Министерство науки и высшего образования Российской Федерации

Новосибирский государственный технический университет

Кафедра теоретической и прикладной информатики

Планирование и анализ эксперимента

Лабораторная работа №3

Факультет:	ФПМИ
Группа:	ПМ-63
Студенты:	Кожекин М.В.
	Майер В.А.
	Назарова Т.А.
	Утюганов Д.С.
Вариант:	9(1)

Новосибирск

2020

1. Цель работы

Изучить алгоритмы, используемые при построении дискретных оптимальных планов эксперимента.

2. Задание

1. Изучить алгоритмы построения дискретных оптимальных планов.

2. Разработать программу построения дискретных оптимальных планов эксперимента, реализующую заданный алгоритм.

3. Для числа наблюдений 20, 25, 30, 35, 40 построить оптимальные планы на каждой из сеток, указанных в варианте задания. Выбрать лучшие дискретные планы для заданного числа наблюдений.

4. Оформить отчет, включающий в себя постановку задачи, результаты проведенных в п. 3 исследований, текст программы.

5. Защитить лабораторную работу.

3. Анализ

Задана двухфакторная модель на квадрате со сторонами [-1, 1].

Дискретное множество $\tilde{X}$ - сетки 10x10 и 20x20. Строить D-оптимальные планы. Алгоритм Фёдорова. Повторные наблюдения допускаются.

$$y = \Theta_0 + \Theta_1 \cdot x_1 + \Theta_2 \cdot x_2 + \Theta_3 \cdot x_1 \cdot x_2 + \Theta_4 \cdot x_1^2 + \Theta_5 \cdot x_2^2$$

Этапы **алгоритма Фёдорова** синтеза непрерывного оптимального плана:

1. Выбирается невырожденный начальный план ε_N^0 и малая константа $\delta > 0$, s = 0.

2. Выбирается пара точек: x_j^s, принадлежащая плану ε_N^s, и x^s, не принадлежащая плану, по правилу

$$\left(x_j^s, x^s\right) = arg\left(\max_{x_j \in \varepsilon_N^s} \max_{x \in \tilde{X}} \Delta(x_j, x)\right)$$

где

$$\Delta(x_j, x) = \frac{1}{N}\left[d(x, \varepsilon_N) - d(x_j, \varepsilon_N)\right]$$

$$-\frac{1}{N^2}\left[d(x, \varepsilon_N) d(x_j, \varepsilon_N) - d^2(x_j, \varepsilon_N)\right]$$

,

$$d(x, \varepsilon) = f^T(x) M^{-1} f(x), \quad d(x, x_j, \varepsilon) = f^T(x) M^{-1} f(x_j)$$

3. Величина $\Delta(x_j, x)$ сравнивается с δ. Если $\Delta(x_j, x) \leq \delta$, то вычисления прекращаются, в противном случае осуществляется переход на шаг 4.

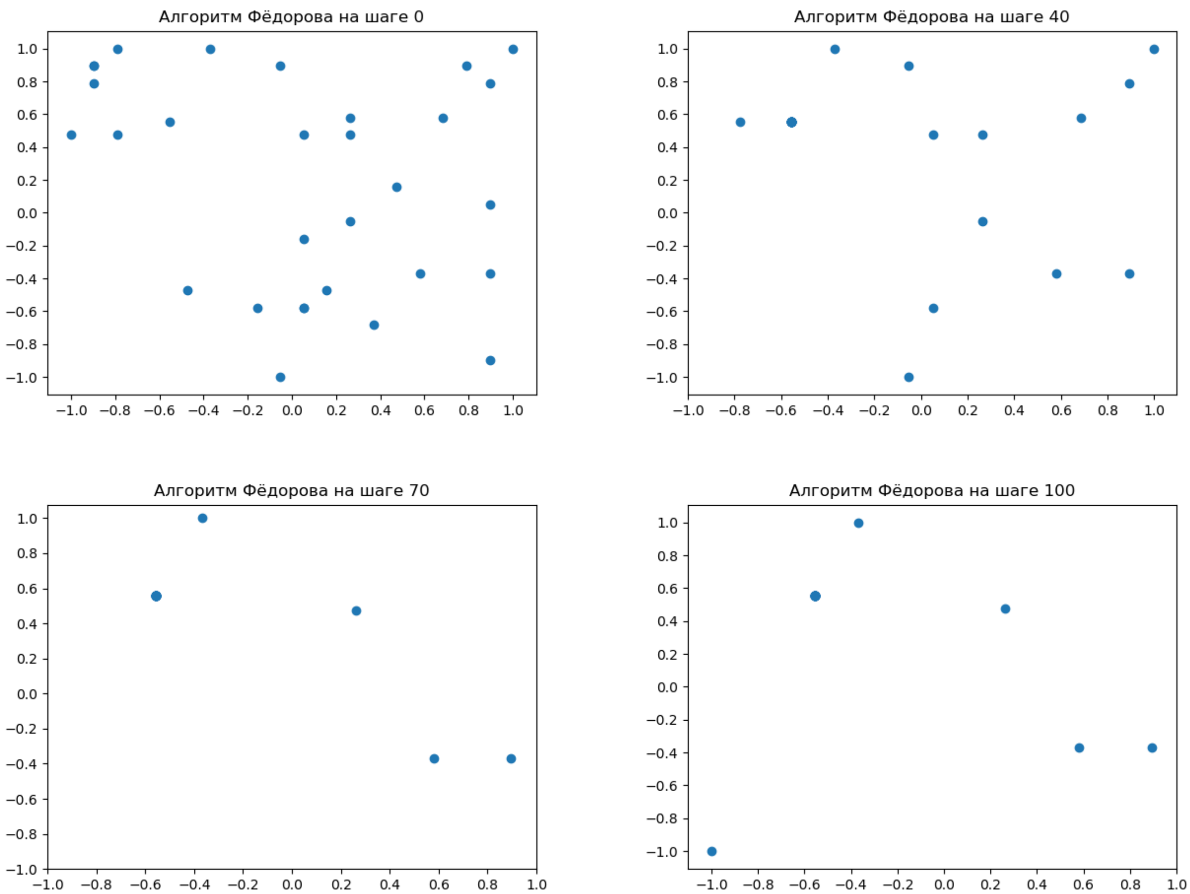
4. Точка x_j заменяется в плане на точку x. В результате получается новый план ε_N^{s+1}. Далее s заменяется s+1 и осуществляется переход на шаг 2.

Оптимизационная процедура, выполнимая на шаге 2, может оказаться слишком трудоёмкой в ввычислительном плане, поэтому на практике ограничиваются поиском первой пары точек $\left(x_j^s, x^s\right)$, для которой выполняется условие $\Delta\left(x_j^s, x^s\right) \geq \delta$. После чего выполняется шаг 4.

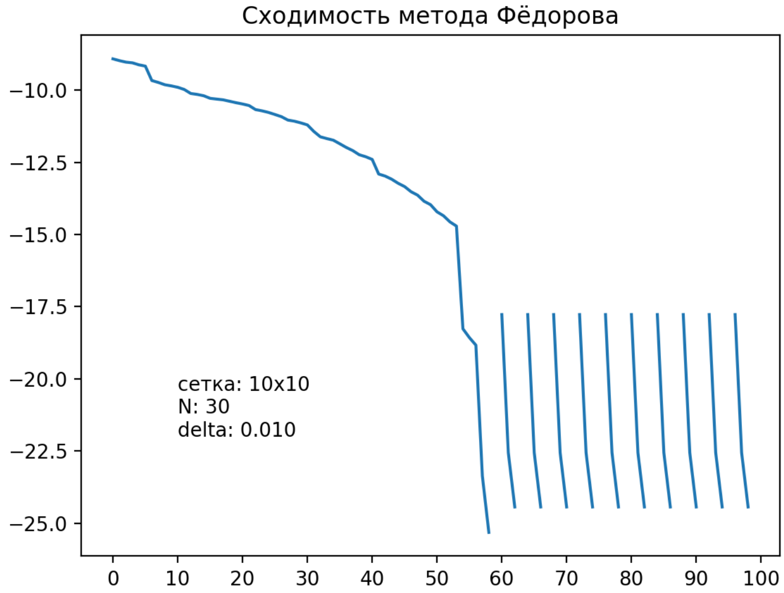
4. Исследования работы алгоритма

4.1. Визуализация сходимости метода

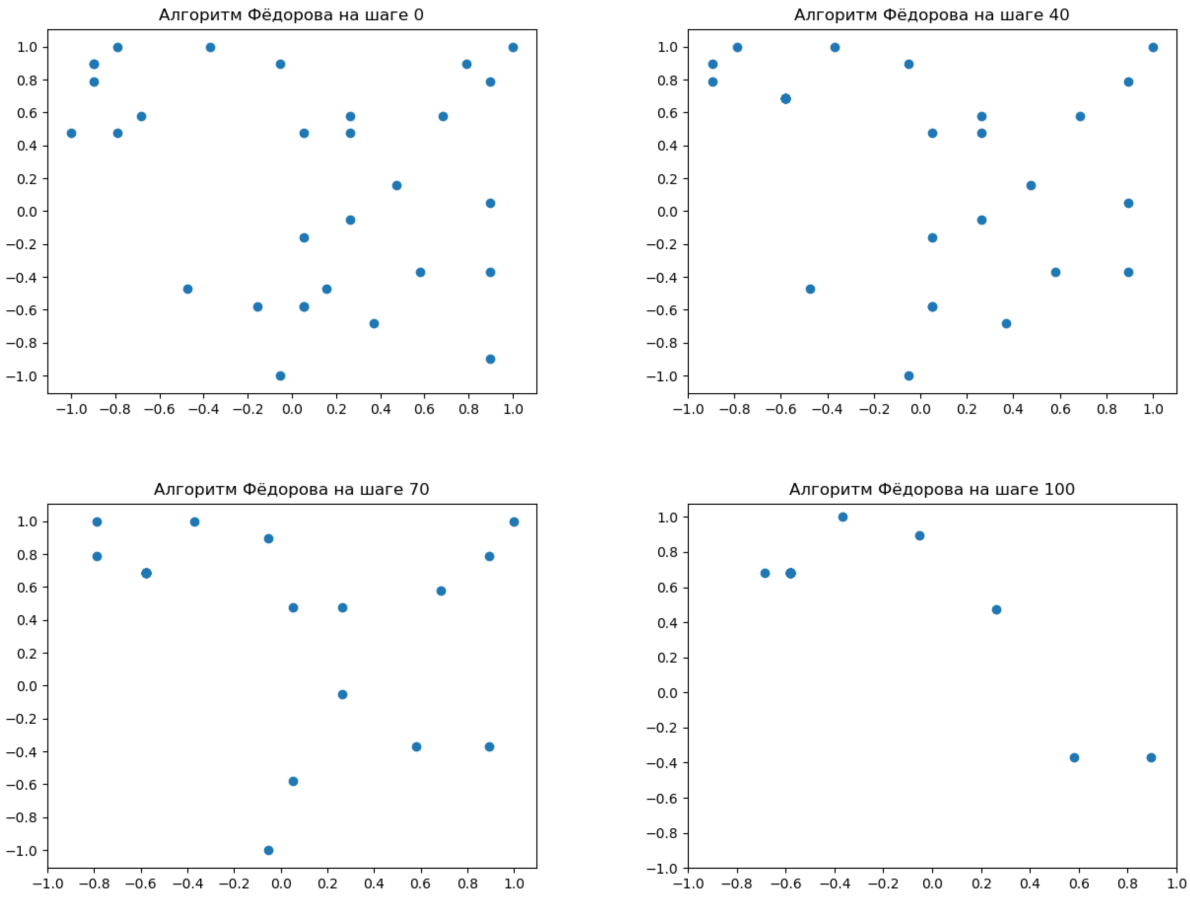
Сходимость алгоритма Фёдорова при N = 30, $\delta = 0.01$ на сетке 10x10



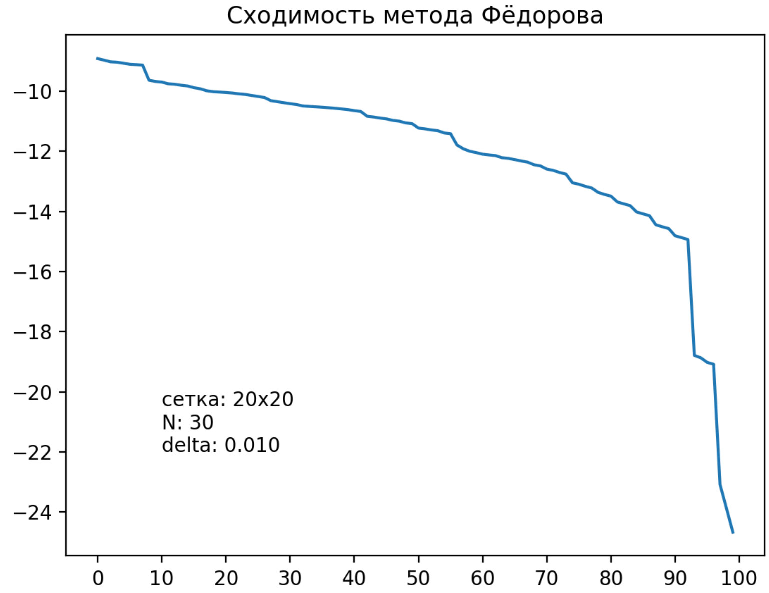
Исходя из графика можно понять, что из-за размера сетки алгоритм расходится на 55-ом шаге.



Сходимость алгоритма Фёдорова при N = 30, $\delta = 0.01$ на сетке 20x20

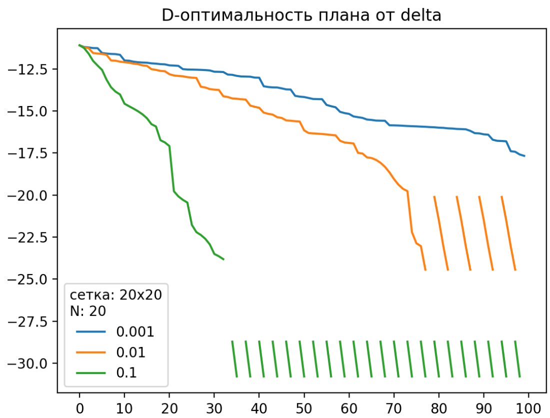


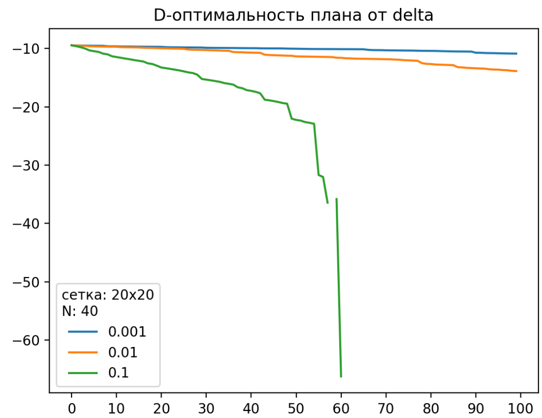
Исходя из графика можно понять, что критерий D-оптимальности уменьшается, значит, наш алгоритм всё ещё сходится.



4.2. Влияние шага метода

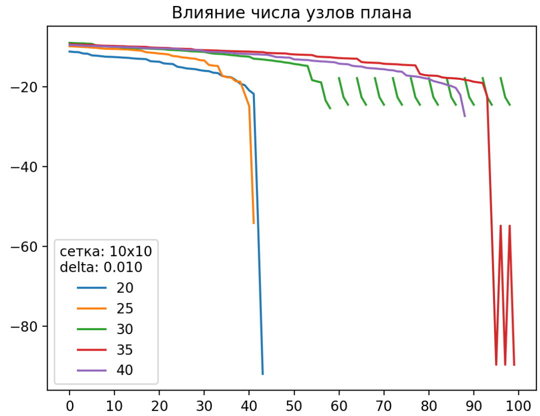
Как видно из графиков при уменьшении δ метод сходится медленнее. При увеличении данного параметра метод может разойтись.

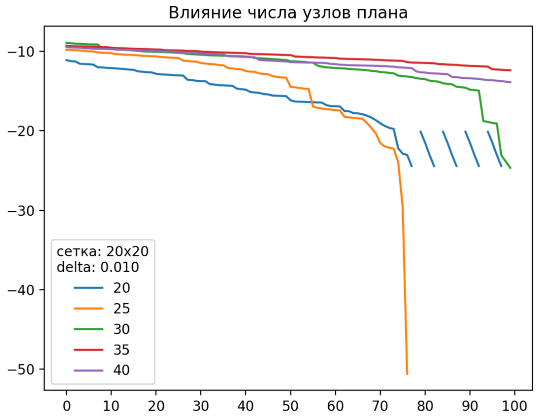




4.3. Влияние числа узлов плана

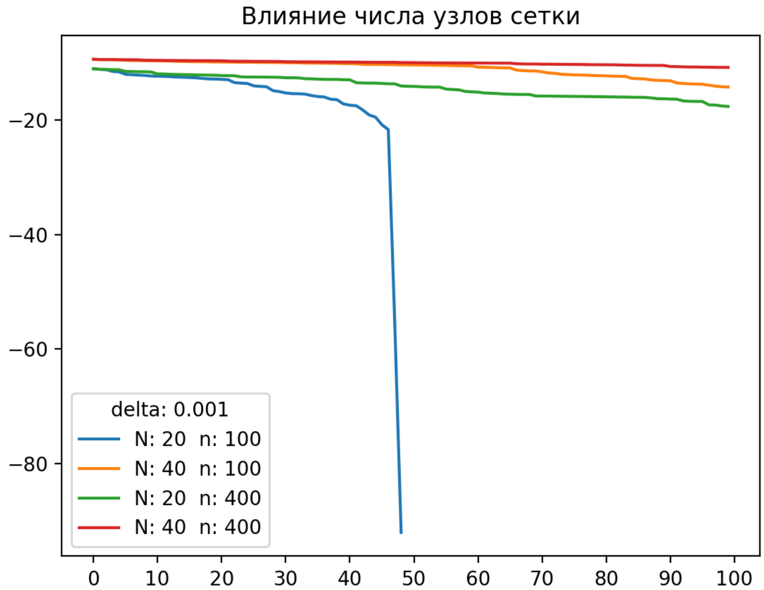
При малом числе точек плана метод становится неустойчивым. Особенно это заметно если множество $\tilde{X}$ также имеет малое число точек





4.4. Влияние числа узлов сетки

При увеличении числа точек сетки метод становится стабильнее, но медленнее сходится. При малых N и n метод расходится.



5. Исходный код программы

lab3.py

```
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import numpy as np
from numpy.linalg import det, inv, norm

k = 2        # число переменных 2: (x,y)
m = 6        # число параметров a + b*x + c*y + d*x*y + e*x^2 + f*y^2
MAX_ITER = 100

```

```
def f(theta, x):
    return  theta[0] + \
            theta[1]*x[0] + \
            theta[2]*x[1] + \
            theta[3]*x[0]*x[1] + \
            theta[4]*x[0]**2 + \
            theta[5]*x[1]**2

def f_vector(x):
    return np.array([
        [1],
        [x[0]],
        [x[1]],
        [x[0]*x[1]],
        [x[0]**2],
        [x[1]**2]
    ])

def f_vector_T(x):
    return np.array([
        1,
        x[0],
        x[1],
        x[0]*x[1],
        x[0]**2,
        x[1]**2
    ])


#--------------------------------------------------------------------------------
#--------------------------------------------------------------------------------
#--------------------------------------------------------------------------------
class Lab3():
    '''
    Класс для 3 лабораторной работы
    Для стандартизации все критерии ищут минимальное значение
    '''
    def __init__(self, N, width, delta):
        ''' Выделение памяти под массивы '''
        n = width**2
        self.x_grid = np.ndarray((n, k))
        self.x_plan = np.ndarray((N, k))
        self.p = np.full(N, 1/N)
        self.M = np.ndarray((m, m))
        self.D = np.ndarray((m, m))
        self.width = width
        self.n = n
        self.delta = delta
        self.max_iter = MAX_ITER
        self.N = N

    def Fedorov_algorithm(self, do_visualisation = False):
        '''
        Алгоритм Фёдорова синтеза дискретного
        оптимального плана эксперимента
        '''
        self.generate_initial_guess()
        do_calc = True
        s = 0
        result = np.zeros(self.max_iter)
```

```
        while do_calc == True and s < self.max_iter:

            self.build_matrix_M()
            self.build_matrix_D()

            max_delta, i, j = self.replace_two_points()
            do_calc = not (max_delta < self.delta)
            result[s] = self.calc_D()

            if s % 10 == 0 and do_visualisation:
                self.draw_plan_on_iteration(s)

            s += 1
            print(s, i, j, max_delta, self.calc_D())

        if do_visualisation:
            self.draw_plan_on_iteration(s)
        return result

    def draw_plan_on_iteration(self, s):
        t = np.linspace(-1, 1, 11)
        plt.title('Алгоритм Фёдорова на шаге ' + str(s))
        plt.scatter([self.x_plan[i][0] for i in range(len(self.x_plan))],[self.x_plan[i][1] for i in range(len(self.x_plan))], )
        plt.xticks(t)
        plt.yticks(t)
        plt.savefig('pics/plan_Fedorov_{}_{}_{:.3f}_{}.png'.format(self.N, self.width, self.delta, s))
        plt.clf()

    def replace_two_points(self):
        ''' Замена точки из плана на внешнюю '''
        # для начала:
        max_delta = -9000
        # перебираем все точки плана и все точки сетки
        for i, int_point in enumerate(self.x_plan):
            for j, ext_point in enumerate(self.x_grid):
                delta = self.Delta(int_point, ext_point)[0]
                # delta = abs(self.Delta(int_point, ext_point))
                # выбрали пару точек получше
                if delta > self.delta:
                    max_delta = delta
                    self.x_plan[i] = self.x_grid[j]
                    return max_delta, i, j
        return max_delta, -1, -1

    def generate_initial_guess(self):
        ''' Задаём начальное приближение '''
        # создаём сетку
        self.t = np.linspace(-1, 1, self.width)
        i = 0
        for x1 in self.t:
            for x2 in self.t:
                self.x_grid[i] = np.array([x1, x2])
                i+=1

        # случайно выбираем точки плана и сохраняем
        # for i in range(self.N):
        #     self.x_plan[i] = self.x_grid[np.random.choice(n)]
```

```
        # np.savetxt('report/plan'+str(self.N)+'.txt', self.x_plan)

        # или же загружаем
        self.x_plan = np.loadtxt('report/plan'+str(self.N)+'.txt', dtype=np.float)

    def build_matrix_M(self):
        ''' Построение информационной матрицы M '''
        self.M = np.zeros((m, m))
        for i in range(self.N):
            self.M += self.p[i] * f_vector(self.x_plan[i]) * f_vector_T(self.x_plan[i])

    def build_matrix_D(self):
        ''' Построение дисперсионной матрицы D '''
        self.D = inv(self.M)

    def calc_D(self):
        '''
        Критерий D — оптимальности. (D — determinant)
        Эллипсоид рассеивания имеет минимальный объём
        '''
        return np.log(det(self.M))

    def Delta(self, x, x_j):
        N = float(self.N)
        return (self.d(x) - self.d(x_j)) / N - (self.d(x) * self.d(x_j) - self.d_2(x,x_j)**2) / N**2

    def d(self, x):
        return f_vector_T(x) @ self.D @ f_vector(x)

    def d_2(self, x, x_j):
        return f_vector_T(x) @ self.D @ f_vector(x_j)

#------------------------------------------------------------------------------
#------------------------------------------------------------------------------
#------------------------------------------------------------------------------
t = np.linspace(0, MAX_ITER, 11)

def perform_experiment(N, width, delta):
    l3 = Lab3(N, width, delta)
    return l3.Fedorov_algorithm()

def research_delta():
    ''' Исследование скорости сходимости и устойчивости от delta '''
    width = 20
    for N in [20, 40]:
        for delta in [0.001, 0.01, 0.1]:
            y = perform_experiment(N, width, delta)
            plt.plot(y, label=str(delta))

        plt.title('Dоптимальность— плана от delta')
        plt.legend(title='сетка: {}x{}\nN: {}'.format(width, width, N))
        plt.xticks(t)
        plt.savefig('pics/research_delta_{}x{}_{}.png'.format(width, width, N), dpi=200)
        plt.clf()

def research_N():
```

```
    ''' Исследование скорости сходимости и устойчивости от N '''
    delta = 0.01
    for width in [10, 20]:
        for N in [20, 25, 30, 35, 40]:
            y = perform_experiment(N, width, delta)
            plt.plot(y, label=str(N))
        plt.title('Влияние числа узлов плана')
        plt.legend(title='сетка: {}x{}\ndelta: {:.3f}'.format(width, width, delta))
        plt.xticks(t)
        plt.savefig('pics/research_N_{}x{}.png'.format(width, width), dpi=200)
        plt.clf()

def research_width():
    ''' Исследование скорости сходимости и устойчивости от числа узлов сетки '''
    delta = 0.001
    for width in [10, 20]:
        for N in [20, 40]:
            y = perform_experiment(N, width, delta)
            plt.plot(y, label='N: {}  n: {}'.format(N, width**2))

    plt.title('Влияние числа узлов сетки')
    plt.legend(title='delta: {:.3f}'.format(delta))
    plt.xticks(t)
    plt.savefig('pics/research_width.png', dpi=200)
    plt.clf()

def show_convergence(N, width, delta):
    l3 = Lab3(N, width, delta)
    y = l3.Fedorov_algorithm(True)
    plt.plot(y)
    plt.title('Сходимость метода Фёдорова')
    plt.text(10, -22, 'сетка: {}x{}\nN: {}\ndelta: {:.3f}'.format(width, width, N, delta))
    plt.xticks(t)
    plt.savefig('pics/convergence_Fedorov_{}_{}_{:.3f}.png'.format(N, width, delta), dpi=200)
    plt.clf()

#------------------------------------------------------------------------------
#------------------------------------------------------------------------------
#------------------------------------------------------------------------------
research_delta()
research_N()
research_width()
show_convergence(30, 10, 0.01)
show_convergence(30, 20, 0.01)
```