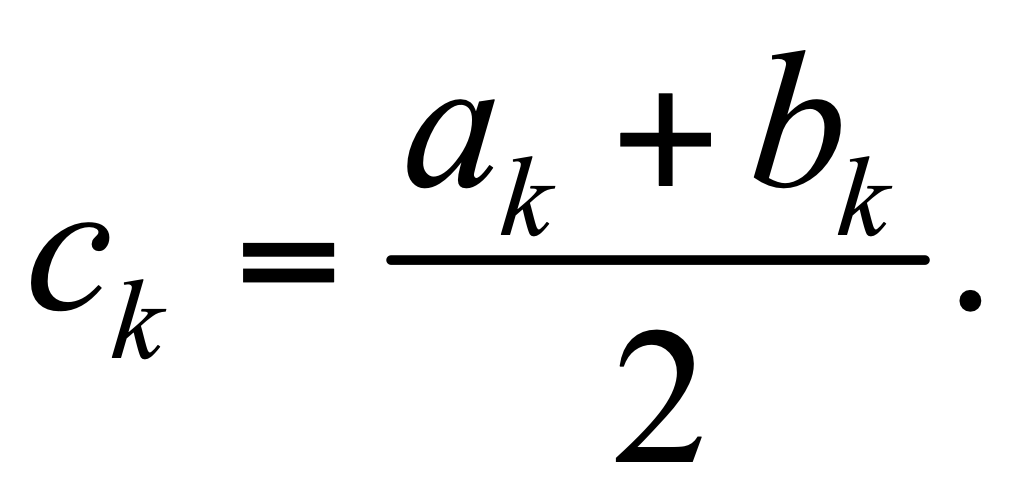
Методы дихотомии, хорд и золотого сечения являются интервальными, т.е. их смысл заключается в уменьшении исходного интервала, содержащего корень, до тех пор, пока размеры интервала не окажутся соизмеримы с требуемой погрешностью.

Для этих методов интервалом поиска корня на некоторой *k*-й итерации будет являться отрезок [*ak*, *bk*], при этом *a*0 = *a*, *b*0 = *b*. Длина интервала в интервальных методах гарантированно уменьшается на каждой итерации решения, поэтому альтернативой условию (2.1.3) будет, очевидно, условие

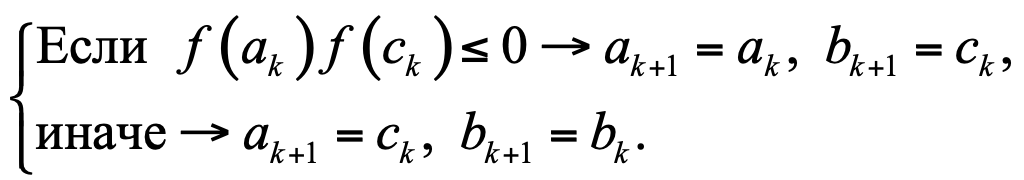
 (2.1.5)

т.к. погрешность определения корня не может превышать половины длины интервала.

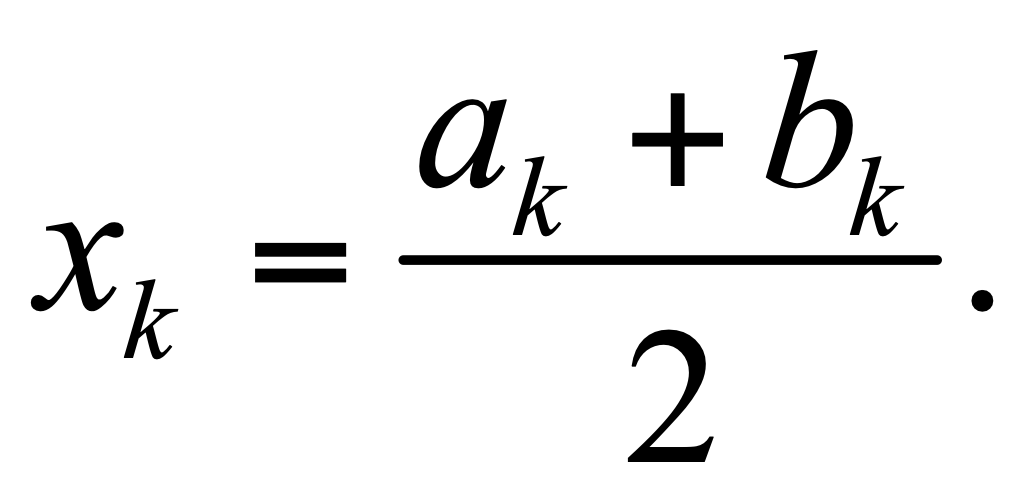
В методе дихотомии интервал разбивается следующим образом. Вычисляется точка, расположенная в середине отрезка:

 (2.1.6)

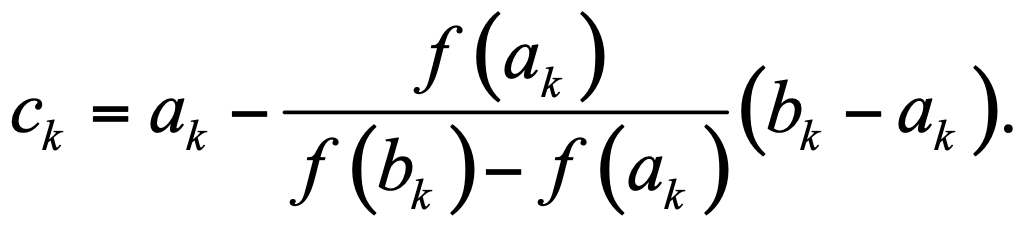
Далее, согласно (2.1.2), проверяется, какому из интервалов – [*ak*, *сk*] или [*сk*, *bk*] – принадлежит корень. Т.е.,

 (2.1.7)

В качестве *k*-го приближения корня берется точка

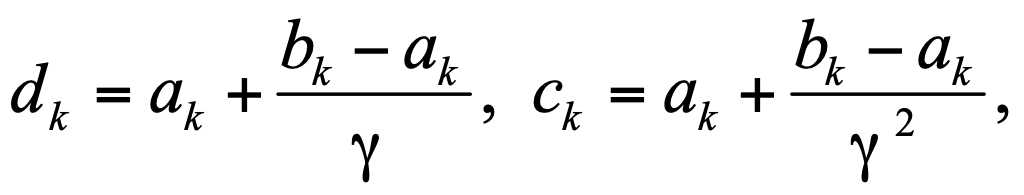
 (2.1.8)

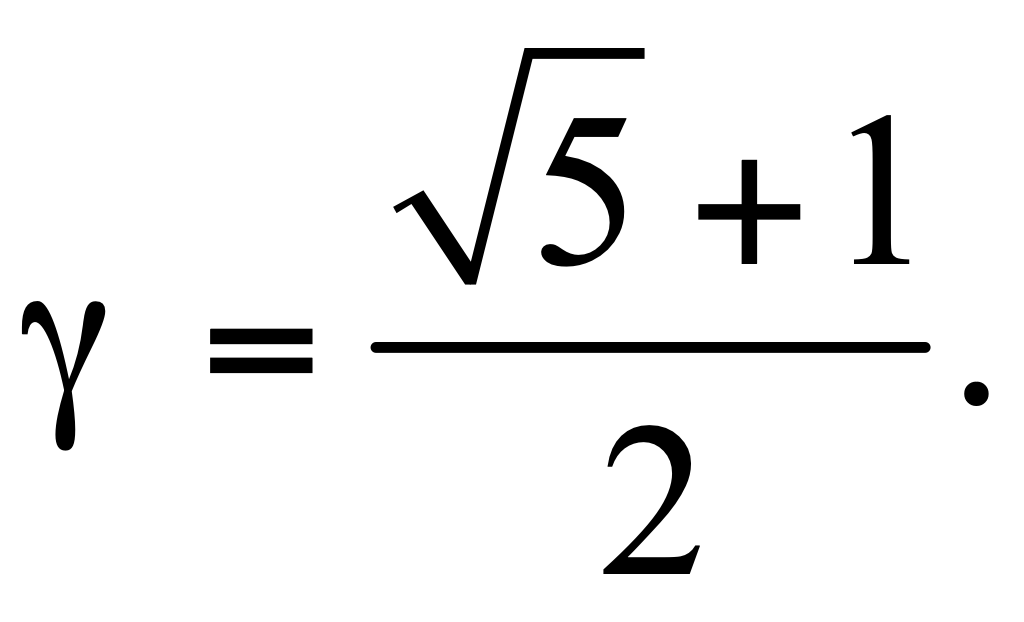
Метод хорд во всем аналогичен методу дихотомии, только интервал разбивается другой точкой:

 (2.1.9)

Выбор интервала осуществляется согласно (2.1.7), а новое приближение корня вычисляется по формуле (2.1.8).

В методе золотого сечения интервал разбивается двумя симметричными относительно границ интервала точками:

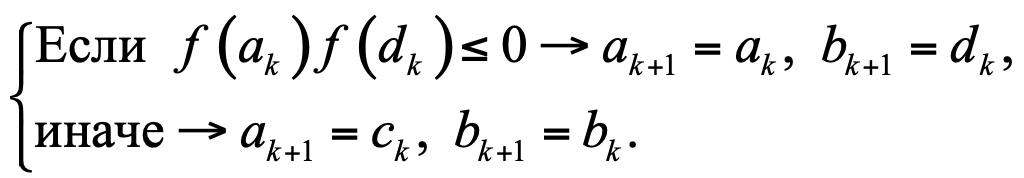
 (2.1.10)

где 

Для упрощения вычислений можно учесть упомянутую симметричность расположения точек *ck* и *dk*:

*ck* – *ak* = *bk* – *dk*. (2.1.11)

Далее, согласно (2.1.2), проверяется, какому из интервалов – [*ak*, *dk*] или [*сk*, *bk*] – принадлежит корень. Т.е.,

 (2.1.12)

Новое приближение корня вычисляется по формуле (2.1.8).

## **Итерационные методы**

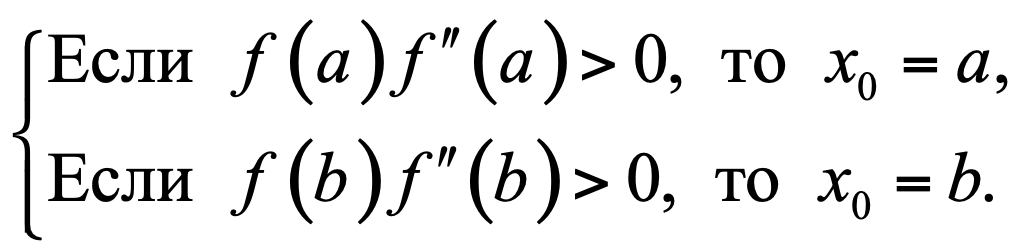
Методы Ньютона (касательных) и итераций являются итеративными (итерационными), на основе некоторого приближения корня *xk* они позволяют на каждой итерации получать новое приближение *xk*+1. При этом используется информация о первой производной функции. Вместо условия (2.1.3) в итеративных методах оценивается расстояние между последним и предпоследним приближениями корня:

|*xk*+1 – *xk*| < *ε*. (2.1.13)

При этом нужно знать начальное приближение *x*0, а дальнейшие приближения на каждой *k*+1-й итерации находятся по итеративной формуле:

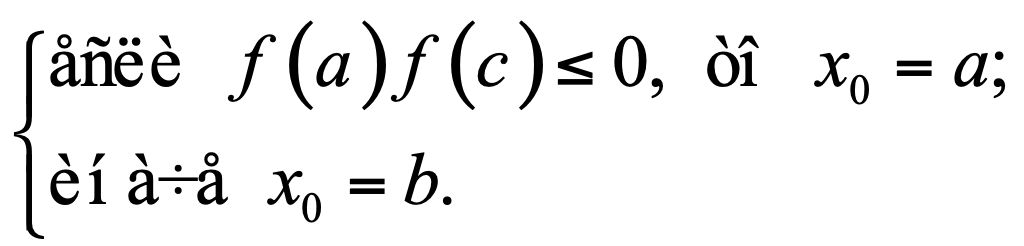
*xk*+1 = *φ*(*xk*). (2.1.14)

В методе Ньютона начальное приближение выбирается в соответствии со следующим условием: если в некоторой точке *x* произведение *f*(*x*)*f"*(*x*) > 0, то точка *x* является подходящей для начала итерационного процесса. Проверяются границы интервала:

 (2.1.15)

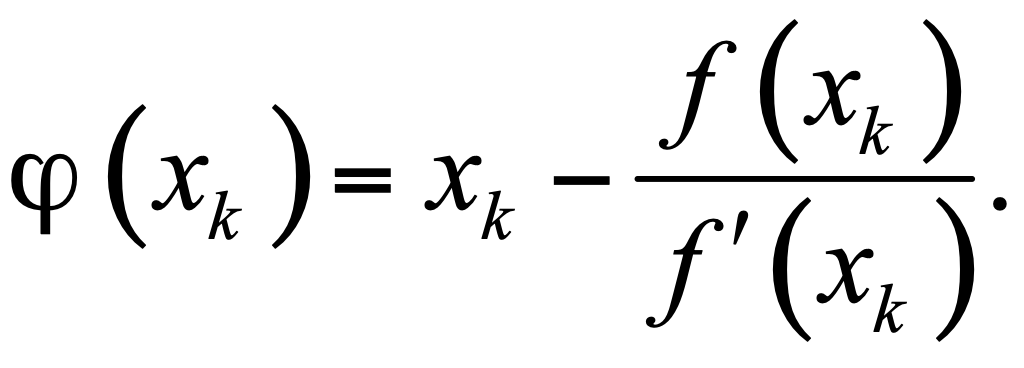
На практике может наблюдаться ситуация, когда оба условия (2.1.15) не выполняются. В этом случае вместо второго условия можно использовать оператор «иначе», либо воспользоваться вторым критерием.

Если вторая производная функции не известна, можно воспользоваться другим критерием. Вычислим точку *c* по формуле (2.1.9), и далее

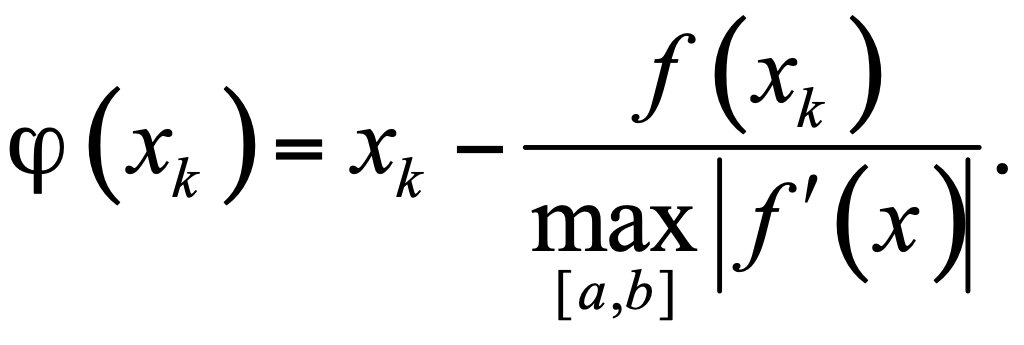
 (2.1.16)

Если начальная точка определена неправильно, то найденное решение уравнения (2.1.1) может находиться за пределами отрезка [*a*, *b*].

Функция *φ*(*xk*) в (2.1.14) для метода Ньютона выглядит следующим образом:

 (2.1.17)

В методе итераций, если выполняется неравенство |*φ'*(*x*)| < 1, процесс сходится независимо от выбора начальной точки. Поэтому можно брать любую из границ интервала, его середину и т.п. А функция *φ*(*xk*) в (2.1.14) выглядит следующим образом:

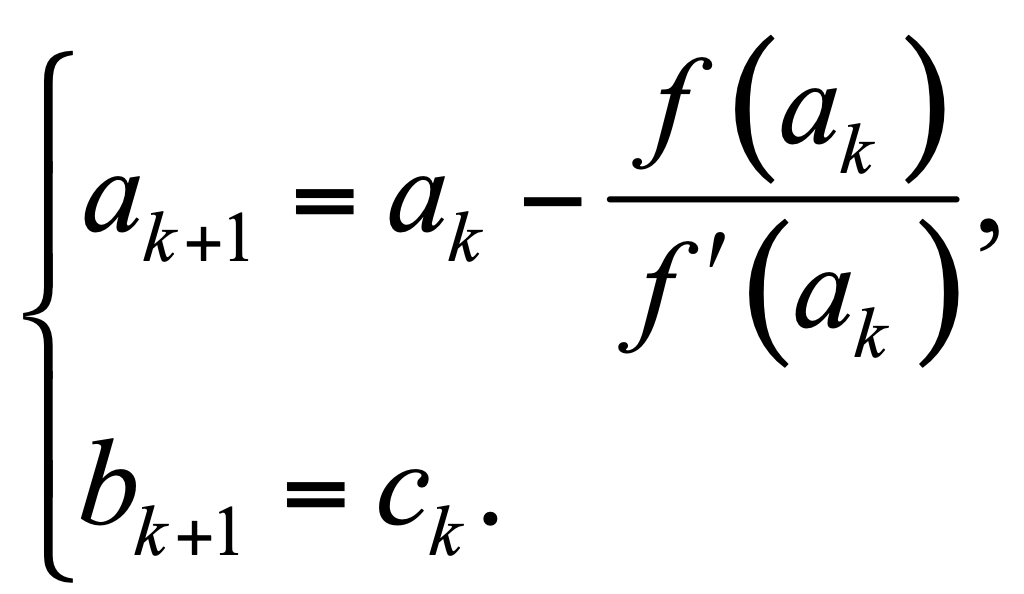
 (2.1.18)

В отличие от интервальных методов, длина исследуемого отрезка в которых на каждой итерации гарантированно уменьшается (например, для метода дихотомии – в два раза, для метода золотого сечения – в *γ* раз), в итеративных методах, в общем случае, расстояние между последовательными приближениями корня может иногда и увеличиваться. То же самое касается и значения функции в этих точках – оно может как уменьшаться, так и увеличиваться. Поэтому для некоторых функций условия (2.1.3) и (2.1.4) могут не выполняться в течение довольно большого числа итераций (или вообще никогда). В этом случае итерации следует прекращать при выполнении хотя бы одного условия.

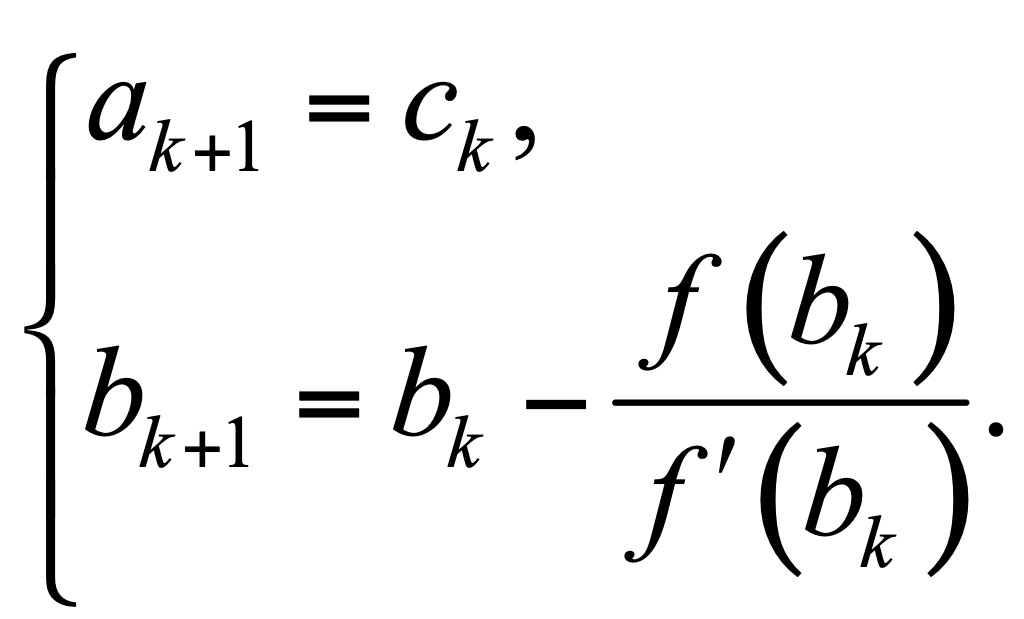
## **Комбинированный метод**

Комбинированный метод сочетает в себе сильные стороны методов хорд и Ньютона, и поэтому является достаточно эффективным для большого класса функций. Т.к. он является интервальным, то для него применимы выражения (2.1.5) и (2.1.8). Исключение интервалов выполняется по следующему алгоритму.

Сначала по формуле (2.1.9) ищется точка пересечения хорды с осью *x*. Далее, согласно (2.1.15), если *f*(*ak*)*f"*(*ak*) > 0 то точку *ak* можно переместить ближе к корню по формуле Ньютона (2.1.14) и (2.1.17). Тогда точка *bk* перемещается по формуле метода хорд (2.1.9):

 (2.1.19)

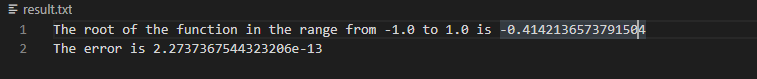
Если же *f*(*bk*)*f"*(*bk*) > 0, то, наоборот, точку *bk* можно переместить ближе к корню по формуле Ньютона, а точку *ak* – по формуле метода хорд:

 (2.1.20)

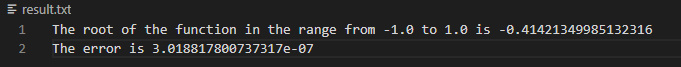
Два упомянутых условия достаточно проверять только один раз, если вторая производная не меняет своего знака на отрезке [*a*, *b*]. Но, т.к. это выполняется не для всех функций, лучше их проверять на каждой итерации. Аналогично (2.1.15), вместо второго условия можно использовать оператор «иначе», чтобы не возникла ситуация, когда оба условия не выполняются.

## **Результаты работы программы**

## Дихотомия



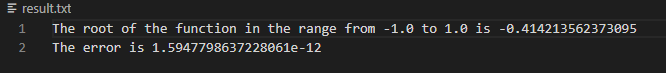
## Хорд



## Ньютон



Комбинированный



## **Код программы**

1. from math import \*
2. from sympy import \*
3. from sympy.sets import Interval
4. def create\_file(name\_file):
5. with open(name\_file,'w') as f:
6. f.write('f(x)=-(x-1)\*(x-1)+2\n')
7. f.write('-1 1\n')
8. f.write('0.000001')
9. def first\_method(func, start, finish, eps):
10. temp\_finish = finish
11. temp\_start = start
12. i = 0
13. while True:
14. i += 1
15. center = (temp\_start + temp\_finish)/2
16. result = eval(func.replace('x', str(center)))
17. if abs(result) < eps:
18. with open('result.txt','w') as f:
19. f.write(f'The root of the function in the range from {start} to {finish} is {center}\n')
20. error = abs(temp\_finish - temp\_start) /(2\*\*i)
21. f.write(f'The error is {error}')
22. break
23. temp\_result = eval(func.replace('x', str(temp\_start)))
24. if result\*temp\_result < 0:
25. temp\_finish = center
26. else:
27. temp\_start = center
28. def second\_method(func, start, finish, eps):
29. temp\_start = start
30. temp\_finish = finish
31. left\_result = result = eval(func.replace('x', str(temp\_start)))
32. right\_result = result = eval(func.replace('x', str(temp\_finish)))
33. x = temp\_start - ((left\_result\*(temp\_finish-temp\_start)) / (right\_result-left\_result))
34. result\_x = result = eval(func.replace('x', str(x)))
35. last\_x = x
36. if result\_x \* right\_result < 0:
37. result = right\_result
38. dot = temp\_finish
39. else:
40. result = left\_result
41. dot = temp\_start
42. while True:
43. result\_n = eval(func.replace('x', str(last\_x)))
44. x = last\_x - (result\_n/(result-result\_n))\*(dot-last\_x)
46. if abs(x - last\_x) < eps:
47. break
48. last\_x = x
49. error = abs(x - last\_x)
50. with open('result.txt','w') as f:
51. f.write(f'The root of the function in the range from {start} to {finish} is {x}\n')
52. f.write(f'The error is {error}')
53. def third\_method(func, start, finish, eps):
54. temp\_start = start
55. temp\_finish = finish
56. left\_result = eval(func.replace('x', str(temp\_start)))
57. x = symbols('x')
58. diff2 = diff(func, x, x)
59. left\_diff2 = eval(str(diff2).replace('x', str(temp\_start)))
60. if left\_result \* left\_diff2 > 0:
61. last\_x = temp\_start
62. else:
63. right\_diff2 = eval(str(diff2).replace('x', str(temp\_finish)))
64. right\_result = eval(func.replace('x', str(temp\_finish)))
65. if right\_diff2\*right\_result > 0:
66. last\_x = temp\_finish
67. else:
68. last\_x = temp\_start
69. print("approximation is impossible")
70. i = 0
71. while True:
72. i+=1
73. diff1 = diff(func, x)
74. result\_diff1 = eval(str(diff1).replace('x', str(last\_x)))
75. result\_last\_x = eval(func.replace('x', str(last\_x)))
76. new\_x = last\_x - result\_last\_x/result\_diff1
77. new\_result\_x = eval(func.replace('x', str(new\_x)))
78. if abs(new\_x-last\_x) < eps and abs(new\_result\_x) < eps:
79. break
80. last\_x = new\_x
82. down = maximum(diff(func,x,x),x,Interval(temp\_start,temp\_finish))/(2\*minimum(diff(func,x),x,Interval(temp\_start,temp\_finish)))
83. up = 2\*\*i - 1
84. error = (down\*\*up)\*(abs(temp\_finish - temp\_start)\*\*(2\*i))
85. with open('result.txt','w') as f:
86. f.write(f'The root of the function in the range from {start} to {finish} is {new\_x}')
87. f.write(f'The error is {error}')
88. def fourth\_method(func, start, finish, eps):
89. temp\_start = start
90. temp\_finish = finish
91. left\_result = eval(func.replace('x', str(temp\_start)))
92. right\_result = eval(func.replace('x', str(temp\_finish)))
93. new\_x= temp\_start - (left\_result/(right\_result-left\_result))\*(temp\_finish-temp\_start)
94. last\_x = new\_x
95. x = symbols('x')
96. diff1= diff(func,x)
97. diff2 = diff(func, x, x)
98. diff2\_result\_right = eval(str(diff2).replace('x', str(temp\_finish)))
99. if right\_result\*diff2\_result\_right>0:
100. dot = temp\_finish
101. else:
102. dot = temp\_start
103. while True:
104. result\_last\_x = eval(func.replace('x', str(last\_x)))
105. result\_dot = eval(func.replace('x', str(dot)))
106. result\_diff1\_dot = eval(str(diff1).replace('x', str(dot)))
107. new\_x = last\_x -result\_last\_x \*(dot -last\_x)/(result\_dot-result\_last\_x)
108. dot = dot-(result\_dot/result\_diff1\_dot)
109. if abs(new\_x-last\_x) < eps and abs(result\_dot) < eps:
110. break
111. last\_x = new\_x
112. error = abs(new\_x - last\_x)
113. with open('result.txt','w') as f:
114. f.write(f'The root of the function in the range from {start} to {finish} is {new\_x}\n')
115. f.write(f'The error is {error}')
116. create\_file(r'input.txt')
117. with open(r'input.txt') as f:
118. func = f.readline().split('=')[1]
119. start, finish = map(float, f.readline().split(' '))
120. eps = float(f.readline())
121. while True:
122. print('Select method(1,2,3,4) or exit', end=' ')
123. method = input()
124. if method == '1':
125. first\_method(func, start, finish, eps)
126. elif method == '2':
127. second\_method(func, start, finish, eps)
128. elif method == '3':
129. third\_method(func, start, finish, eps)
130. elif method == '4':
131. fourth\_method(func, start, finish, eps)
132. else:
133. print('Repeat enter')