Университет ИТМО

**Факультет: ПИиКТ**

**Дисциплина: Вычислительная математика**

**Лабораторная работа №2**

Вариант “2вг”

Выполнили: **Кудлаков Роман**

Группа: **P3231**

Преподаватель: **Перл Ольга Вячеславовна**

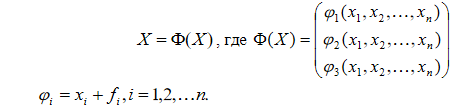
Санкт-Петербург, 2021

**Метод** **простой** **итерации**, называемый также **методом** последовательного приближения, – это математический алгоритм нахождения значения неизвестной величины путем постепенного ее уточнения.

На первом шаге сами определяем интервал, на котором будем искать решения.

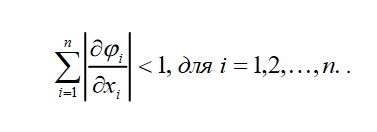
Далее задаем точность вычислений.

После приводим систему к нормализованному виду



И выбираем начальное приближение, где значение каждой неизвестной лежит в своем интервале.

Теперь, чтобы воспользоваться методом нужно проверить выполняется ли условие сходимости на выбранном интервале:

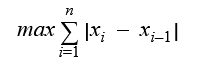


Теперь подставим в правую часть каждого уравнения значение из начального приближения и получим новые значения неизвестных. Дальше будем подставлять уже новые, только что полученные значения, в уравнения.

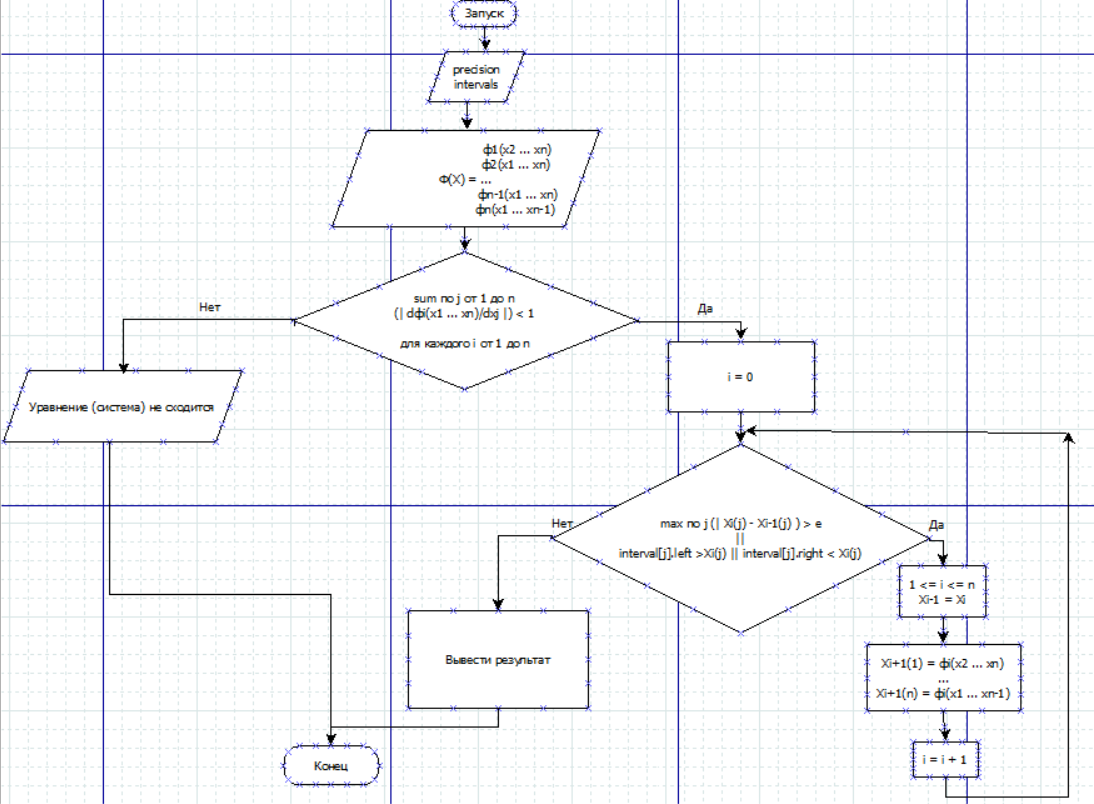


Так будем продолжать делать до тех пор, пока не получим точность меньше, чем данная.

Расчет точности на i-том шаге:



**Блок-схема алгоритма**



**Метод касательных**

Метод предназначен для приближенного нахождения нулей функции. Суть метода касательных состоит в разбиении отрезка [a; b] (при условии f(a) \* f(b) < 0) на два отрезка с помощью касательной и выборе нового отрезка от точки пересечения касательной с осью абсцисс до неподвижной точки, на котором функция меняет знак и содержит решение, причём подвижная точка приближается к ε-окрестности решения.

Метод касательных применим для решения уравнения вида **f(x) = 0** на отрезке **[a; b]**

Условие неподвижной точки для метода касательных **f(x) \* f’’(x) < 0**.

Условие начальной точки для метода касательных **f(x) \* f’’(x) > 0**.

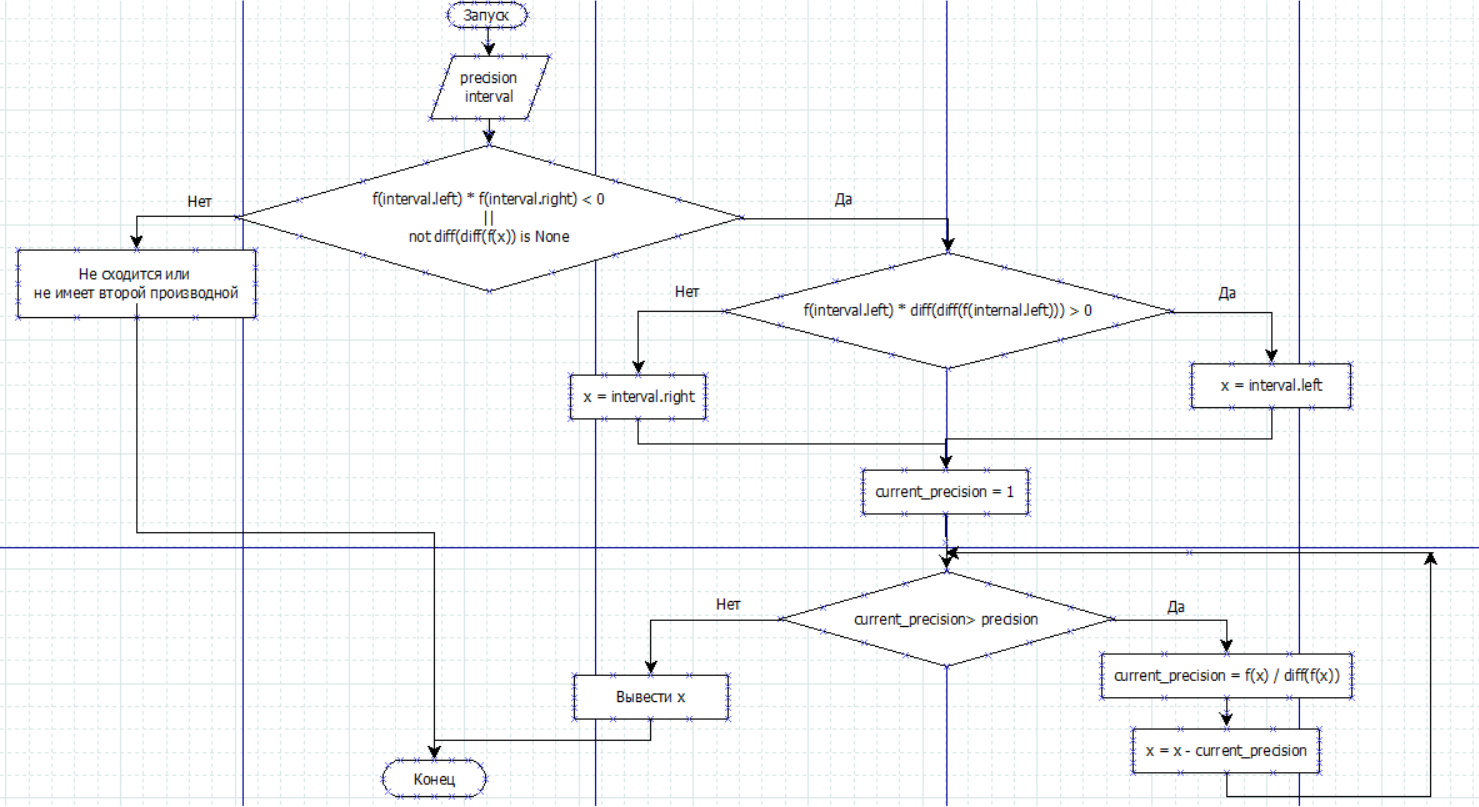
Сначала выбираем отрезок **[a; b]** такой, что функция **f(x)** дважды непрерывно дифференцируема и меняет знак на отрезке, то есть **f(a) \* f(b) < 0**

1. Находим начальную точку, если **f(a) \* f’’(a) > 0**, то **x = a**, иначе если **f(b) \* f’’(b) > 0**, то **x = b**

2. Дальше воспользуемся формулой нахождения следующей точки x

Если , тогда делаем шаг два заново, иначе выводим результат.

**Блок-схема алгоритма**



**Листинг кода**

**def** **express\_unknowns**(function, variable):  
 return sp.solve(sp.parse\_expr(function), variable)  
  
  
**def** **make\_expressed\_functions**(funcs, name\_vars):  
 ex\_functions = []  
 for i in range(len(funcs)):  
 ex\_functions.append(express\_unknowns(funcs[i], name\_vars[i])[0])  
 return ex\_functions  
  
  
**def** **take\_derivative**(function, variable):  
 return sp.diff(function, variable)  
  
  
**def** **find\_derivatives**(funcs, name\_vars):  
 diffs = []  
 for i in range(len(funcs)):  
 row\_of\_diffs = []  
 for j in range(len(funcs)):  
 row\_of\_diffs.append(take\_derivative(funcs[i], name\_vars[j]))  
 diffs.append(row\_of\_diffs)  
 return diffs  
  
  
**def** **find\_critical\_points**(diffs, name\_vars):  
 second\_diffs = []  
 for i in range(len(diffs)):  
 row\_of\_diffs = []  
 for j in range(len(diffs[i])):  
 row\_of\_diffs.append(take\_derivative(diffs[i][j], name\_vars[j]))  
 second\_diffs.append(row\_of\_diffs)  
  
 crit\_points = []  
 for i in range(len(second\_diffs)):  
 row\_of\_crit\_points = []  
 for j in range(len(second\_diffs[i])):  
 row\_of\_crit\_points.append(sp.solve(second\_diffs[i][j], name\_vars[j]))  
 crit\_points.append(row\_of\_crit\_points)  
 return crit\_points  
  
  
**def** **find\_max\_abs\_on\_the\_interval**(crit\_points\_func, function, variable, interval):  
 maximum = 0  
 for crit\_point in crit\_points\_func:  
 if interval[0] <= crit\_point <= interval[1]:  
 maximum = max(maximum, abs(function.subs(variable, crit\_point)))  
  
 left\_border = abs(function.subs(variable, interval[0]))  
 right\_border = abs(function.subs(variable, interval[1]))  
 maximum\_between\_borders = max(left\_border, right\_border)  
 return max(maximum, maximum\_between\_borders)  
  
  
**def** **is\_converging**(crit\_points, diffs, name\_vars, intervals):  
 for i in range(len(crit\_points)):  
 sum = 0  
 for j in range(len(crit\_points[i])):  
 sum += find\_max\_abs\_on\_the\_interval(crit\_points[i][j], derivatives[i][j], variables[j], intervals[j])  
 if sum >= 1:  
 return False  
 return True  
  
  
**def** **do\_method\_of\_simple\_iterations**(ex\_functions, intervals, precision, name\_vars):  
 result = {'unknowns': [], 'precisions': [], 'iterations': 0}  
 current\_step = [0.0, 0.0]  
 previous\_step = [np.mean(intervals[0]), np.mean(intervals[1])]  
 precisions = [0.0, 0.0]  
 current\_precision = 1  
 iteration = 0  
 while current\_precision >= pow(0.1, precision):  
 for i in range(len(previous\_step)):  
 if intervals[i][0] > previous\_step[i] or intervals[i][1] < previous\_step[i]:  
 return result  
  
 for i in range(len(ex\_functions)):  
 current\_step[i] = (ex\_functions[i].subs([(name\_vars[0], previous\_step[0]), (name\_vars[1], previous\_step[1])])).evalf()  
  
 for i in range(len(current\_step)):  
 precisions[i] = abs(current\_step[i] - previous\_step[i])  
  
 current\_precision = max(precisions)  
  
 for i in range(len(current\_step)):  
 previous\_step[i] = current\_step[i]  
 iteration += 1  
  
 result['unknowns'] = previous\_step  
 result['precisions'] = precisions  
 result['iterations'] = iteration  
 return result

**def is\_converge\_by\_curve**(left\_limit, right\_limit, func):

return func.subs('x', left\_limit).evalf() \* func.subs('x', right\_limit).evalf() < 0

**def is\_start\_point**(x, func):

return func.subs('x', x).evalf() \* take\_derivative(take\_derivative(func, 'x'), 'x').subs('x', x).evalf() > 0

**def do\_iterations**(current\_point, func, precision):

coef = 1

iteration = 0

while abs(coef) > 0.1 \*\* precision:

coef = func.subs('x', current\_point).evalf() / take\_derivative(func, 'x').subs('x', current\_point).evalf()

current\_point -= coef

iteration += 1

accuracy = coef

return {'iterations': iteration, 'result': current\_point, 'accuracy': accuracy}

**def do\_method\_of\_the\_curve**(left\_limit, right\_limit, func, precision):

if not is\_converge\_by\_curve(left\_limit, right\_limit, func):

return {'error': "Doesn't converge by method of the curve"}

start\_point = oo

if is\_start\_point(left\_limit, func):

start\_point = left\_limit

if start\_point == oo and is\_start\_point(right\_limit, func):

start\_point = right\_limit

if start\_point != oo:

return do\_iterations(start\_point, func, precision)

else:

return {'error': "No start point"}

**def find\_critical\_points**(func, name\_var):

diff = take\_derivative(func, name\_var)

crit\_points = [x for x in sp.solve(diff, name\_var) if x.is\_real]

return crit\_points

**def count\_coefficient**(left\_limit, right\_limit, func):

diff = take\_derivative(func, 'x')

crit\_points = find\_critical\_points(diff, 'x')

max\_value = max(abs(diff.subs('x', left\_limit).evalf()), abs(diff.subs('x', right\_limit).evalf()))

for crit\_point in crit\_points:

if left\_limit <= crit\_point <= right\_limit:

max\_value = max(max\_value, abs(func.subs('x', crit\_point).evalf()))

return max\_value

**def is\_converge\_by\_simple\_iterations**(left\_limit, right\_limit, func):

max\_value = count\_coefficient(left\_limit, right\_limit, func)

if max\_value >= 1:

return False

mid = (right\_limit + left\_limit) / 2

return abs(func.subs('x', mid).evalf() - mid) < (1 - max\_value) \* (mid - left\_limit)

**def do\_iterations\_method\_of\_simple\_iterations**(current\_point, func, precision):

prev\_point = current\_point

current\_precision = 1

iteration = 0

while current\_precision > 0.1 \*\* precision:

current\_point = func.subs('x', prev\_point).evalf()

current\_precision = abs(current\_point - prev\_point)

prev\_point = current\_point

iteration += 1

return {'iterations': iteration, 'result': current\_point, 'accuracy': current\_precision}

**def do\_method\_of\_simple\_iterations\_equations**(left\_limit, right\_limit, func, precision):

if not is\_converge\_by\_simple\_iterations(left\_limit, right\_limit, func):

return {'error': "Doesn't converge by method of simple iterations"}

answer = do\_iterations\_method\_of\_simple\_iterations((left\_limit + right\_limit) / 2, func, precision)

coef = count\_coefficient(left\_limit, right\_limit, func)

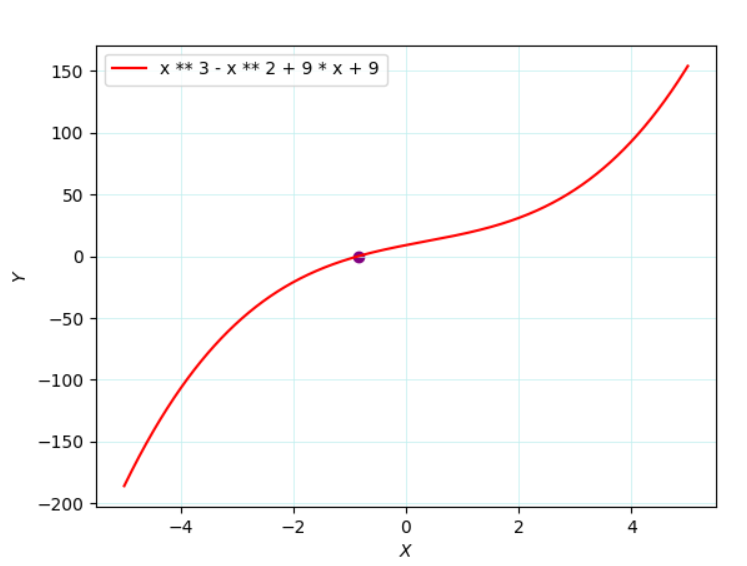
answer['coef'] = coef

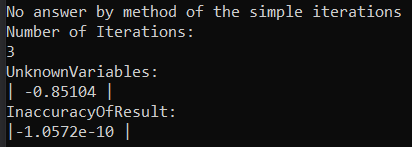
return answer

**Тесты и результаты**

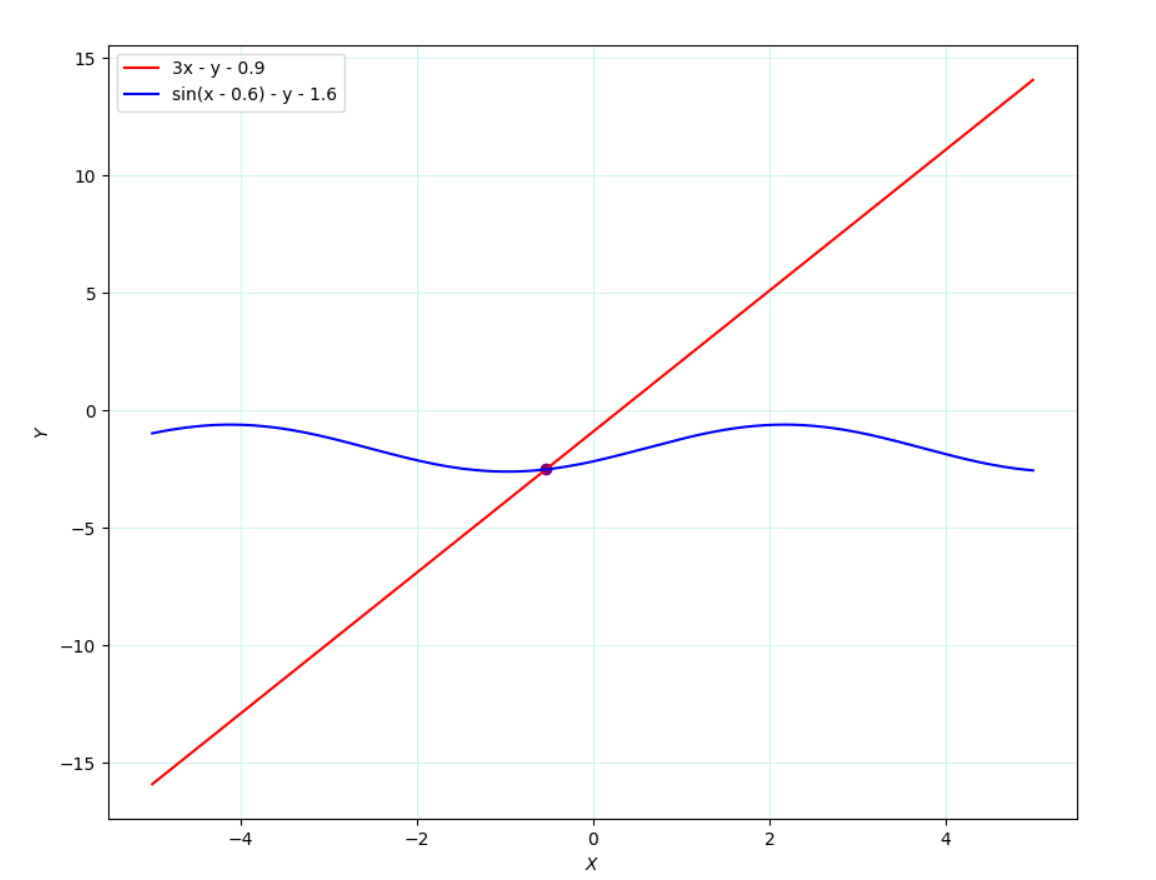
**Тест 1.**

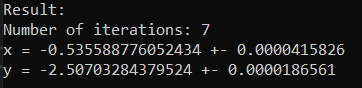
x^3 - x^2 + 9x + 9 = 0





**Тест 2.**





**Вывод.**

Метод касательных имеет высокую скорость сходимости, так как имеет квадратичную сходимость, однако, чтобы воспользоваться этим методом функция должна обязательно:

1. Иметь вторую производную

2. Первая производная не равна 0

3. Знакопостоянство первой и второй производных

Поэтому желательно использовать данный метод с другими методами и выбирать как можно меньший интервал поиска решения.

Метод простой итерации может использоваться не на очень большом количестве уравнений, так как нужно, чтобы выполнялся признак сходимости и решение было единственно на отрезке.