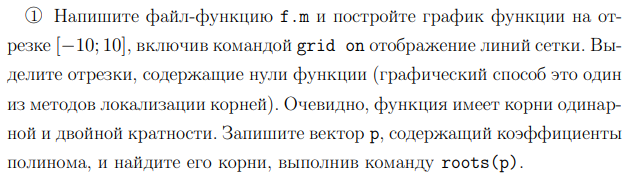
# ЛР2, В15

image.png

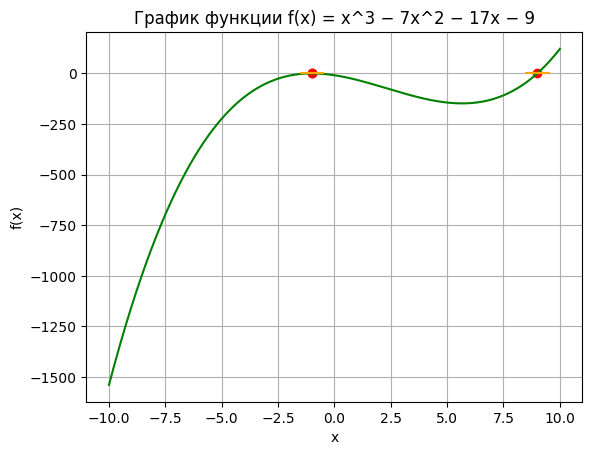
import numpy as np  
import sympy as sp  
from sympy import \*  
import matplotlib.pyplot as plt  
from scipy.optimize import root\_scalar  
import math

# 1.1

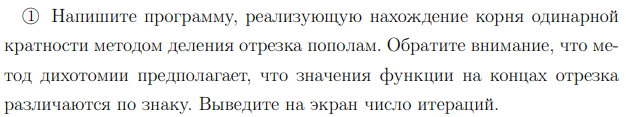


def f(x):  
 return x\*\*3 - 7\*x\*\*2 - 17\*x - 9  
  
def Plotting(roots):  
 x = np.linspace(-10, 10, 1000)  
 y = f(x)  
  
 plt.plot(x, y, color = 'green')  
 plt.xlabel('x')  
 plt.ylabel('f(x)')  
 plt.title('График функции f(x) = x^3 − 7x^2 − 17x − 9')  
  
 for root in roots:  
 print("x: ", root)  
 x1 = np.linspace(root - 0.5, root + 0.5, 1000)  
 y1 = f(x1)  
 #plt.plot(x1, y1, color = 'orange')  
 plt.plot([root - 0.5, root + 0.5],[0, 0], color = 'orange')  
 plt.scatter(root, f(root), color = 'red')  
  
 # ax = plt.gca()  
 # ax.axhline(y=0, color='k')  
 # ax.axvline(x=0, color='k')  
   
 plt.grid()  
 plt.show()  
  
p = [1, -7, -17, -9]  
roots = np.roots(p)  
Plotting(roots)

x: (8.999999999999998+0j)  
x: (-1.0000000000000004+3.624553613857577e-09j)  
x: (-1.0000000000000004-3.624553613857577e-09j)



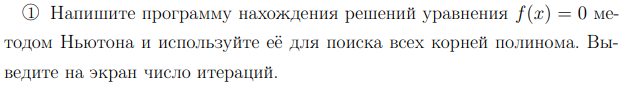
# 1.2



def f(x):  
 return x\*\*3 - 7\*x\*\*2 - 17\*x - 9  
  
def SingleRoot(a, b, eps):  
 if f(a) \* f(b) >= 0:  
 print("Метод деления отрезка пополам не применим, так как значения функции на концах отрезка одного знака.")  
 else:  
 i = 0  
 while (b - a) > eps:  
 x = (a + b) / 2  
 if f(x) == 0:  
 break  
 elif f(x) \* f(a) < 0:  
 b = x  
 else:  
 a = x  
 i += 1  
   
 print("Найденный корень: ", x)  
 print("Число итераций: ", i)  
  
  
a = -10  
b = 10  
eps = 0.000001  
  
SingleRoot(a, b, eps)

Найденный корень: 9.000000357627869  
Число итераций: 25

# 1.3



def NewtonMethod(f, x0, eps):  
 i = 0  
 a = x0  
 x0 = x0 - f.subs(x, x0) / sp.diff(f, x).subs(x, x0)  
 while abs(a - x0) > eps:  
 a = x0  
 x0 = x0 - f.subs(x, x0) / sp.diff(f, x).subs(x, x0)  
 i += 1  
  
 print("Найденный корень: ", x0)  
 print("Число итераций: ", i)  
  
  
x = sp.symbols('x')  
f = x\*\*3 - 7\*x\*\*2 - 17\*x - 9  
x0 = 5.8  
eps = 0.000001  
NewtonMethod(f, x0, eps)

Найденный корень: 9.00000000000001  
Число итераций: 10

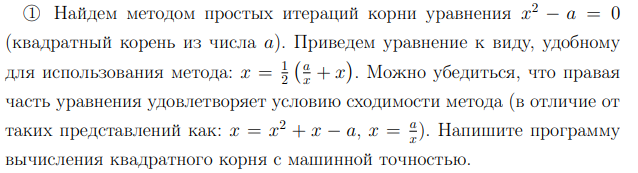
# 2.1

image.png

def NewtonMethodTwo(f, x0, eps, p):  
 i = 0  
 a = x0  
 x0 = x0 - p \* f.subs(x, x0) / sp.diff(f, x).subs(x, x0)  
 while abs(a - x0) > eps:  
 a = x0  
 x0 = x0 - p \* f.subs(x, x0) / sp.diff(f, x).subs(x, x0)  
 i += 1  
  
 print("Найденный корень: ", x0)  
 print("Число итераций: ", i)  
  
  
x = sp.symbols('x')  
f = x\*\*3 - 7\*x\*\*2 - 17\*x - 9  
x0 = 4.5  
eps = 0.000001  
NewtonMethodTwo(f, x0, eps, 1)  
NewtonMethodTwo(f, x0, eps, 2)

Найденный корень: -1.00000086187226  
Число итераций: 21  
Найденный корень: -1.00000000973578  
Число итераций: 5

# 1.4



def SimpleIterations(f, x0, eps):  
 xk = f.subs(x, x0)  
 i = 0  
 n = 10000  
 while (abs(xk - x0) > eps) and (i < n):  
 x0 = xk  
 xk = f.subs(x, xk)  
 i += 1  
 if i == n:  
 print(f"За {i} итераций не удалось найти корень")  
 else:  
 print("Найденный корень: ", xk)  
 print("Число итераций: ", i)  
   
  
x = sp.symbols('x')  
a = 2  
x0 = 1  
f1 = 1/2 \* (a/x + x)  
f2 = x\*\*2 + x - a  
f3 = a/x  
eps = 0.00001  
SimpleIterations(f1, x0, eps)

Найденный корень: 1.41421356237469  
Число итераций: 3

# 2.2

image.png

x = sp.symbols('x')  
a = sp.symbols('a')  
f = x\*\*2 + x - a  
df = sp.diff(f, x)  
inequality = np.abs(df) < 1  
solution = sp.solve\_univariate\_inequality(inequality, x)  
solution

(-1 < x) & (x < 0)

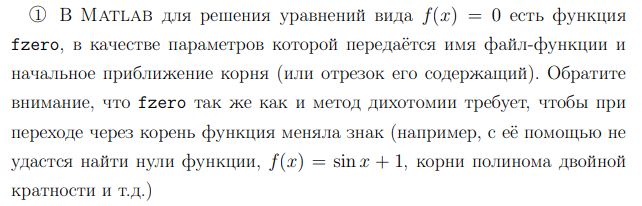
x = sp.symbols('x')  
a = 0.2  
x0 = -0.5  
f = x\*\*2 + x - a  
eps = 0.00001  
SimpleIterations(f, x0, eps)

Найденный корень: -0.447213931484371  
Число итераций: 4

x = sp.symbols('x')  
a = 0.2  
x0 = 1  
f = x\*\*2 + x - a  
eps = 0.00001  
SimpleIterations(f, x0, eps)

За 10000 итераций не удалось найти корень

# 1.5



def f(x):  
 return x\*\*2 - 2  
  
sol = root\_scalar(f, bracket=[0, 2])  
  
print("f(x) = x\*\*2 - 2:")  
if sol.converged:  
 print("\tНайденный корень:", sol.root)  
 print("\tЧисло итераций:", sol.iterations)  
else:  
 print("\tМетод не сошелся")  
  
  
def g(x):  
 return np.sin(x) + 1  
  
solg = root\_scalar(g, bracket=[-2, 0])  
  
print("g(x) = sin(x) + 1:")  
if solg.converged:  
 print("\tНайденный корень:", solg.root)  
 print("\tЧисло итераций:", solg.iterations)  
else:  
 print("\tМетод не сошелся")

f(x) = x\*\*2 - 2:  
 Найденный корень: 1.4142135623731364  
 Число итераций: 8

---------------------------------------------------------------------------  
ValueError Traceback (most recent call last)  
Cell In[61], line 17  
 14 def g(x):  
 15 return np.sin(x) + 1  
---> 17 solg = root\_scalar(g, bracket=[-2, 0])  
 19 print("g(x) = sin(x) + 1:")  
 20 if solg.converged:  
  
File /lib/python3.11/site-packages/scipy/optimize/\_root\_scalar.py:279, in root\_scalar(f, args, method, bracket, fprime, fprime2, x0, x1, xtol, rtol, maxiter, options)  
 277 a, b = bracket[:2]  
 278 try:  
--> 279 r, sol = methodc(f, a, b, args=args, \*\*kwargs)  
 280 except ValueError as e:  
 281 # gh-17622 fixed some bugs in low-level solvers by raising an error  
 282 # (rather than returning incorrect results) when the callable  
 283 # returns a NaN. It did so by wrapping the callable rather than  
 284 # modifying compiled code, so the iteration count is not available.  
 285 if hasattr(e, "\_x"):  
  
File /lib/python3.11/site-packages/scipy/optimize/\_zeros\_py.py:799, in brentq(f, a, b, args, xtol, rtol, maxiter, full\_output, disp)  
 797 raise ValueError(f"rtol too small ({rtol:g} < {\_rtol:g})")  
 798 f = \_wrap\_nan\_raise(f)  
--> 799 r = \_zeros.\_brentq(f, a, b, xtol, rtol, maxiter, args, full\_output, disp)  
 800 return results\_c(full\_output, r)  
  
ValueError: f(a) and f(b) must have different signs

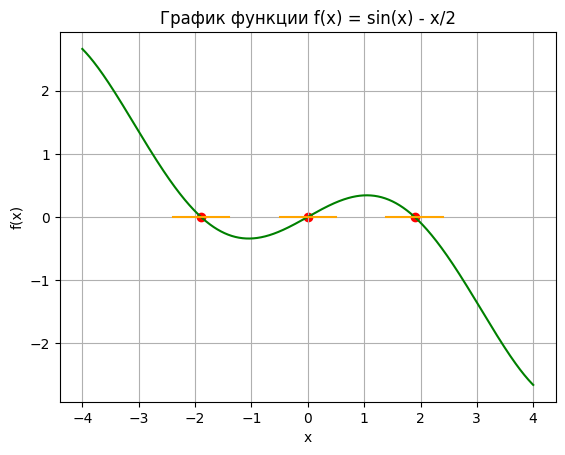
# 3.1

image.png

image.png

def f(x):  
 return x\*\*9/362880 -x\*\*7/5040 + x\*\*5/120 - x\*\*3/6 + x/2  
  
def Plotting(roots):  
 x = np.linspace(-4, 4, 1000)  
 y = f(x)  
  
 plt.plot(x, y, color = 'green')  
 plt.xlabel('x')  
 plt.ylabel('f(x)')  
 plt.title('График функции f(x) = sin(x) - x/2')  
  
 for root in roots:  
 if -4 < root < 4:  
 print("x: ", root)  
 x1 = np.linspace(root - 0.5, root + 0.5, 1000)  
 y1 = f(x1)  
 #plt.plot(x1, y1, color = 'orange')  
 plt.plot([root - 0.5, root + 0.5],[0, 0], color = 'orange')  
 plt.scatter(root, f(root), color = 'red')  
  
 # ax = plt.gca()  
 # ax.axhline(y=0, color='k')  
 # ax.axvline(x=0, color='k')  
   
 plt.grid()  
 plt.show()  
  
p = [1/362880, 0, -1/5040, 0, 1/120, 0, -1/6, 0, 1/2, 0]  
roots = np.roots(p)  
Plotting(roots)

x: (-1.8955282003070628+0j)  
x: (1.8955282003070641+0j)  
x: 0j



def f(x):  
 return np.sin(x) - x/2  
   
a = 1  
b = 4  
eps = 0.00001  
  
SingleRoot(a, b, eps)

Найденный корень: 1.8954944610595703  
Число итераций: 19

x = sp.symbols('x')  
f = sp.sin(x) - x/2  
x0 = 1.5  
eps = 0.000001  
NewtonMethod(f, x0, eps)

Найденный корень: 1.89549426703398  
Число итераций: 4

x = sp.symbols('x')  
x0 = 1.8  
f = sp.sin(x) + x/2  
eps = 0.000001  
SimpleIterations(f, x0, eps)

Найденный корень: 1.89549411963415  
Число итераций: 7

x = sp.symbols('x')  
x0 = 1.8  
f = 2 \* sp.sin(x)  
eps = 0.000001  
SimpleIterations(f, x0, eps)

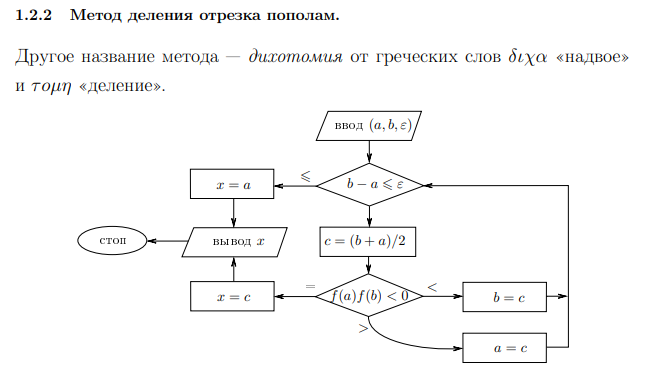
Найденный корень: 1.89549397541213  
Число итераций: 27

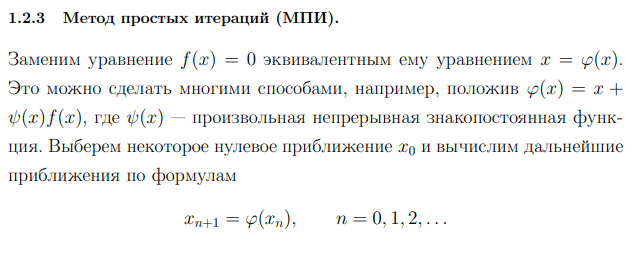
## Контрольные вопросы

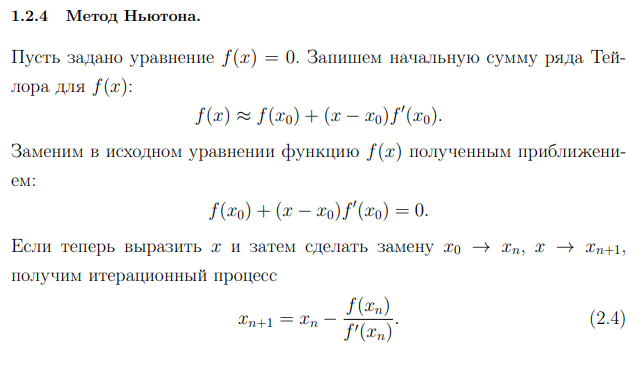
### 1

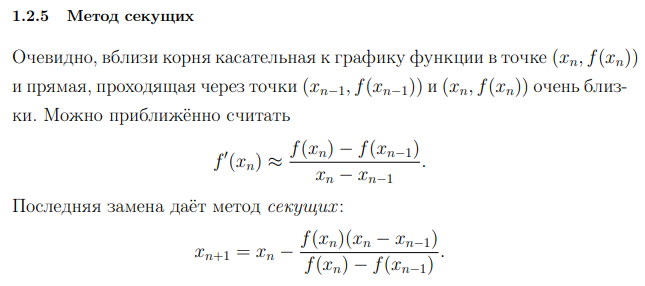
image.png

### Ответ:

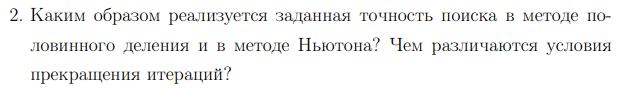








### 2

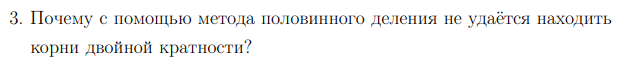


### Ответ:

В методе половинного деления заданная точность достигается путем сравнения длины текущего интервала с требуемой точностью. Итерации прекращаются, когда длина интервала становится меньше заданной точности.

В методе Ньютона заданная точность обычно связана с разницей между двумя последовательными приближениями корня. Итерации могут завершиться, когда разница между двумя последовательными приближениями становится меньше заданной точности.

### 3

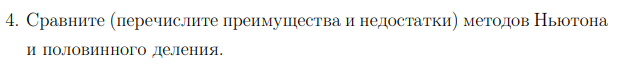


### Ответ:

Текст задания:"Обратите внимание, что метод дихотомии предполагает, что значения функции на концах отрезка различаются по знаку

На концах отрезка, на котором находится корень двойной кратности, значения функции НЕ различаются по знаку."

### 4



### Ответ:

Метод половинного деления

+:

1. Простота реализации
2. Гарантированное нахождение корня

-:

1. Медленная сходимость, особенно при большом интервале
2. Не подходит для корней двойной (чётной) кратности

Метод Ньютона:

+:

1. Быстрая сходимость, особенно при достаточно близком начальном приближении к корню
2. Модифицированный метод находит кратные корни

-:

1. Из-за плохого начального приближения метод зацикливается
2. Требуется вычисление производной функции на каждой итерации

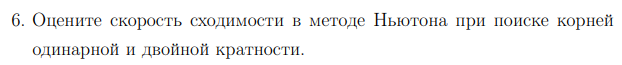
### 5

image.png

### Ответ:

1. Непрерывность и дифференцируемость функции в окрестности корня
2. Производная не должна обращаться в 0 в окрестности корня
3. Хорошее начальное приближение

### 6



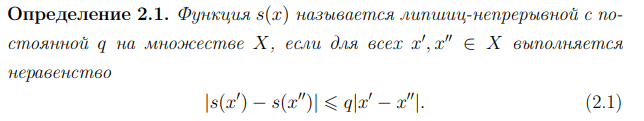
### Ответ:

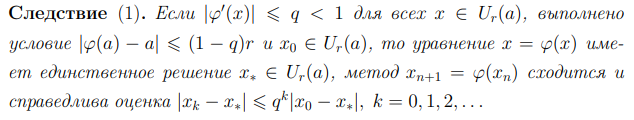
Для корней одинарной кратности скорость сходимости квадратичная. Для корней двойной кратности скорость сходимости линейная. Но у модифицированного метода для кратных корней скорость сходимости квадратичная.

### 7

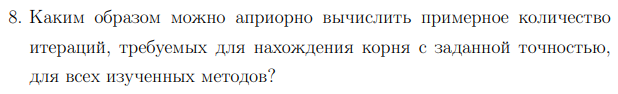
image.png

### Ответ:



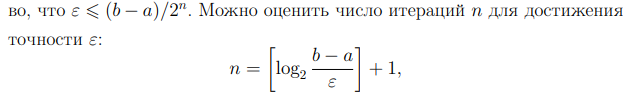


### 8



### Ответ:

Метод деления отрезка пополам:



Метод Ньютона:

image.png

image.png

x = sp.symbols('x')  
a = 1  
  
fun = x\*\*2 + x - a  
df = sp.diff(fun, x)  
  
df2 = lambda x1: 2\*x1 + 1  
  
a01 = sp.solve(2\*x + 2, x)[0]  
a02 = sp.solve(2\*x + 0, x)[0]  
  
x\_vals = np.linspace(-4, 4, 100)  
plt.plot(x\_vals, df2(x\_vals), color = 'blue')  
plt.axhline(0, color='black')  
plt.axvline(0, color='black')  
plt.axhline(1, color='red', linestyle='--')  
plt.axhline(-1, color='red', linestyle='--')  
plt.plot(a01, df2(a01), 'g\*')  
plt.plot(a02, df2(a02), 'g\*')  
plt.xlabel('x')  
plt.ylabel('df(x)')  
plt.grid()  
plt.show()

def Sinus(n):  
 x = sp.symbols('x')  
 c = -x/2  
 for i in range(0, n):  
 s = ((-1)\*\*i\*x\*\*(2\*i+1))/(math.factorial(2\*i+1))  
 c += s  
 return c  
print(Sinus(2))

def f(x):  
 return np.sin(x) - x/2  
  
def Plotting():  
 x = np.linspace(-4, 4, 1000)  
 y = f(x)  
  
  
   
 plt.plot(x, y, color = 'green')  
 plt.xlabel('x')  
 plt.ylabel('f(x)')  
  
   
 plt.grid()  
 plt.show()  
  
Plotting()

import numpy as np  
from scipy.optimize import fsolve, root\_scalar  
import matplotlib.pyplot as plt  
  
# Определяем функцию уравнения  
def equation(x):  
 return np.sin(x) - x/2  
  
# Используем fsolve для нахождения корней  
roots\_fsolve = fsolve(equation, [0, 3, 6])  
  
# Используем root\_scalar для нахождения корней  
root\_1 = root\_scalar(equation, bracket=[-2, -1], method='brentq')  
root\_2 = root\_scalar(equation, bracket=[-1, 1], method='brentq')  
root\_3 = root\_scalar(equation, bracket=[1, 2], method='brentq')  
  
print("Корни, найденные с использованием fsolve:", roots\_fsolve)  
print("Корень 1, найденный с использованием root\_scalar:", root\_1.root)  
print("Корень 2, найденный с использованием root\_scalar:", root\_2.root)  
print("Корень 3, найденный с использованием root\_scalar:", root\_3.root)  
  
# Строим график функций для визуализации  
x\_vals = np.linspace(-2\*np.pi, 2\*np.pi, 1000)  
y1 = np.sin(x\_vals)  
y2 = x\_vals / 2  
  
plt.figure()  
plt.plot(x\_vals, y1, label='sin(x)')  
plt.plot(x\_vals, y2, label='x/2')  
plt.plot(x\_vals, y1 + y2, label='sin(x) + x/2')  
plt.axhline(0, color='black', linestyle='--')  
plt.legend()  
plt.grid(True)  
plt.show()

import sympy as sp  
  
def FixedPointIteration(g, x0, epsilon, max\_iter=10000):  
 x = sp.symbols('x')  
 n = 0  
 while n < max\_iter:  
 x1 = g.subs(x, x0)  
 if abs(x1 - x0) < epsilon:  
 return x1, n  
 x0 = x1  
 n += 1  
 return None, max\_iter  
  
# Определяем символьную переменную x и параметр a  
x, a = sp.symbols('x a')  
  
# Определяем итерационную функцию g(x) = a - x\*\*2  
g = x\*\*2 + x - a  
  
# Задаем начальное приближение и точность  
x0 = 1 # Начальное приближение  
epsilon = 1e-6 # Точность  
  
# Исследуем область сходимости представления x = x^2 + x - a  
a\_values = [0.5] # Значения параметра a для проверки  
  
for a\_val in a\_values:  
 root, num\_iterations = FixedPointIteration(g.subs(a, a\_val), x0, epsilon)  
 print(f"Для a = {a\_val}:")  
 if root is not None:  
 print(f"Найденный корень: {root}")  
 print(f"Число итераций: {num\_iterations}")  
 else:  
 print("Метод не сошелся за указанное количество итераций.")

p = np.roots([1, -7, -17, -9])  
r = 1 + 1 / max(abs(p))  
flag = 1  
  
for i in range(len(p)):  
 if (np.imag(p[i]) != 0) and (abs(p[i]) > r):  
 flag = 0  
  
if flag:  
 print('true')  
print(r)

Условия сходимости метода простых итераций:

1. Сжимающее отображение: Для сходимости метода простых итераций необходимо, чтобы преобразование ( g(x) ) было сжимающим на заданном интервале. Это означает, что для всех ( x ) из интервала должно выполняться условие: [ |g'(x)| \leq k < 1, ] где ( k ) - константа, ( 0 < k < 1 ), и ( g'(x) ) - производная функции ( g(x) ).
2. Сходимость начального приближения: Начальное приближение должно быть выбрано достаточно близко к корню уравнения для обеспечения сходимости метода.
3. Непрерывность функции ( g(x) ): Функция ( g(x) ) должна быть непрерывной на интервале поиска корня.
4. Условие Липшица: Для обеспечения сходимости метода простых итераций также требуется выполнение условия Липшица. Это означает, что существует константа ( L ), такая что для всех ( x ) и ( y ) из интервала выполняется: [ |g(x) - g(y)| \leq L|x - y|. ]

При соблюдении указанных условий метод простых итераций может сходиться к корню уравнения. Однако важно помнить, что успешная сходимость также зависит от выбора функции ( g(x) ) и начального приближения.