Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

Факультет программной инженерии и компьютерной техники

**Отчет по лабораторной работе №4**

**по дисциплине «Вычислительная математика»**

**Вариант 11**

**Выполнили**:

Студенты группы P3265

Кручинина Дарья Сергеевна

Москвитина Полина

**Преподаватель:**

Машина Екатерина Алексеевна

**Цели работы**

Найти функцию, являющуюся

наилучшим приближением заданной табличной функции по методу

наименьших квадратов.

**Порядок выполнения работы**

**Вычислительная реализация задачи**

Вычислительная часть лабораторной работы должна быть представлена

только в отчете.

Задание:

1. Сформировать таблицу табулирования заданной функции на

указанном интервале (см. табл. 1)

2. Построить линейное и квадратичное приближения по 11 точкам

заданного интервала;

3. Найти среднеквадратические отклонения для каждой

аппроксимирующей функции. Ответы дать с тремя знаками после

запятой;

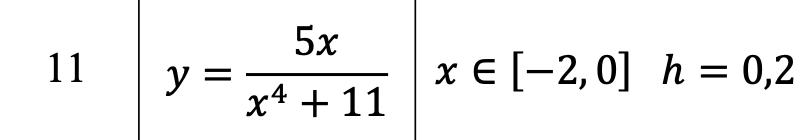
4. Выбрать наилучшее приближение;

5. Построить графики заданной функции, а также полученные

линейное и квадратичное приближения;

6. Привести в отчете **подробные вычисления**.





**Таблица табулирования:**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Xi | -2 | -1.8 | -1.6 | -1.4 | -1.2 | -1.0 | -0.8 | -0.6 | -0.4 | -0.2 | 0 |
| Yi | -0.37 | -0.419 | -0.456 | -0.472 | -0.459 | -0.417 | -0.351 | -0.27 | -0.181 | -0.091 | 0 |

**Линейное приближение:**

Вычисляем суммы:

Получаем систему линейных уравнений:

Из которой находим (правило Крамера):

a = 0,205

b = -0,112

Вычисляем значения аппроксимирующей функции :

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Xi | -2 | -1.8 | -1.6 | -1.4 | -1.2 | -1.0 | -0.8 | -0.6 | -0.4 | -0.2 | 0 |
| Yi | -0.37 | -0.419 | -0.456 | -0.472 | -0.459 | -0.417 | -0.351 | -0.27 | -0.181 | -0.091 | 0 |
|  | -0.522 | -0.481 | -0.44 | -0.399 | -0.358 | -0.317 | -0.276 | -0.235 | -0.194 | -0.153 | -0.112 |
| Ei | -0.152 | -0.062 | 0.016 | 0.073 | 0.101 | 0.100 | 0.075 | 0.035 | -0.013 | -0.062 | -0.112 |

**Квадратичное приближение:**

Вычисляем суммы:

Получили систему линейных уравнений, решив которую, определим значения

коэффициентов эмпирической формулы:

Из которой находим:

a = 0,027

b = 0,668

c = 0,232

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Xi | -2 | -1.8 | -1.6 | -1.4 | -1.2 | -1.0 | -0.8 | -0.6 | -0.4 | -0.2 | 0 |
| Yi | -0.37 | -0.419 | -0.456 | -0.472 | -0.459 | -0.417 | -0.351 | -0.27 | -0.181 | -0.091 | 0 |
|  | -0.383 | -0.425 | -0.449 | -0.454 | -0.441 | -0.409 | -0.359 | -0.291 | -0.203 | -0.098 | -0.026 |
| Ei | -0.013 | -0.006 | 0.007 | 0.018 | 0.018 | 0.008 | -0.008 | -0.021 | -0.022 | -0.007 | -0.026 |

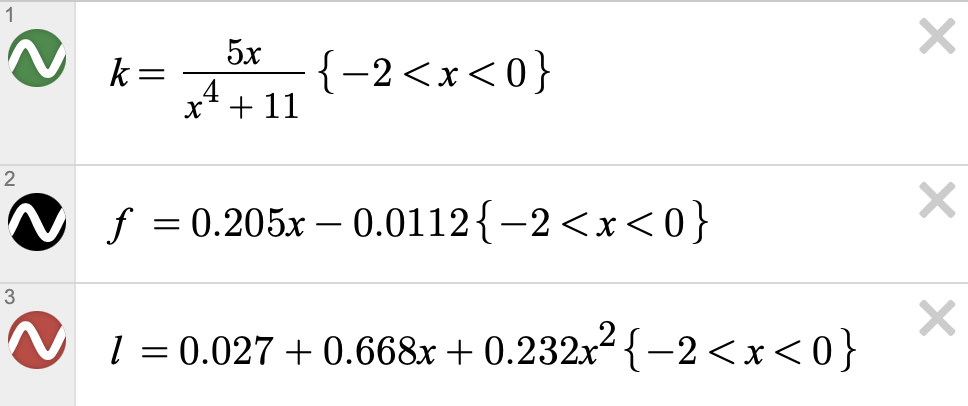
**Среднеквадратические отклонения:**

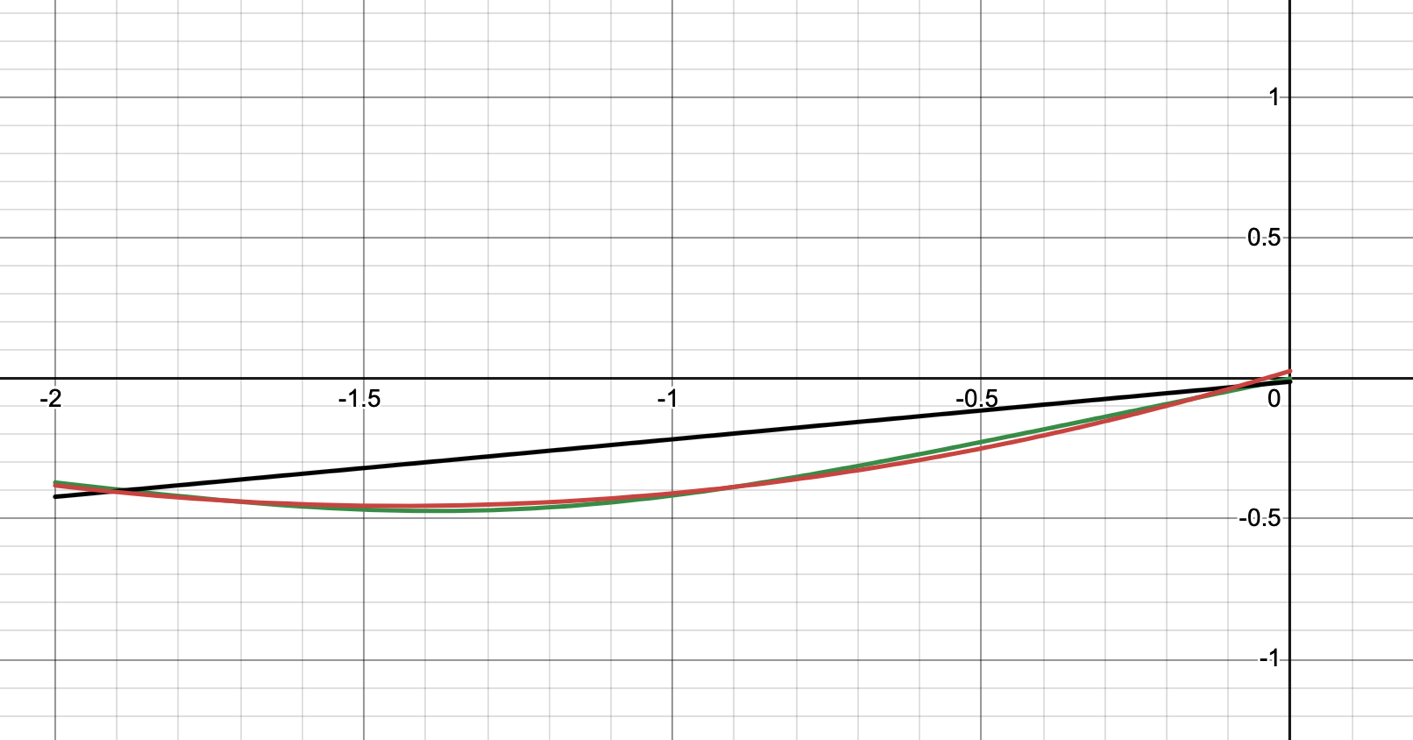
(отклонение линейного приближения)

(отклонение квадратичного приближения)

Наилучшим приближением является квадратичное, так как оно меньше

Графики:





**Программная реализация задачи**

**Для исследования использовать:**

линейную функцию,

полиномиальную функцию 2-й степени,

полиномиальную функцию 3-й степени,

экспоненциальную функцию,

логарифмическую функцию,

степенную функцию.

**from** \_\_future\_\_ **import** annotations

**import** math

**import** os

**from** parser **import** Parser

**from** dto **import** PointTable, ApproxRes

**from** approx **import** APPROXIMATORS

**from** utils **import** draw\_graph

**def** main() -> **None**:

os.chdir('/' + '/'.join(\_\_file\_\_.split('/')[:-2]) + '/resources')

data: PointTable | **None** = Parser.parse\_table\_from\_file(input('Type input filename -> '))

**if** data **is** **None**:

**return**

Parser.clear\_file()

min\_sko: float = math.inf

min\_type: str | **None** = **None**

**for** approximator **in** APPROXIMATORS:

res: ApproxRes = approximator(data)

Parser.print\_res(res)

**if** res.data **is** **not** **None**:

draw\_graph(res.data, res.type)

**if** res.data.sko < min\_sko:

min\_sko = res.data.sko

min\_type = res.type

**if** min\_type **is** **not** **None**:

print(f'Best approximator is {min\_type} with sko = {min\_sko:.4g}')

**if** \_\_name\_\_ == '\_\_main\_\_':

main()

**import** math

**from** dto **import** \*

**from** utils **import** avg

**from** enum **import** Enum

**from** slau\_solver.matrix **import** Equation

**def** calc\_pirson\_kf(def\_data: DataTable) -> float:

x\_list: list[float] = def\_data.x\_list

y\_list: list[float] = def\_data.y\_list

n: int = len(x\_list)

avg\_x = avg(x\_list)

avg\_y = avg(y\_list)

**return** (sum([(x\_list[i] - avg\_x) \* (y\_list[i] - avg\_y) **for** i **in** range(n)]) /

math.sqrt(sum([(x\_list[i] - avg\_x) \*\* 2 **for** i **in** range(n)]) \*

sum([(y\_list[i] - avg\_y) \*\* 2 **for** i **in** range(n)])

))

**def** calc\_det\_kf(def\_data: DataTable) -> float:

phi\_list: list[float] = def\_data.phi\_x

y\_list: list[float] = def\_data.y\_list

n: int = len(phi\_list)

avg\_phi = avg(phi\_list)

**return** (1 - sum([(y\_list[i] - phi\_list[i]) \*\* 2 **for** i **in** range(n)]) /

sum([(y\_list[i] - avg\_phi) \*\* 2 **for** i **in** range(n)]))

**def** calc\_sko(eps: list[float]) -> float:

n: int = len(eps)

**return** math.sqrt(sum([eps[i] \*\* 2 **for** i **in** range(n)]) / n)

**def** get\_def\_data(points: PointTable, func: callable) -> DataTable:

x\_list = points.get\_all\_x()

y\_list = points.get\_all\_y()

phi\_x = [func(x) **for** x **in** x\_list]

eps = [phi\_x[i] - y\_list[i] **for** i **in** range(points.n)]

**return** DataTable(x\_list, y\_list, phi\_x, eps)

**def** calc\_linear\_kfs(points: PointTable) -> tuple[float, float]:

sx, sy, sxx, sxy, n = (points.sx(), points.sy(),

points.sxx(), points.sxy(), points.n)

eq: Equation = Equation.create([

[sxx, sx],

[sx, n],

[sxy, sy]

])

eq.solve()

a, b = eq.answers.elems

**return** a, b

**def** approx\_linear(points: PointTable) -> ApproxRes:

a, b = calc\_linear\_kfs(points)

callback: callable = **lambda** x: a \* x + b

func\_view: str = f'{a:.3g}x'

**if** b > 0:

func\_view += f' + {b:.3g}'

**elif** b < 0:

func\_view += f' - {-b:.3g}x'

def\_data: DataTable = get\_def\_data(points, callback)

**return** ApproxRes(

type='Линейная аппроксимация',

data=LinearApproxData(

func\_view=func\_view,

callback=callback,

sko=calc\_sko(def\_data.eps),

x\_list=def\_data.x\_list,

y\_list=def\_data.y\_list,

phi\_x=def\_data.phi\_x,

eps=def\_data.eps,

pirson\_kf=calc\_pirson\_kf(def\_data),

det\_kf=calc\_det\_kf(def\_data)

),

error\_message=**None**

)

**def** approx\_quad(points: PointTable) -> ApproxRes:

sx, sxx, s3x, s4x, sy, sxy, sxxy, n = \

(points.sx(), points.sxx(), points.s3x(), points.s4x(),

points.sy(), points.sxy(), points.sxxy(), points.n)

eq: Equation = Equation.create([

[s4x, s3x, sxx],

[s3x, sxx, sx],

[sxx, sx, n],

[sxxy, sxy, sy]

])

eq.solve()

a, b, c = eq.answers.elems

callback: callable = **lambda** x: a \* x \* x + b \* x + c

func\_view: str = f'{a:.3g}x^2'

**if** b > 0:

func\_view += f' + {b:.3g}x'

**elif** b < 0:

func\_view += f' - {-b:.3g}x'

**if** c > 0:

func\_view += f' + {c:.3g}'

**elif** c < 0:

func\_view += f' - {-c:.3g}'

def\_data: DataTable = get\_def\_data(points, callback)

**return** ApproxRes(

type='Квадратичная аппроксимация',

data=ApproxData(

func\_view=func\_view,

callback=callback,

sko=calc\_sko(def\_data.eps),

x\_list=def\_data.x\_list,

y\_list=def\_data.y\_list,

phi\_x=def\_data.phi\_x,

eps=def\_data.eps,

det\_kf=calc\_det\_kf(def\_data)

),

error\_message=**None**

)

**def** approx\_cube(points: PointTable) -> ApproxRes:

sx, sxx, s3x, s4x, s5x, s6x, sy, sxy, sxxy, s3xy, n = \

(points.sx(), points.sxx(), points.s3x(), points.s4x(),

points.s5x(), points.s6x(), points.sy(), points.sxy(),

points.sxxy(), points.s3xy(), points.n)

eq: Equation = Equation.create([

[s6x, s5x, s4x, s3x],

[s5x, s4x, s3x, sxx],

[s4x, s3x, sxx, sx],

[s3x, sxx, sx, n],

[s3xy, sxxy, sxy, sy]

])

eq.solve()

a, b, c, d = eq.answers.elems

callback: callable = **lambda** x: a \* (x \*\* 3) + b \* (x \*\* 2) + c \* x + d

func\_view: str = f'{a:.3g}x^3'

**if** b > 0:

func\_view += f' + {b:.3g}x^2'

**elif** b < 0:

func\_view += f' - {-b:.3g}x^2'

**if** c > 0:

func\_view += f' + {c:.3g}x'

**elif** c < 0:

func\_view += f' - {-c:.3g}x'

**if** d > 0:

func\_view += f' + {d:.3g}'

**elif** d < 0:

func\_view += f' - {-d:.3g}'

def\_data: DataTable = get\_def\_data(points, callback)

**return** ApproxRes(

type='Кубическая аппроксимация',

data=ApproxData(

func\_view=func\_view,

callback=callback,

sko=calc\_sko(def\_data.eps),

x\_list=def\_data.x\_list,

y\_list=def\_data.y\_list,

phi\_x=def\_data.phi\_x,

eps=def\_data.eps,

det\_kf=calc\_det\_kf(def\_data)

),

error\_message=**None**

)

**def** approx\_exp(points: PointTable) -> ApproxRes:

**if** **not** points.log\_y\_is\_safe():

**return** ApproxRes(

type='Экспоненциальная аппроксимация',

data=**None**,

error\_message="Can't approximate with negative ordinates"

)

points\_copy: PointTable = points.copy()

**for** i **in** range(points.n):

points\_copy[i].y = math.log(points\_copy[i].y)

b, A = calc\_linear\_kfs(points\_copy)

a = math.exp(A)

callback: callable = **lambda** x: a \* math.exp(b \* x)

func\_view: str = f'{a:.3g}e^({b:.3g}x)'

def\_data: DataTable = get\_def\_data(points, callback)

**return** ApproxRes(

type='Экспоненциальная аппроксимация',

data=ApproxData(

func\_view=func\_view,

callback=callback,

sko=calc\_sko(def\_data.eps),

x\_list=def\_data.x\_list,

y\_list=def\_data.y\_list,

phi\_x=def\_data.phi\_x,

eps=def\_data.eps,

det\_kf=calc\_det\_kf(def\_data)

),

error\_message=**None**

)

**def** approx\_log(points: PointTable) -> ApproxRes:

**if** **not** points.log\_x\_is\_safe():

**return** ApproxRes(

type='Логарифмическая аппроксимация',

data=**None**,

error\_message="Can't approximate with negative abscisses"

)

points\_copy: PointTable = points.copy()

**for** i **in** range(points.n):

points\_copy[i].x = math.log(points\_copy[i].x)

a, b = calc\_linear\_kfs(points\_copy)

callback: callable = **lambda** x: a \* math.log(x) + b

func\_view: str = f'{a:.3g}ln(x)'

**if** b > 0:

func\_view += f' + {b:.3g}'

**elif** b < 0:

func\_view += f' - {-b:.3g}'

def\_data: DataTable = get\_def\_data(points, callback)

**return** ApproxRes(

type='Логарифмическая аппроксимация',

data=ApproxData(

func\_view=func\_view,

callback=callback,

sko=calc\_sko(def\_data.eps),

x\_list=def\_data.x\_list,

y\_list=def\_data.y\_list,

phi\_x=def\_data.phi\_x,

eps=def\_data.eps,

det\_kf=calc\_det\_kf(def\_data)

),

error\_message=**None**

)

**def** approx\_step(points: PointTable) -> ApproxRes:

**if** **not** points.log\_x\_is\_safe() **or** **not** points.log\_y\_is\_safe():

**return** ApproxRes(

type='Степпеная аппроксимация',

data=**None**,

error\_message="Can't approximate with negative abscisses or negative ordinates"

)

points\_copy: PointTable = points.copy()

**for** i **in** range(points.n):

points\_copy[i].x = math.log(points\_copy[i].x)

points\_copy[i].y = math.log(points\_copy[i].y)

b, A = calc\_linear\_kfs(points\_copy)

a = math.exp(A)

callback: callable = **lambda** x: a \* (x \*\* b)

func\_view: str = f'{a:.3g}x^({b:.3g})'

def\_data: DataTable = get\_def\_data(points, callback)

**return** ApproxRes(

type='Степпеная аппроксимация',

data=ApproxData(

func\_view=func\_view,

callback=callback,

sko=calc\_sko(def\_data.eps),

x\_list=def\_data.x\_list,

y\_list=def\_data.y\_list,

phi\_x=def\_data.phi\_x,

eps=def\_data.eps,

det\_kf=calc\_det\_kf(def\_data)

),

error\_message=**None**

)

APPROXIMATORS: list[callable] = [

approx\_linear,

approx\_quad,

approx\_cube,

approx\_exp,

approx\_log,

approx\_step

]

**from** \_\_future\_\_ **import** annotations

**from** dataclasses **import** dataclass

**import** pylab **as** p

@dataclass

**class** Point:

x: float

y: float

**def** \_\_str\_\_(self) -> str:

**return** f'({self.x}, {self.y})'

**def** copy(self) -> Point:

**return** Point(self.x, self.y)

**class** PointTable:

**def** \_\_init\_\_(self, points: list[Point]) -> **None**:

self.points = sorted(points, key=**lambda** point: point.x)

self.n = len(self.points)

**def** copy(self) -> PointTable:

**return** PointTable([point.copy() **for** point **in** self.points])

**def** \_\_getitem\_\_(self, index: int) -> Point:

**return** self.points[index]

**def** \_\_setitem\_\_(self, index: int, point: Point) -> **None**:

self.points[index] = point

**def** log\_y\_is\_safe(self) -> bool:

**for** point **in** self.points:

**if** point.y <= 0 **or** point.y == 1:

**return** **False**

**return** **True**

**def** log\_x\_is\_safe(self) -> bool:

**for** point **in** self.points:

**if** point.x <= 0 **or** point.x == 1:

**return** **False**

**return** **True**

**def** get\_all\_x(self) -> list[float]:

**return** [point.x **for** point **in** self.points]

**def** get\_all\_y(self) -> list[float]:

**return** [point.y **for** point **in** self.points]

**def** sx(self) -> float:

**return** sum([point.x **for** point **in** self.points])

**def** sxx(self) -> float:

**return** sum([point.x \*\* 2 **for** point **in** self.points])

**def** s3x(self) -> float:

**return** sum([point.x \*\* 3 **for** point **in** self.points])

**def** s4x(self) -> float:

**return** sum([point.x \*\* 4 **for** point **in** self.points])

**def** s5x(self) -> float:

**return** sum([point.x \*\* 5 **for** point **in** self.points])

**def** s6x(self) -> float:

**return** sum([point.x \*\* 6 **for** point **in** self.points])

**def** sy(self) -> float:

**return** sum([point.y **for** point **in** self.points])

**def** sxy(self) -> float:

**return** sum([point.x \* point.y **for** point **in** self.points])

**def** sxxy(self) -> float:

**return** sum([point.x \* point.x \* point.y **for** point **in** self.points])

**def** s3xy(self) -> float:

**return** sum([(point.x \*\* 3) \* point.y **for** point **in** self.points])

@dataclass

**class** DataTable:

x\_list: list[float]

y\_list: list[float]

phi\_x: list[float]

eps: list[float]

@dataclass

**class** ApproxData:

func\_view: str

callback: callable

sko: float

x\_list: list[float]

y\_list: list[float]

phi\_x: list[float]

eps: list[float]

det\_kf: float

**def** \_\_str\_\_(self) -> str:

**return** (f"phi: {self.func\_view}\n" +

f"sko: {self.sko:.3g}\n" +

f"det\_kf: {self.det\_kf:.3g}\n\n")

@dataclass

**class** LinearApproxData(ApproxData):

pirson\_kf: float

**def** \_\_str\_\_(self) -> str:

**return** f'{super().\_\_str\_\_()[:-1]}pirson: {self.pirson\_kf:.3g}\n\n'

@dataclass

**class** ApproxRes:

type: str

data: ApproxData | **None**

error\_message: str | **None**

**def** \_\_str\_\_(self) -> str:

prefix = f'-- {self.type}\n'

**if** self.error\_message **is** **not** **None**:

**return** f'{prefix}\tERROR: {self.error\_message}\n\n'

**else**:

**return** f'{prefix}{self.data}'

**from** \_\_future\_\_ **import** annotations

**from** dto **import** PointTable, Point, ApproxRes

**from** utils **import** to\_float

DEFAULT\_FILE\_IN = 'test'

DEFAULT\_FILE\_OUT = 'output'

**class** Parser:

@staticmethod

**def** parse\_table\_from\_file(filename: str) -> PointTable | **None**:

**if** filename **is** **None** **or** filename == '':

filename = DEFAULT\_FILE\_IN

**try**:

x\_line: list[float]

y\_line: list[float]

**with** open(filename, 'r') **as** f:

x\_line = list(map(to\_float, f.readline().strip().split(' ')))

y\_line = list(map(to\_float, f.readline().strip().split(' ')))

**if** len(x\_line) != len(y\_line):

print('Length of X and Y are not equals')

**return** **None**

**if** len(x\_line) < 8 **or** len(x\_line) > 12:

print('Number of points should be between 8 and 12')

**return** **None**

points: list[Point] = []

**for** i **in** range(len(x\_line)):

points.append(Point(x\_line[i], y\_line[i]))

**return** PointTable(points)

**except** (IOError, ValueError):

print('File not exist or invalid')

@staticmethod

**def** print\_res(res: ApproxRes, filename: str = **None**) -> **None**:

**if** filename **is** **None** **or** filename == '':

filename = DEFAULT\_FILE\_OUT

**with** open(filename, 'a') **as** file:

file.write(str(res))

@staticmethod

**def** clear\_file(filename: str = **None**) -> **None**:

**if** filename **is** **None** **or** filename == '':

filename = DEFAULT\_FILE\_OUT

**with** open(filename, 'w'):

**pass**

**from** matplotlib.axes **import** Axes

**from** dto **import** \*

**import** matplotlib.pyplot **as** plt

**def** to\_float(val: str) -> float:

**return** float(val.replace(',', '.'))

**def** avg(data: list[float]) -> float:

**return** sum(data) / len(data)

**def** draw\_graph(data: ApproxData, title: str) -> **None**:

ax: Axes = plt.gca()

ax.spines['left'].set\_position('zero')

ax.spines['bottom'].set\_position('zero')

ax.spines['right'].set\_color('none')

ax.spines['top'].set\_color('none')

ax.set\_aspect('equal', adjustable='box')

plt.plot(data.x\_list, data.y\_list, 'ro', linewidth=3)

plt.plot(data.x\_list, data.phi\_x, linewidth=1, label=data.func\_view)

plt.grid(**True**)

plt.legend()

plt.title(title)

plt.show()

**from** \_\_future\_\_ **import** annotations

**from** typing **import** List

**import** random

**from** utils **import** to\_float

**class** Vector:

elems: List[float] = **None**

**def** \_\_init\_\_(self, elems\_or\_size: int | List[float]) -> **None**:

**if** isinstance(elems\_or\_size, List):

self.elems = elems\_or\_size

**else**:

self.elems = [0 **for** \_ **in** range(elems\_or\_size)]

**def** size(self):

**return** len(self.elems)

**def** copy(self) -> Vector:

**return** Vector(self.elems[:])

**def** \_\_getitem\_\_(self, index: int | slice) -> float | Vector:

**if** isinstance(index, slice):

**return** Vector(self.elems[index])

**else**:

**return** self.elems[index]

**def** \_\_setitem\_\_(self, key: int, value: float) -> **None**:

self.elems[key] = value

**def** swap\_elems(self, index1: int, index2: int) -> **None**:

self[index1], self[index2] = self[index2], self[index1]

**def** has\_only\_zeros(self) -> bool:

**return** max(self.elems) == 0 **and** min(self.elems) == 0

**def** \_\_mul\_\_(self, num\_or\_vector: float | Vector) -> Vector:

new\_vec: Vector = Vector(self.size())

**for** i **in** range(self.size()):

**if** isinstance(num\_or\_vector, float):

new\_vec[i] = self[i] \* num\_or\_vector

**else**:

new\_vec[i] = self[i] \* num\_or\_vector[i]

**return** new\_vec

**def** sum(self) -> float:

**return** sum(self.elems)

**def** \_\_add\_\_(self, other) -> Vector:

new\_vec: List[float] = [0 **for** \_ **in** range(self.size())]

**for** i **in** range(self.size()):

new\_vec[i] = self[i] + other[i]

**return** Vector(new\_vec)

**class** Matrix:

elems: List[Vector] = **None**

**def** \_\_init\_\_(self, elems\_or\_size: List[Vector] | int) -> **None**:

**if** isinstance(elems\_or\_size, int):

self.elems = [Vector(elems\_or\_size) **for** \_ **in** range(elems\_or\_size)]

**else**:

self.elems = elems\_or\_size

**def** \_\_getitem\_\_(self, index: int) -> Vector:

**return** self.elems[index]

**def** \_\_setitem\_\_(self, key: int, row: Vector) -> **None**:

self.elems[key] = row

**def** swap\_rows(self, index1: int, index2: int) -> **None**:

self[index1], self[index2] = self[index2], self[index1]

**def** size(self) -> int:

**return** len(self.elems)

**def** copy(self) -> Matrix:

**return** Matrix([row.copy() **for** row **in** self.elems])

**def** is\_correct(self) -> bool:

**for** row **in** self.elems:

**if** row.has\_only\_zeros():

**return** **False**

**for** i **in** range(self.size()):

col\_has\_non\_zero: bool = **False**

**for** j **in** range(self.size()):

**if** self[j][i] != 0:

col\_has\_non\_zero = **True**

**if** **not** col\_has\_non\_zero:

**return** **False**

**return** **True**

**class** Equation:

matrix: Matrix = **None**

b\_vector: Vector = **None**

det: float = **None**

triangled: Equation = **None**

answers: Vector = **None**

residuals: Vector = **None**

**def** \_\_init\_\_(self, matrix: Matrix, b\_vector: Vector) -> **None**:

self.matrix = matrix

self.b\_vector = b\_vector

@staticmethod

**def** create(data: list[list[float]]) -> Equation | **None**:

size: int = len(data) - 1

**for** row **in** data:

**if** len(row) != size:

print('Invalid data in Equation.create')

**return** **None**

matrix: Matrix = Matrix(size)

b\_vector: Vector = Vector(data[-1])

**for** i **in** range(size):

row: Vector = Vector(data[i])

matrix[i] = row

**return** Equation(matrix, b\_vector)

**def** copy(self) -> Equation:

**return** Equation(self.matrix.copy(), self.b\_vector.copy())

**def** size(self) -> int:

**return** self.b\_vector.size()

# Приведение к треугольному виду

**def** triangle(self) -> **None**:

perm\_count = 0

triangled\_eq: Equation = self.copy()

**for** i **in** range(triangled\_eq.size() - 1):

# Выбор главного элемента

max\_row\_idx: int = i

**for** j **in** range(i, triangled\_eq.size()):

**if** abs(triangled\_eq.matrix[j][i]) > abs(triangled\_eq.matrix[max\_row\_idx][i]):

max\_row\_idx = j

# Перестановка строк

**if** max\_row\_idx != i:

triangled\_eq.matrix.swap\_rows(i, max\_row\_idx)

triangled\_eq.b\_vector.swap\_elems(i, max\_row\_idx)

perm\_count += 1

# Убираем x[i][i] из каждой строки ниже i

**for** j **in** range(i + 1, triangled\_eq.size()):

kf: float = - triangled\_eq.matrix[j][i] / triangled\_eq.matrix[i][i]

adding\_vec = triangled\_eq.matrix[i] \* kf

triangled\_eq.matrix[j] += adding\_vec

adding\_b = kf \* triangled\_eq.b\_vector[i]

triangled\_eq.b\_vector[j] += adding\_b

self.triangled: Equation = triangled\_eq

self.set\_det(perm\_count)

# Считает детерминант треугольной матрицы

**def** set\_det(self, k: int) -> **None**:

self.det: float = 1

**for** i **in** range(self.size()):

self.det \*= self.triangled.matrix[i][i]

self.det \*= ((-1) \*\* k)

# Решает уравнение, считает вектор ответов

**def** calc\_answers(self) -> **None**:

**if** **not** self.matrix.is\_correct():

**return**

self.triangle()

**if** self.det == 0:

**return**

answers: Vector = Vector(self.size())

**for** i **in** range(self.size() - 1, -1, -1):

answer = self.triangled.b\_vector[i]

**for** j **in** range(self.size() - 1, i, -1):

answer -= self.triangled.matrix[i][j] \* answers[j]

answers[i] = answer / self.triangled.matrix[i][i]

self.answers = answers

**def** solve(self) -> **None**:

self.calc\_answers()

**if** self.answers **is** **None**:

print('Invalid matrix! No answers or any answer')