Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Методы численного анализа

**ОТЧЁТ**

к лабораторной работе

на тему

Решение краевых задач. Метод коллокаций, наименьших квадратов и Галеркина, стрельбы и разности аппроксимаций

Выполнил: студент группы 153503

Киселёва Елизавета Андреевна

Проверил: Анисимов Владимир Яковлевич

Минск 2023

**Оглавление**

[Цели работы: 3](file:///C:\Users\Acer\Downloads\Telegram%20Desktop\МЧА%2011.docx#_Toc126867856)

[Краткие теоретические сведения 4](file:///C:\Users\Acer\Downloads\Telegram%20Desktop\МЧА%2011.docx#_Toc126867857)

[Задание 8](file:///C:\Users\Acer\Downloads\Telegram%20Desktop\МЧА%2011.docx#_Toc126867858)

[Алгоритм задания 9](file:///C:\Users\Acer\Downloads\Telegram%20Desktop\МЧА%2011.docx#_Toc126867859)

[Программная реализация 10](file:///C:\Users\Acer\Downloads\Telegram%20Desktop\МЧА%2011.docx#_Toc126867860)

[Результат выполнения программы 18](file:///C:\Users\Acer\Downloads\Telegram%20Desktop\МЧА%2011.docx#_Toc126867861)

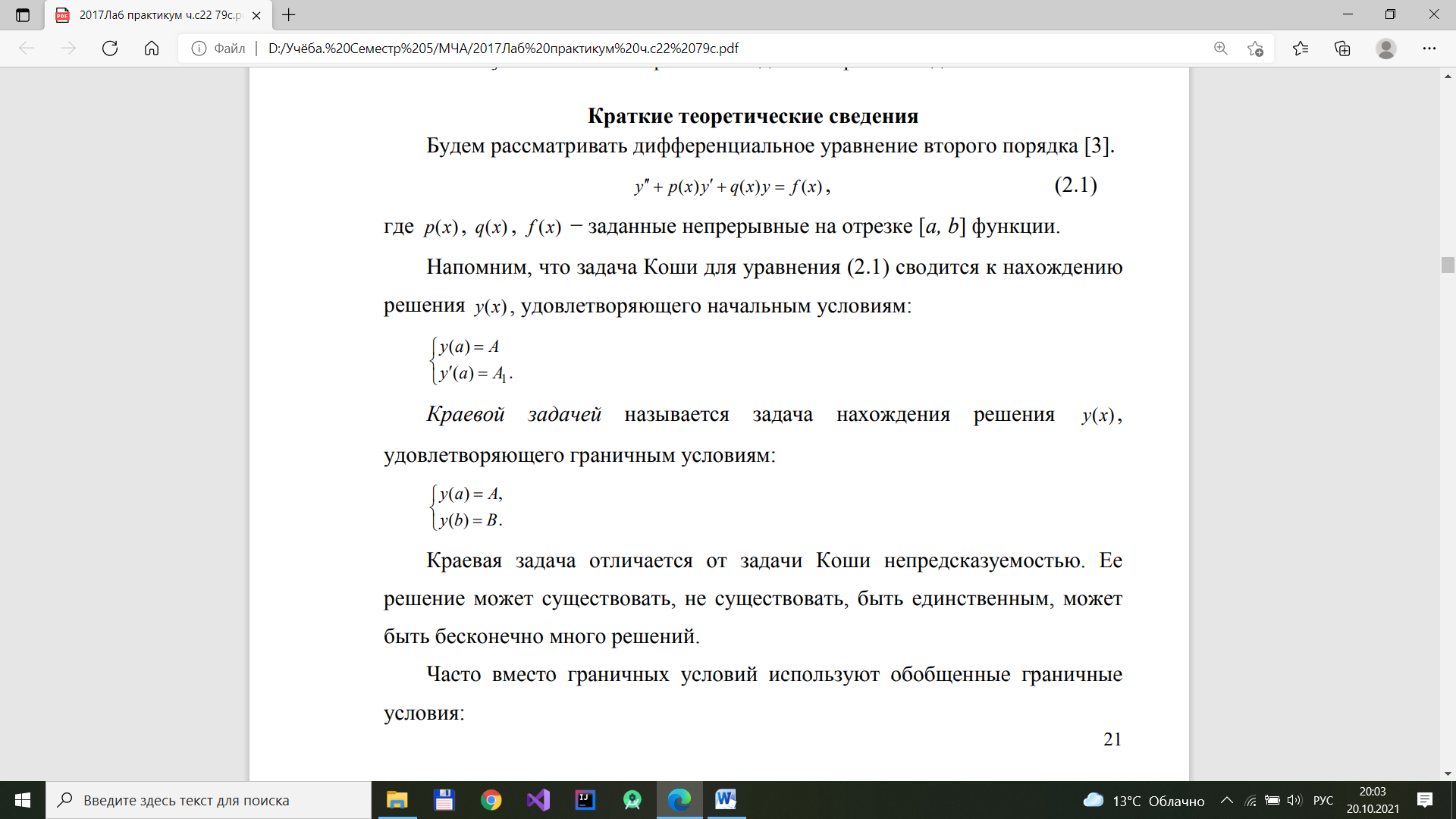
[Тестовые примеры 15](file:///C:\Users\Acer\Downloads\Telegram%20Desktop\МЧА%2011.docx#_Toc126867862)

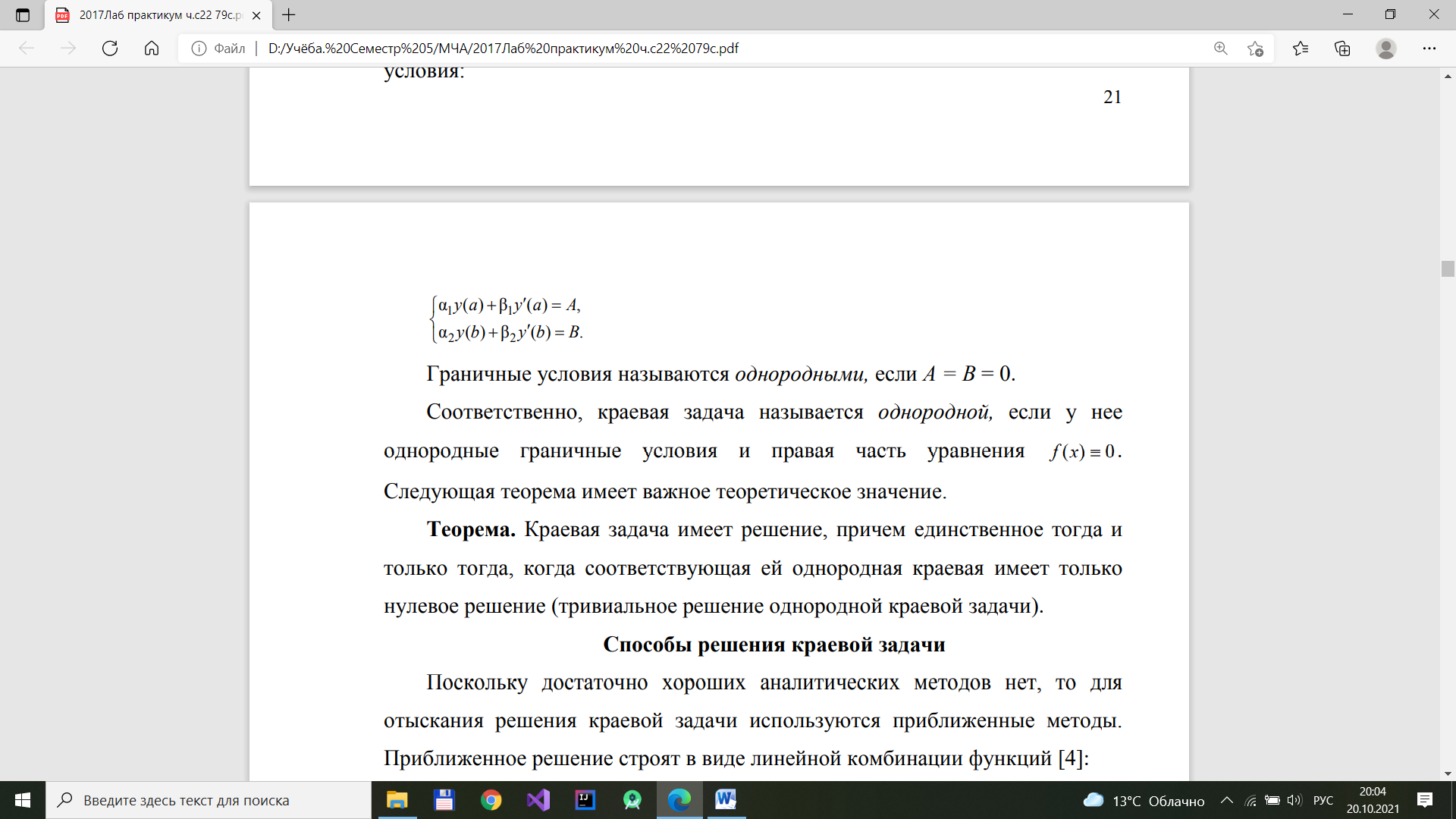
[Вывод 18](file:///C:\Users\Acer\Downloads\Telegram%20Desktop\МЧА%2011.docx#_Toc126867863)

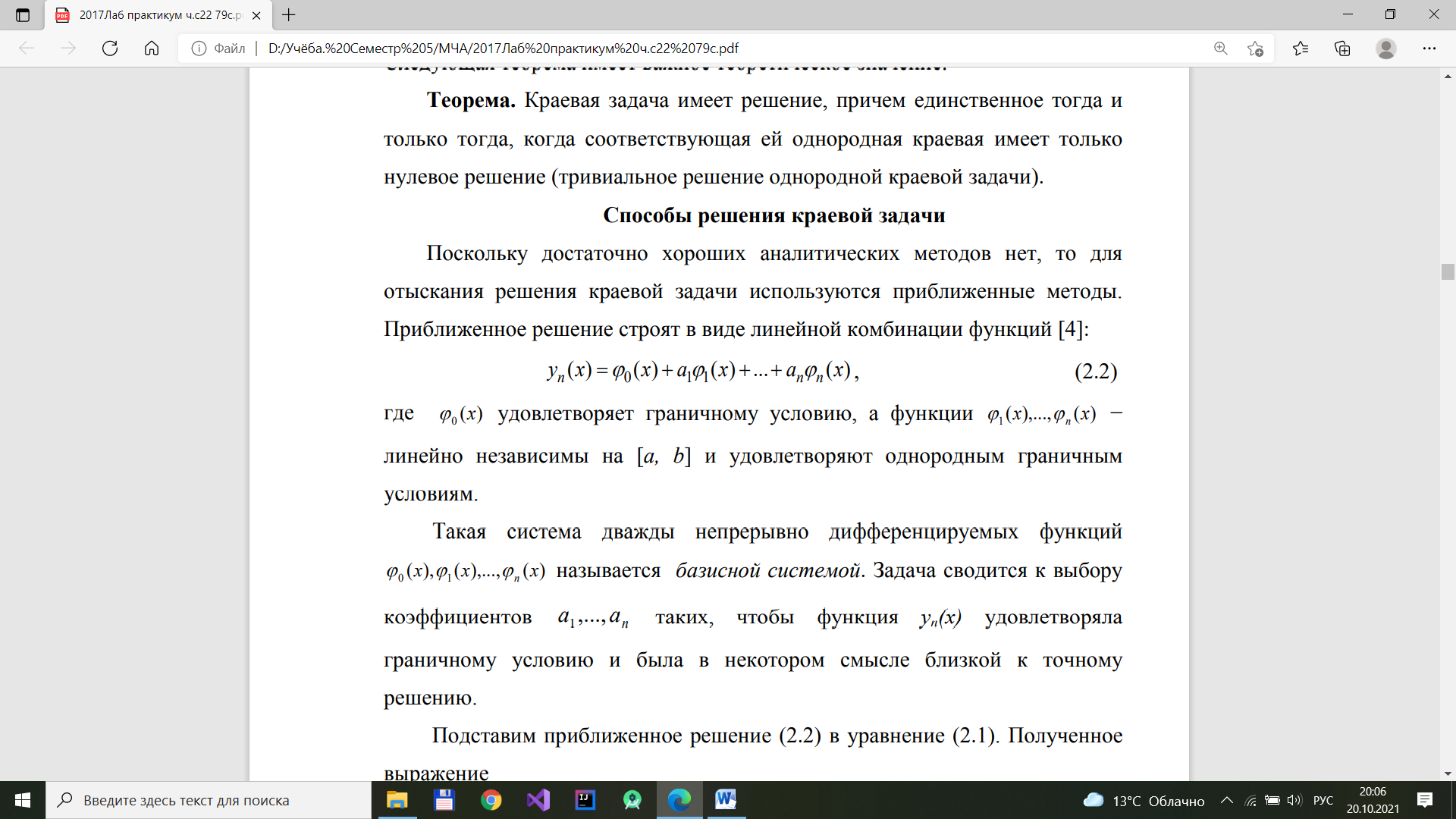
### Цели работы:

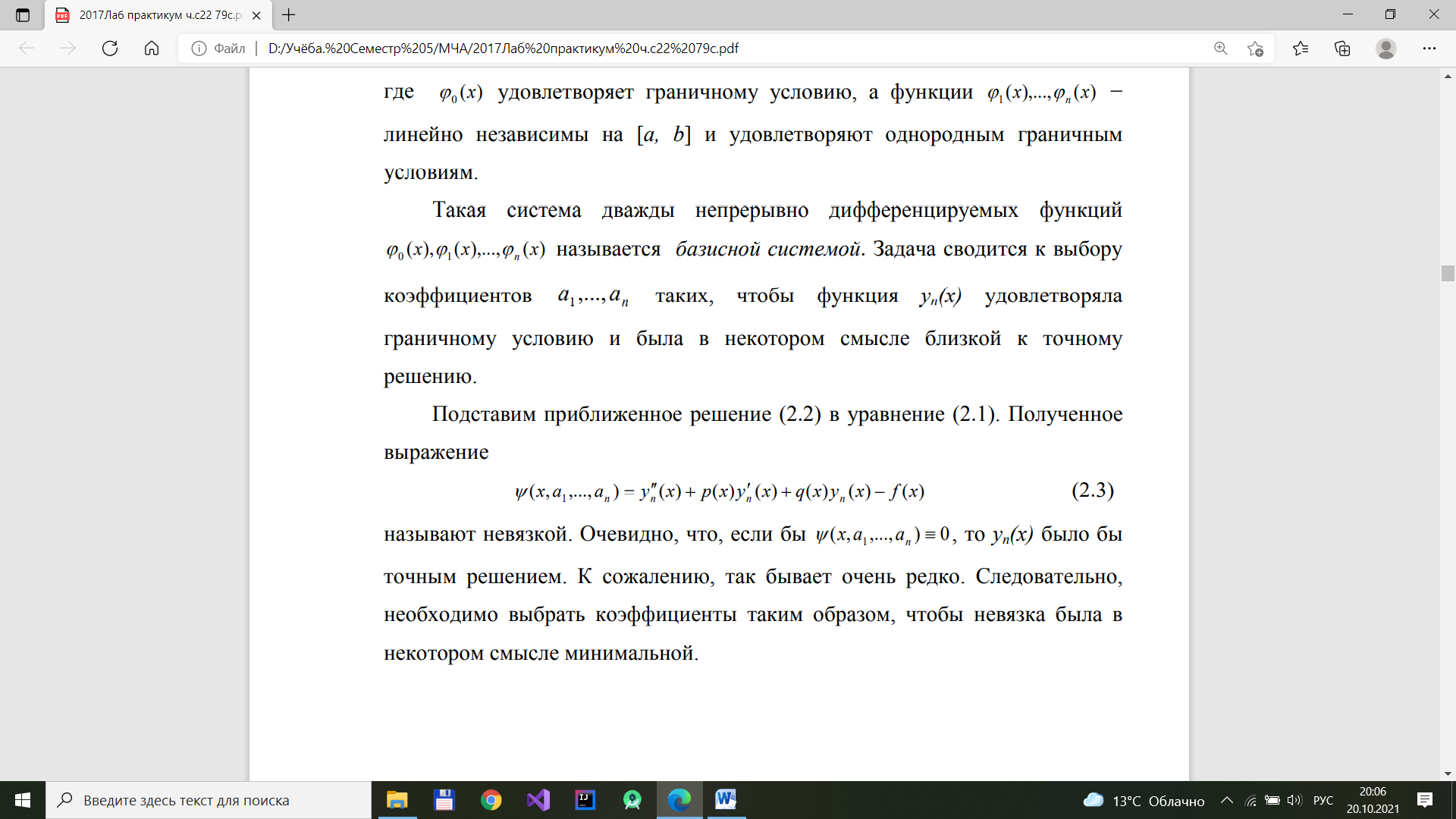
* изучить методы коллокаций, наименьших квадратов и Галеркина;
* составить алгоритмы методов и программы их реализаций;
* составить алгоритм решения краевых задач указанными методами, применимыми для организации вычислений на ПЭВМ;
* составить программу решения краевых задач по разработанным алгоритмам;
* выполнить тестовые примеры и проверить правильность работы программы;
* получить численное решение заданной краевой задачи.

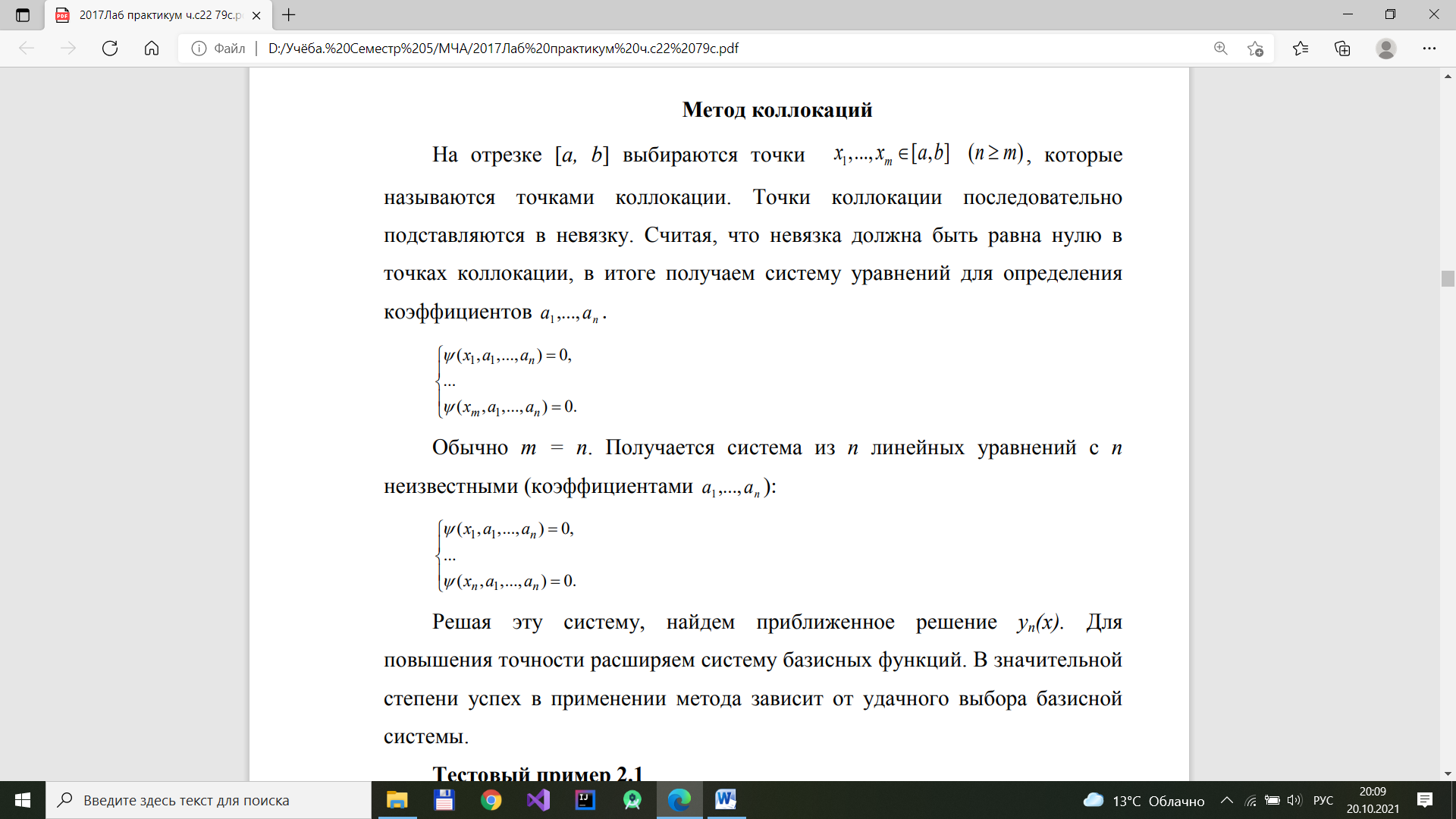
### Краткие теоретические сведения

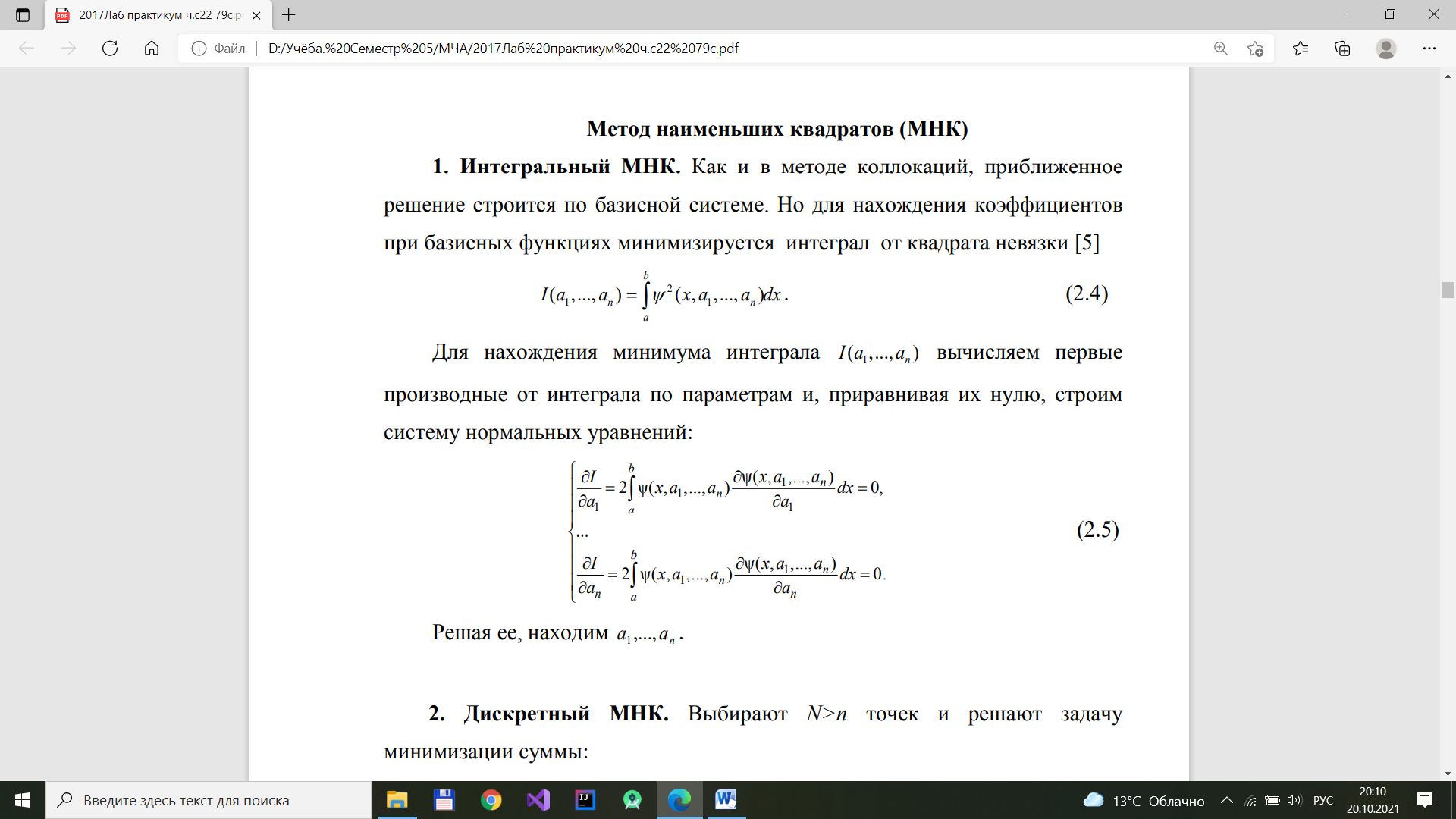


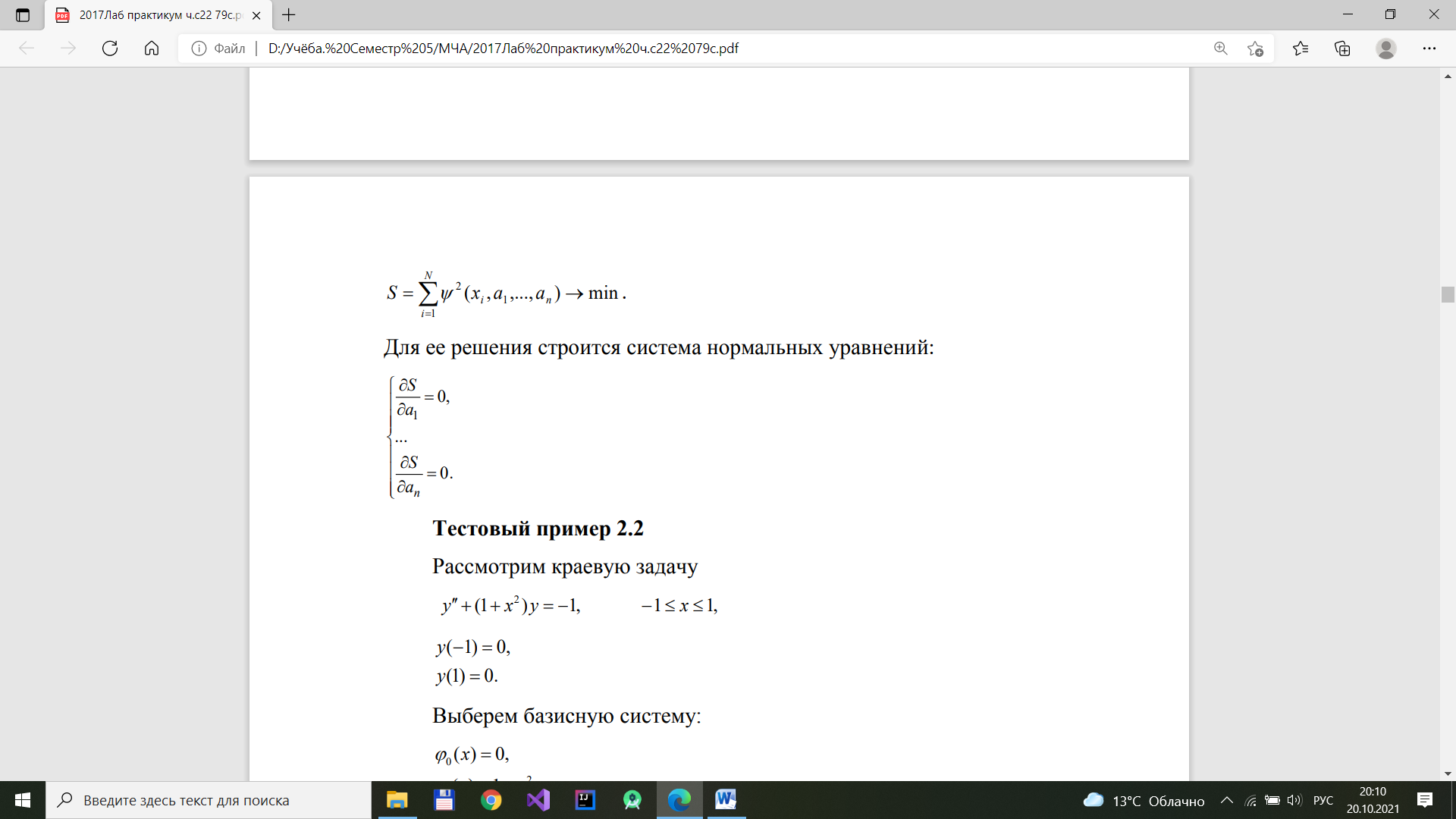


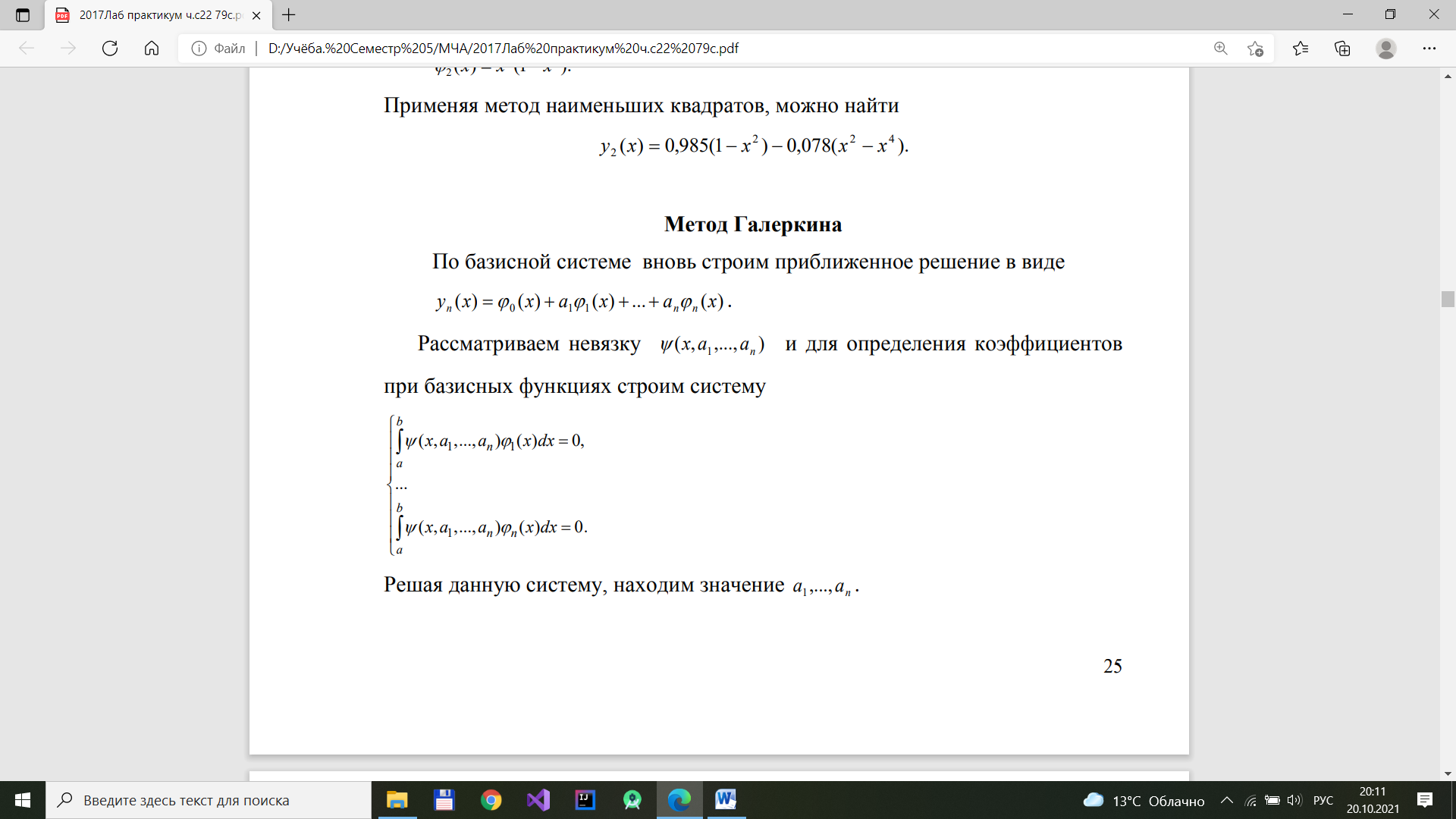






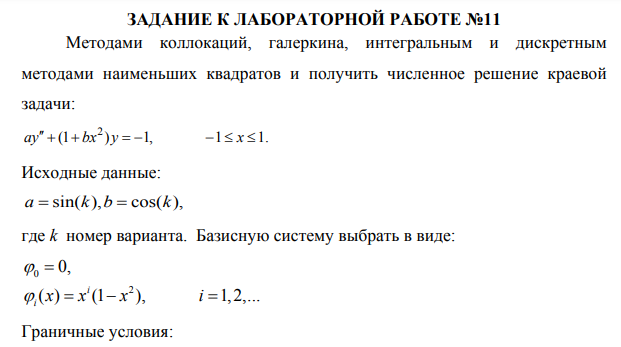




****

### Задание

Вариант 9.



y(-1) = 0,

y(1) = 0.

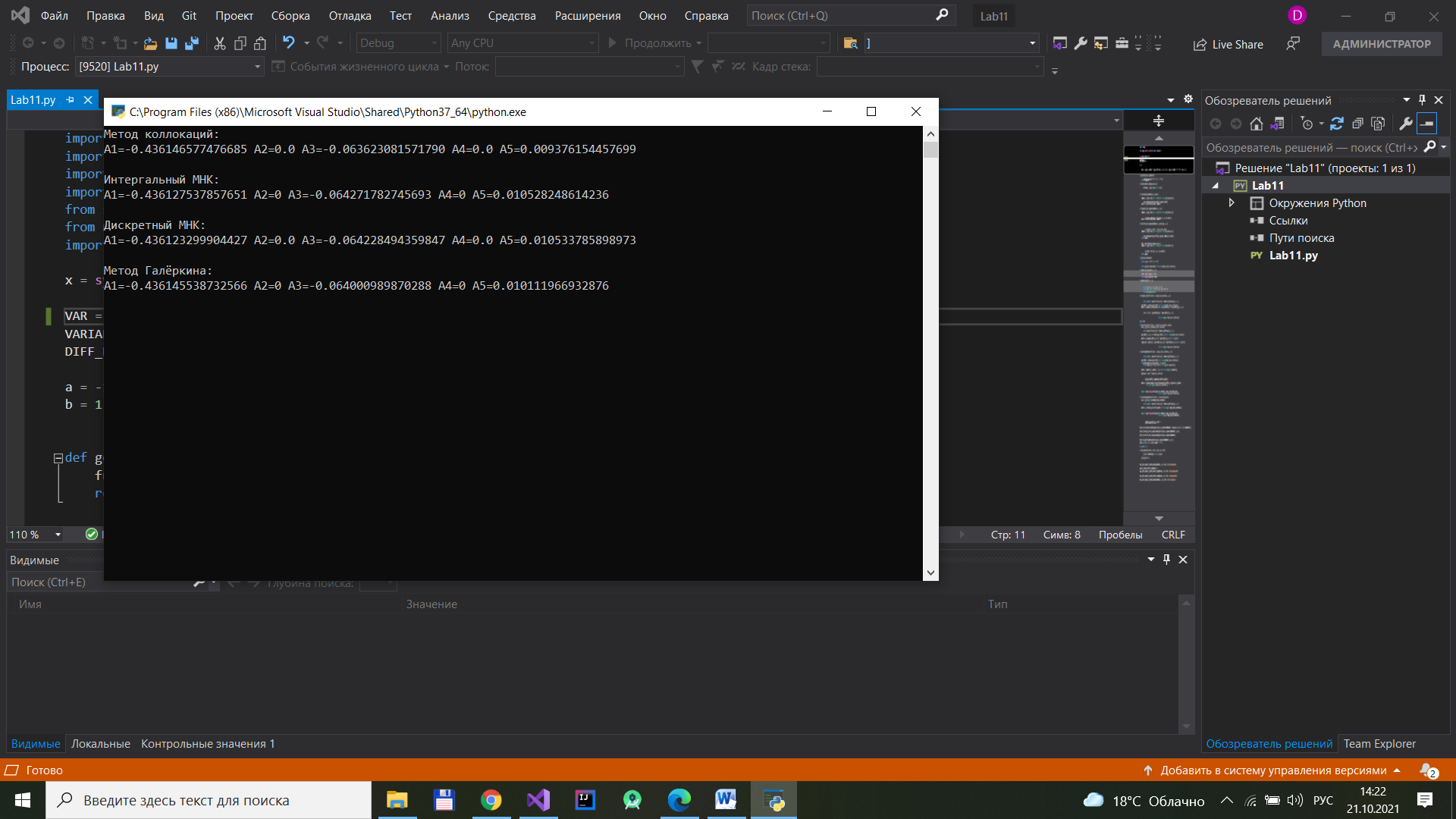
### Алгоритм задания

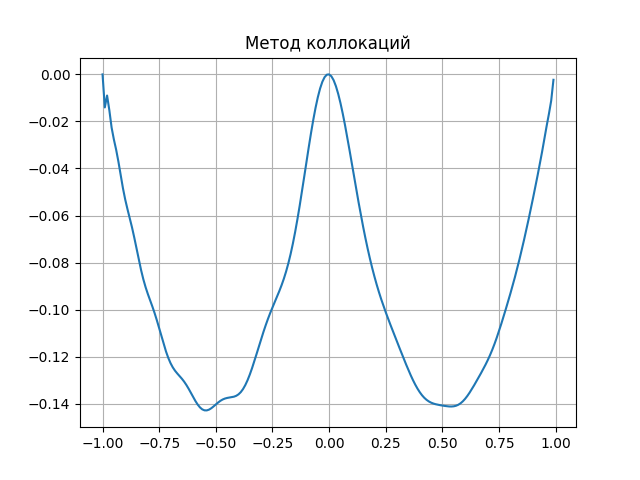
### Программная реализация

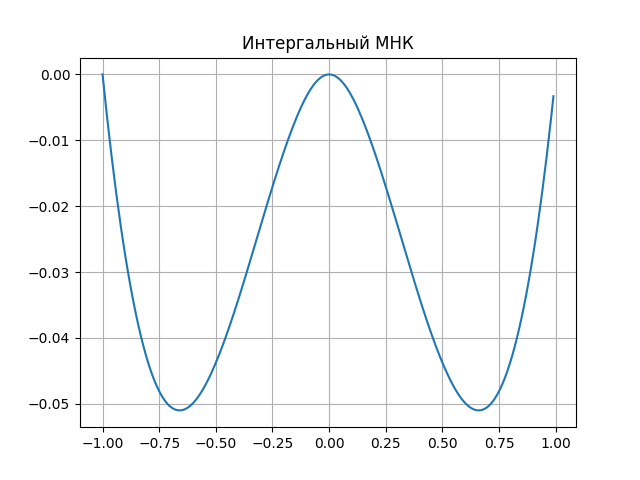
import math  
import numpy  
from numpy import linspace  
import pylab  
import sympy  
from sympy.solvers.solveset import linsolve  
import functools  
  
x = sympy.Symbol('x')  
  
k = 9  
num\_var = 5  
  
a = -1  
b = 1  
  
def task():  
 dif\_p = 5  
 coeffs = [lambda x: 1 + math.cos(k) \* x \*\* 2,  
 lambda x: 0,  
 lambda x: math.sin(k)]  
 f = lambda x: -1  
 print("")  
 print(linspace(a + 0.2, b - 0.2, num\_var))  
 print("")  
 print("Метод коллокаций: ", list(  
 CollocationMethod.collocations\_method(generate\_basis\_sequence(num\_var), linspace(a + 0.2, b - 0.2, num\_var))))  
 print("Интегральный МНК: ", list(LSM.integral\_least\_square\_method(generate\_basis\_sequence(num\_var), a, b)))  
 print("Дискретный МНК: ",  
 list(LSM.discrete\_least\_square\_method(generate\_basis\_sequence(num\_var), num\_var + dif\_p, a, b)))  
 print("Метод Галеркина: ", list(GalerkinMethod.galerkin\_method(generate\_basis\_sequence(num\_var), a, b)))  
 collocation\_points = numpy.linspace(a, b, 100)  
 res\_collocation = CollocationMethod.collocation\_method(coeffs, f, collocation\_points)  
 res\_integral = LSM.integral\_LSM(coeffs, f, 50, a, b)  
 points = numpy.linspace(a, b, 200)  
 res\_discrete = LSM.discrete\_LSM(coeffs, f, 100, points)  
 res\_galerkin = GalerkinMethod.Galerkin\_method(coeffs, f, 50, a, b)  
 show\_plots([[res\_collocation]], a, b, 0.01, "Метод коллокаций")  
 show\_plots([[res\_integral]], a, b, 0.01, "Интегральный МНК")  
 show\_plots([[res\_discrete]], a, b, 0.01, "Дискретный МНК")  
 show\_plots([[res\_galerkin]], a, b, 0.01, "Метод Галеркина")  
  
  
# show\_plots([[res\_collocation], [res\_integral], [res\_discrete],[res\_galerkin]], a, b, 0.01, "All plots combined: ")  
  
def func\_for\_substantiation(subs):  
 func = sympy.sin(k) \* sympy.diff(subs, x, x) + ((1 + sympy.cos(k) \* x \*\* 2) \* subs + 1)  
  
 return func  
  
  
def generate\_basis\_sequence(n):  
 sequence = []  
 for i in range(n):  
 sequence.append((x \*\* i) \* (1 - x \*\* 2))  
  
 return sequence  
  
  
def build\_function\_from\_basis(basis):  
 result = 0  
 for i in range(len(basis)):  
 current\_a = sympy.Symbol('a' + str(i))  
 result += current\_a \* basis[i]  
  
 return result  
  
  
def get\_basis\_function(n):  
 if not n:  
 return lambda x: 0  
  
 return lambda x: x \*\* (n - 1) \* (1 - x \*\* 2)  
  
  
def get\_basis\_system(num\_of\_basis\_functions):  
 return [get\_basis\_function(n) for n in range(num\_of\_basis\_functions)]  
  
  
def numerical\_integration(f, a, b):  
 n = 100  
 dx = (b - a) / n  
 xlist = numpy.arange(a, b, dx)  
 ylist = [f(p) for p in xlist]  
  
 return numpy.trapz(ylist, dx=dx)  
  
  
def numerical\_diff(f, x, n):  
 h = 0.001  
 if not n:  
 return f(x)  
 elif n == 1:  
 return (f(x + h) - f(x - h)) / (2 \* h)  
 elif n == 2:  
 return (f(x - h) - 2 \* f(x) + f(x + h)) / (h \*\* 2)  
 else:  
 raise NotImplementedError  
  
  
def show\_plots(functions, start\_x, end\_x, dx, title):  
 for f in functions:  
 for function in f:  
 x\_list = numpy.arange(start\_x, end\_x, dx)  
 y\_list = [function(p) for p in x\_list]  
 pylab.plot(x\_list, y\_list)  
 pylab.title(title)  
 pylab.grid(True)  
 pylab.show()  
  
  
class CollocationMethod:  
 def collocations\_method(basis, points):  
 func = build\_function\_from\_basis(basis)  
 psi\_func = func\_for\_substantiation(func)  
 symbols = [sympy.Symbol('a' + str(i)) for i in range(len(points))]  
 lin\_system = []  
 for point in points:  
 lin\_system.append(psi\_func.subs(x, point).evalf())  
 answer = linsolve(lin\_system, \*symbols)  
  
 return answer  
  
  
 def collocation\_method(coefficients, f, collocation\_points):  
 num\_of\_basis\_functions = len(collocation\_points) + 1  
 basis = get\_basis\_system(num\_of\_basis\_functions)  
  
 def resudial\_part\_diff(a):  
 return lambda x: sum(coefficients[i](x) \* numerical\_diff(basis[a], x, i)  
 for i in range(len(coefficients)))  
  
 matrix = [[resudial\_part\_diff(i)(point) for i in range(1, num\_of\_basis\_functions)]  
 for point in collocation\_points]  
 right\_side = [f(point) - resudial\_part\_diff(0)(point)  
 for point in collocation\_points]  
 answer = numpy.linalg.solve(numpy.matrix(matrix), numpy.array(right\_side))  
  
 return lambda x: basis[0](x) + sum(answer[i - 1] \* basis[i](x) for i in range(1, num\_of\_basis\_functions))  
  
  
class GalerkinMethod:  
 def Galerkin\_method(coefficients, f, num\_of\_basis\_functions, a, b):  
 basis = get\_basis\_system(num\_of\_basis\_functions)  
  
 def resudial\_part\_diff(a):  
 return lambda x: sum(coefficients[i](x) \* numerical\_diff(basis[a], x, i)  
 for i in range(len(coefficients)))  
  
 part\_diffs = [resudial\_part\_diff(i) for i in range(num\_of\_basis\_functions)]  
  
 def optimal\_numerical\_integration(f1, f2, points):  
 dx = 2 / len(points);  
  
 return sum([(f1[i] \* f2[i] \* dx) for i in range(0, len(points))])  
  
 points = numpy.linspace(a, b, 100)  
 points = [(points[i] + points[i - 1]) / 2 for i in range(1, len(points))]  
  
 opt\_part\_diffs = [None] \* num\_of\_basis\_functions  
 opt\_basis = [None] \* num\_of\_basis\_functions  
 for i in range(1, num\_of\_basis\_functions):  
 opt\_part\_diffs[i] = []  
 opt\_basis[i] = []  
 for point in points:  
 opt\_part\_diffs[i].append(part\_diffs[i](point))  
 opt\_basis[i].append(basis[i](point))  
  
 matrix = [[optimal\_numerical\_integration(opt\_part\_diffs[j], opt\_basis[i], points)  
 for j in range(1, num\_of\_basis\_functions)]  
 for i in range(1, num\_of\_basis\_functions)]  
 right\_side = [numerical\_integration(  
 lambda x: (f(x) - part\_diffs[0](x)) \* basis[i](x), a, b)  
 for i in range(1, num\_of\_basis\_functions)]  
 answer = numpy.linalg.solve(numpy.matrix(matrix), numpy.array(right\_side))  
 return lambda x: basis[0](x) + sum(answer[i - 1] \* basis[i](x)  
 for i in range(1, num\_of\_basis\_functions))  
  
 def galerkin\_method(basis, a, b):  
 func = build\_function\_from\_basis(basis)  
 psi\_func = func\_for\_substantiation(func)  
 symbols = [sympy.Symbol('a' + str(i)) for i in range(len(basis))]  
 lin\_system = []  
 for i in range(len(basis)):  
 lin\_system.append(sympy.integrate(  
 psi\_func \* basis[i], (x, a, b)).evalf())  
 answer = linsolve(lin\_system, \*symbols)  
  
 return answer  
  
  
class LSM:  
 def integral\_LSM(coefficients, f, num\_of\_basis\_functions, a, b):  
 basis = get\_basis\_system(num\_of\_basis\_functions)  
  
 def resudial\_part\_diff(a):  
 return lambda x: sum(coefficients[i](x) \* numerical\_diff(basis[a], x, i)  
 for i in range(len(coefficients)))  
  
 part\_diffs = [resudial\_part\_diff(i) for i in range(num\_of\_basis\_functions)]  
  
 matrix = [[numerical\_integration(lambda x: part\_diffs[i](x) \* part\_diffs[j](x), a, b)  
 for j in range(1, num\_of\_basis\_functions)]  
 for i in range(1, num\_of\_basis\_functions)]  
 right\_side = [numerical\_integration(  
 lambda x: (f(x) - part\_diffs[0](x)) \* part\_diffs[i](x), a, b)  
 for i in range(1, num\_of\_basis\_functions)]  
 answer = numpy.linalg.solve(numpy.matrix(matrix), numpy.array(right\_side))  
  
 return lambda x: basis[0](x) + sum(answer[i - 1] \* basis[i](x)  
 for i in range(1, num\_of\_basis\_functions))  
  
 def discrete\_LSM(coefficients, f, num\_of\_basis\_functions, points):  
 basis = get\_basis\_system(num\_of\_basis\_functions)  
  
 def resudial\_part\_diff(a, x):  
 return sum(coefficients[i](x) \* numerical\_diff(basis[a], x, i)  
 for i in range(len(coefficients)))  
  
 part\_diffs\_x = {(i, x): resudial\_part\_diff(i, x) for i in range(num\_of\_basis\_functions)  
 for x in points}  
  
 matrix = [[sum(part\_diffs\_x[(i, x)] \* part\_diffs\_x[(j, x)] for x in points)  
 for j in range(1, num\_of\_basis\_functions)]  
 for i in range(1, num\_of\_basis\_functions)]  
 right\_side = [sum([(f(x) - part\_diffs\_x[(0, x)]) \* part\_diffs\_x[(i, x)] for x in points])  
 for i in range(1, num\_of\_basis\_functions)]  
 answer = numpy.linalg.solve(numpy.matrix(matrix), numpy.array(right\_side))  
  
 return lambda x: basis[0](x) + sum(answer[i - 1] \* basis[i](x)  
 for i in range(1, num\_of\_basis\_functions))  
  
 def integral\_least\_square\_method(basis, a, b):  
 func = build\_function\_from\_basis(basis)  
 psi\_func = func\_for\_substantiation(func)  
 symbols = [sympy.Symbol('a' + str(i)) for i in range(len(basis))]  
 lin\_system = []  
 for i in range(len(basis)):  
 lin\_system.append(sympy.integrate(2 \* sympy.diff(  
 psi\_func, symbols[i]) \* psi\_func, (x, a, b)).evalf())  
 answer = linsolve(lin\_system, \*symbols)  
  
 return answer  
  
 def discrete\_least\_square\_method(basis, points\_num, a, b):  
 func = build\_function\_from\_basis(basis)  
 psi\_func = func\_for\_substantiation(func)  
 seq = [psi\_func.subs(x, point) \*\* 2 for point  
 in linspace(a + 0.05, b - 0.05, points\_num)]  
 psi\_sqr\_sum = functools.reduce((lambda a, b: a + b), seq)  
 symbols = [sympy.Symbol('a' + str(i)) for i in range(len(basis))]  
 lin\_system = []  
 for i in range(len(basis)):  
 lin\_system.append(sympy.diff(psi\_sqr\_sum, symbols[i]).evalf())  
 answer = linsolve(lin\_system, \*symbols)  
   
 return answer  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 task()

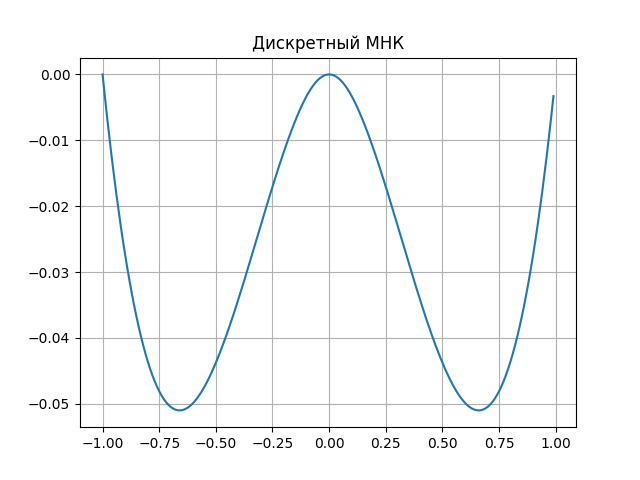
### Тестовые примеры

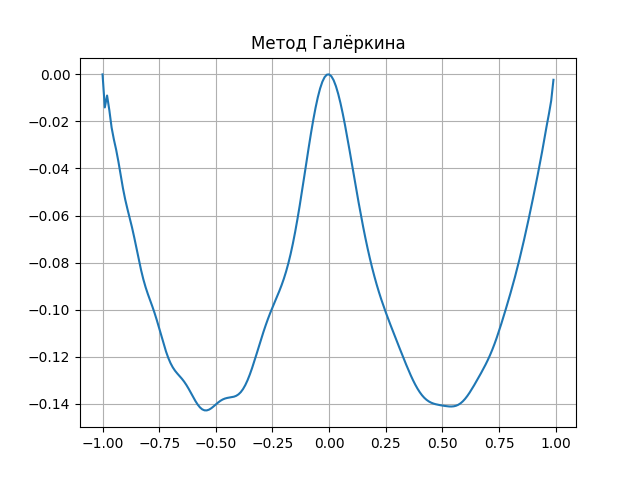
Заменим аргумент тригонометрической функции на 4:











### Результат выполнения программы

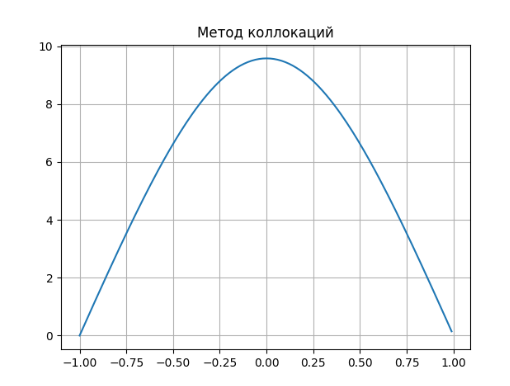
[-0.8 -0.4 0. 0.4 0.8]

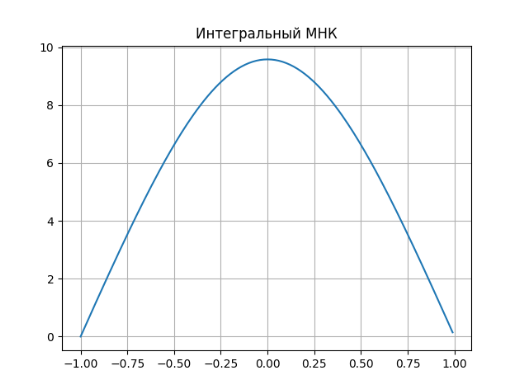
Метод коллокаций: [(9.54531681807719, 3.28885084715462e-16, -3.24871838746704, -9.12413989113227e-19, 1.01089376469211)]

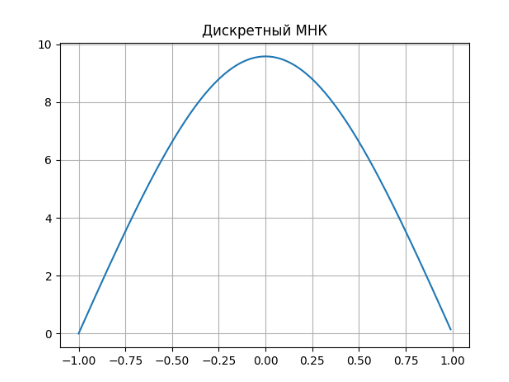
Интегральный МНК: [(9.52576430789249, 0, -3.14936350952563, 0, 0.851996975159579)]

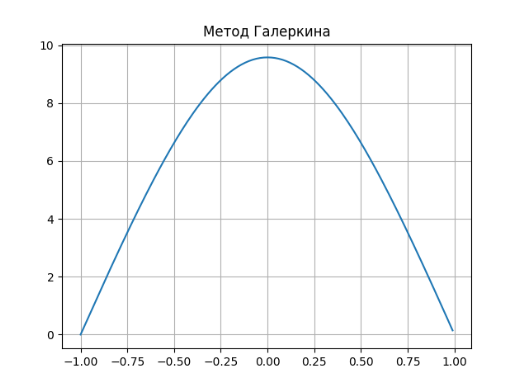
Дискретный МНК: [(9.52495177233788, 3.49953781742324e-16,   
-3.15547365762523, -3.19391306049021e-16, 0.852031412642482)]

Метод Галеркина: [(9.57636494004129, 0, -3.20583862717072, 0, 0.912158166786396)]









### Вывод

В ходе лабораторной работы были изучены четыре метода решения краевой задачи коллокаций, наименьших квадратов и Галеркина, составлены алгоритмы методов и программы их реализаций. Также была составлена программа решения краевых задач по разработанным алгоритмам, выполнены тестовые примеры и проверена правильность работы программы. В завершении работы было получено численное решение заданной краевой задачи.