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

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

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

,

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

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.

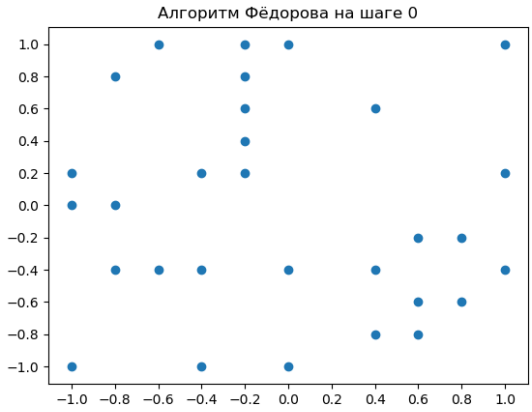
Оптимизационная процедура, выполнимая на шаге 2, может оказаться слишком трудоёмкой в ввычислительном плане, поэтому на практике ограничиваются поиском первой пары точек $\left(x_j^s, x^s\right)$, для которой выполняется условие $\Delta\left(x_j^s, x^s\right) \geq \delta$. После чего выполняется шаг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

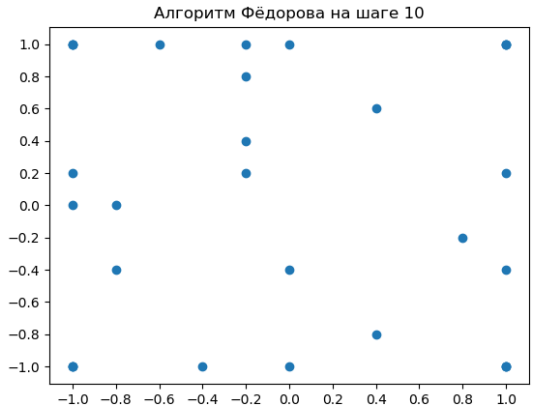
4. Исследования работы алгоритма

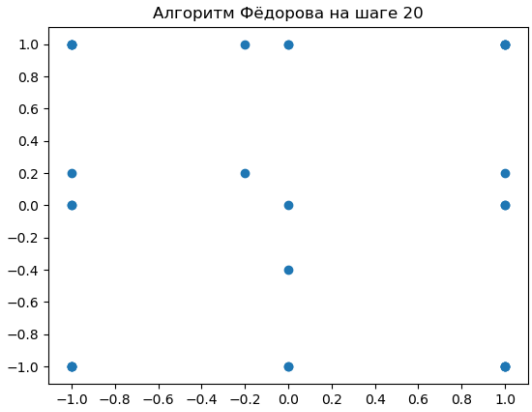
Сходимость алгоритма Фёдорова при N = 30, $\delta = 0.01$ на сетке 11x11, 21x21.

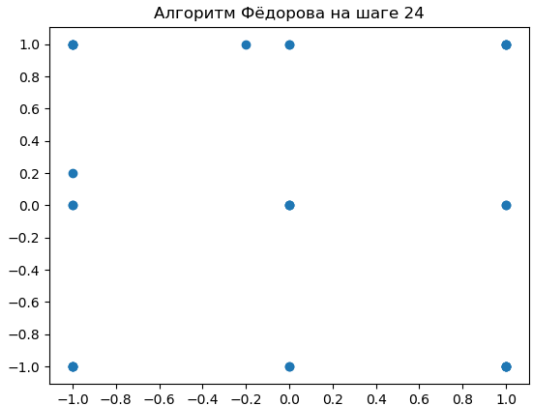
Как видно из 4-х изображений работы метода и графика $\log(|M^{-1}(\varepsilon)|)$ на каждой итерации можно понять, что метод на сетке 11x11 сходится за 24 итерации, а на сетке 21x21 узел за 29 шагов.

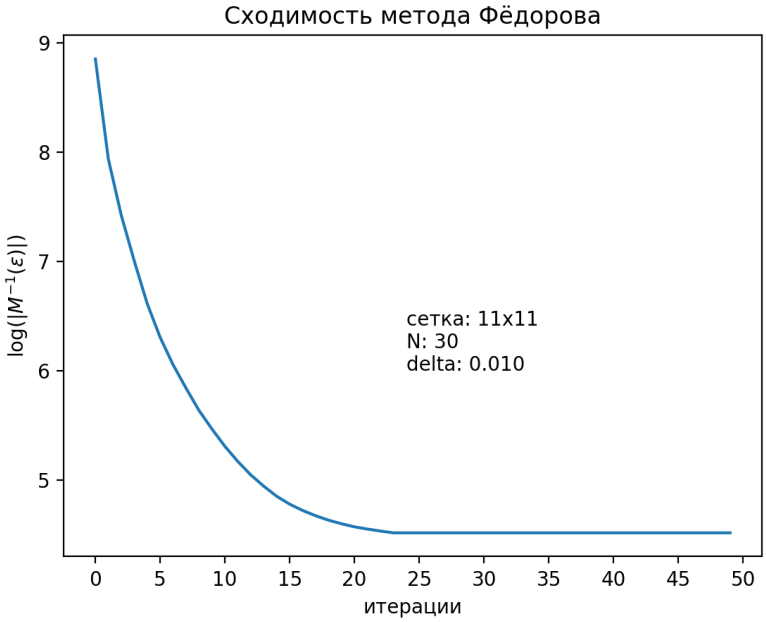
4.1. Сетка 11x11



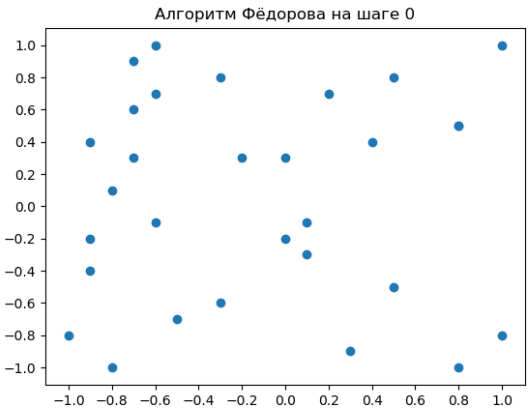


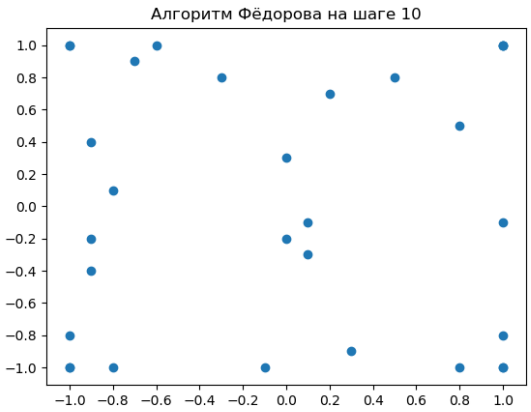


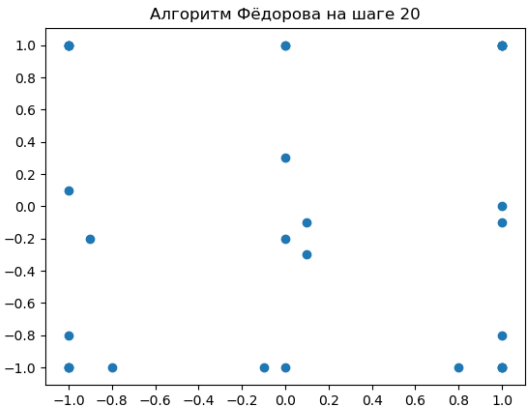


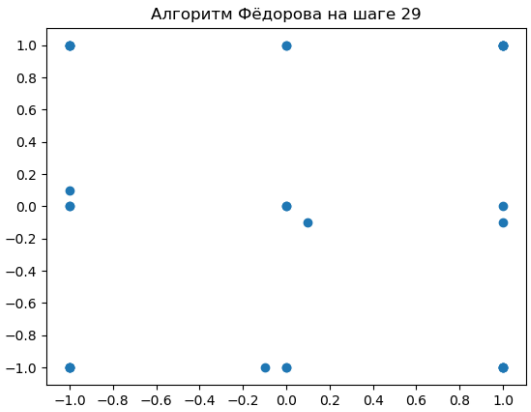


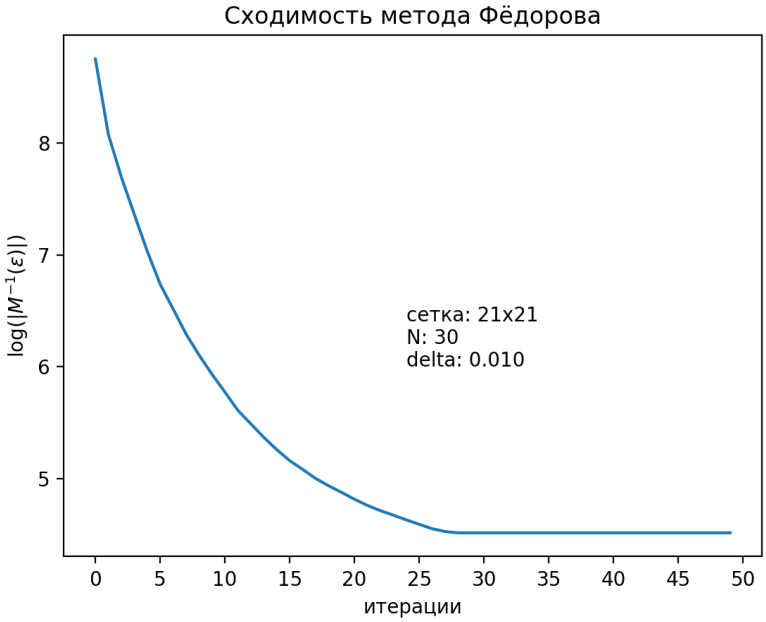
4.2. Сетка 21x21







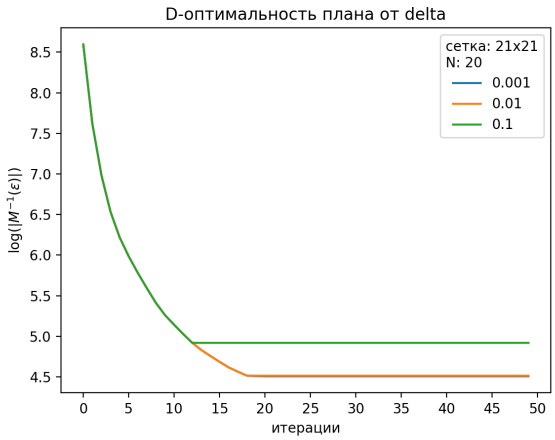


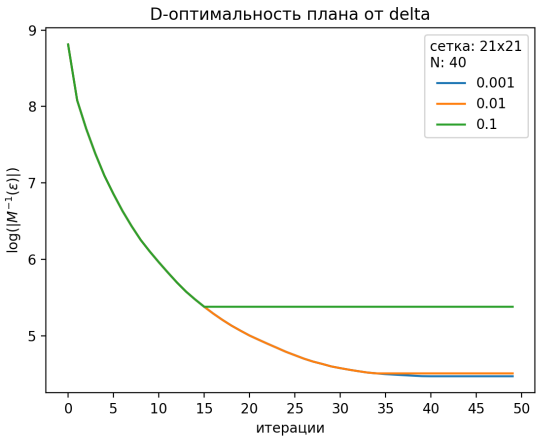


4.3. Влияние шага метода

Как видно из графиков при изменении δ в [0.1, 0.01, 0.001] метод первые 12 итераций (при сетке 11x11 и 15 на сетке 21x21) сходится одинаково быстро. Затем алгоритм с меньшим шагом находет более оптимальное решение.

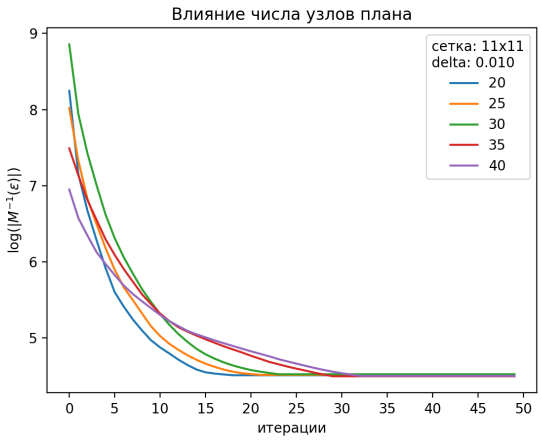
Особенно это заметно при большем числе узлов сетки.

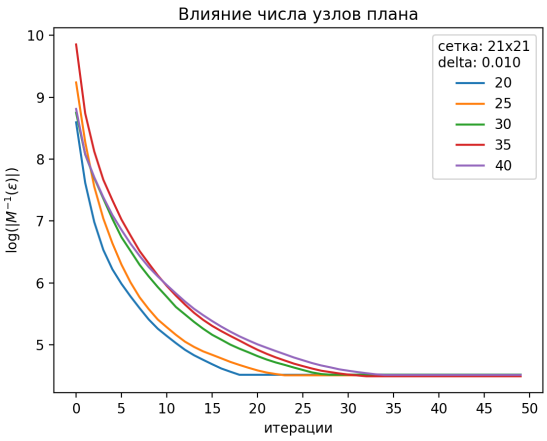




4.4. Влияние числа узлов плана

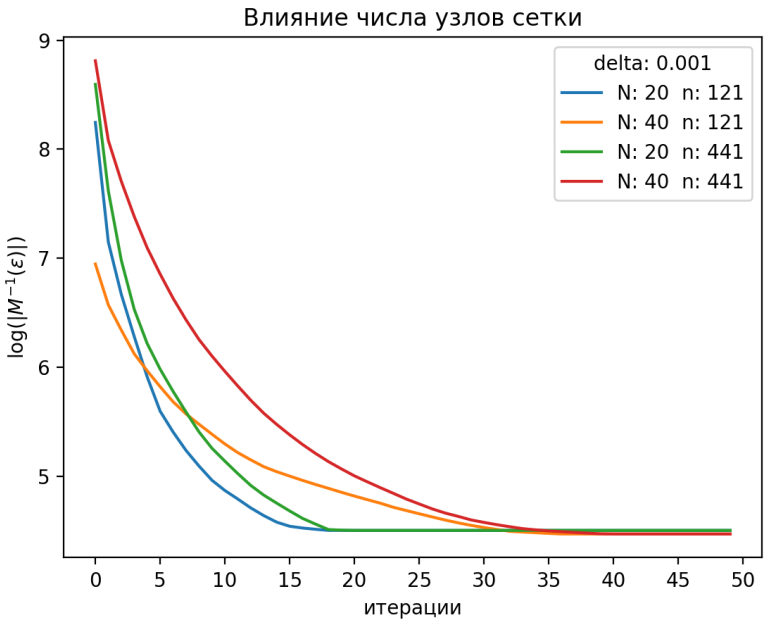
При малом числе точек плана алгоритм сходится сначала медленнее, а потом быстрее. Однако это не так значительно для решения задачи





4.5. Влияние числа узлов сетки

При увеличиении числа точек сетки методу требуется больше итераций, однако, результат остаётся примерно одинаковым. 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 = 50
```

```
9
10 def f(theta, x):
11     return  theta[0] + \
12             theta[1]*x[0] + \
13             theta[2]*x[1] + \
14             theta[3]*x[0]*x[1] + \
15             theta[4]*x[0]**2 + \
16             theta[5]*x[1]**2
17
18 def f_vector(x):
19     return np.array([
20         [1],
21         [x[0]],
22         [x[1]],
23         [x[0]*x[1]],
24         [x[0]**2],
25         [x[1]**2]
26     ])
27
28 def f_vector_T(x):
29     return np.array([
30         1,
31         x[0],
32         x[1],
33         x[0]*x[1],
34         x[0]**2,
35         x[1]**2
36     ])
37
38
39 #-----------------------------------------------------------------------------
40 #-----------------------------------------------------------------------------
41 #-----------------------------------------------------------------------------
42 class Lab3():
43     '''
44     Класс для 3 лабораторной работы
45     Для стандартизации все критерии ищут минимальное значение
46     '''
47     def __init__(self, N, width, delta):
48         ''' Выделение памяти под массивы '''
49         n = width**2
50         self.x_grid = np.ndarray((n, k))
51         self.x_plan = np.ndarray((N, k))
52         self.p = np.full(N, 1/N)
53         self.M = np.ndarray((m, m))
54         self.D = np.ndarray((m, m))
55         self.width = width
56         self.n = n
57         self.delta = delta
58         self.max_iter = MAX_ITER
59         self.N = N
60
61     def Fedorov_algorithm(self, do_visualisation = False):
62         '''
63         Алгоритм Фёдорова синтеза дискретного
64         оптимального плана эксперимента
65         '''
66         self.generate_initial_guess()
67         do_calc = True
68         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()
            D = self.calc_D()
            max_delta, i, j = self.find_two_points()
            self.x_plan[i] = self.x_grid[j]
            do_calc = not (max_delta < self.delta)
            result[s] = D
            if s % 10 == 0 and do_visualisation:
                self.draw_plan_on_iteration(s)
            s += 1
            print(s, i, j, max_delta, D)

        if do_calc == False:
            for i in range(s, self.max_iter):
                result[i] = result[s-1]

        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 find_two_points(self):
        ''' Поиск точки плана и сетки для их замены '''
        # для начала:
        indicies = np.ndarray((self.N, 2), dtype = np.int64) # Пары i,j
        max_deltas = np.full(self.N, -9000.0)

        # 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]
                # выбрали пару точек получше
                if delta > max_deltas[i]:
                    max_deltas[i] = delta
                    indicies[i][0] = i
                    indicies[i][1] = j

        max_delta_res = -9000.0
        I = 0
        J = 0
        for i, max_delta in enumerate(max_deltas):
            if max_delta > max_delta_res:
                max_delta_res = max_delta
                I = indicies[i][0]
                J = indicies[i][1]

        return max_delta_res, I, J
```

```
127
128     def generate_initial_guess(self):
129         ''' Задаём начальное приближение '''
130         # создаём сетку
131         self.t = np.linspace(-1, 1, self.width)
132         i = 0
133         for x1 in self.t:
134             for x2 in self.t:
135                 self.x_grid[i] = np.array([x1, x2])
136                 i+=1
137
138         # случайно выбираем точки плана и сохраняем
139         # for i in range(self.N):
140         #     self.x_plan[i] = self.x_grid[np.random.choice(self.n)]
141         # np.savetxt('plans/plan_{}x{}_{}.txt'.format(self.width, self.width,
    self.N), self.x_plan)
142
143         # или же загружаем
144         self.x_plan = np.loadtxt('plans/plan_{}x{}_{}.txt'.format(self.width,
    self.width, self.N), dtype=np.float)
145
146     def build_matrix_M(self):
147         ''' Построение информационной матрицы M '''
148         self.M = np.zeros((m, m))
149         for i in range(self.N):
150             self.M += (self.p[i] * f_vector(self.x_plan[i]) * f_vector_T(self.
    x_plan[i]))
151
152     def build_matrix_D(self):
153         ''' Построение дисперсионной матрицы D '''
154         self.D = inv(self.M)
155
156     def calc_D(self):
157         '''
158         Критерий D - оптимальности. (D - determinant)
159         Эллипсоид рассеивания имеет минимальный объём
160         '''
161         return np.log(det(self.D))
162
163     def Delta(self, x_j, x):
164         N = float(self.N)
165         return ((self.d(x) - self.d(x_j)) / N) - \
166             ((self.d(x) * self.d(x_j) - self.d_2(x,x_j)**2)/(N**2))
167
168     def d(self, x):
169         return f_vector_T(x) @ self.D @ f_vector(x)
170
171     def d_2(self, x, x_j):
172         return f_vector_T(x) @ self.D @ f_vector(x_j)
173
174 #-------------------------------------------------------------------------
175 #-------------------------------------------------------------------------
176 #-------------------------------------------------------------------------
177 t = np.linspace(0, MAX_ITER, 11)
178
179 def perform_experiment(N, width, delta):
180     l3 = Lab3(N, width, delta)
181     return l3.Fedorov_algorithm()
182
183 def research_delta():
```

```
    ''' Исследование скорости сходимости и устойчивости от delta '''
    print('Исследование скорости сходимости и устойчивости от delta')
    width = 21
    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.xlabel('итерации')
        plt.ylabel(r'$\log(\left| M^{-1}(\varepsilon) \right|)$')
        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 '''
    print('Исследование скорости сходимости и устойчивости от N')
    delta = 0.01
    for width in [11, 21]:
        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.xlabel('итерации')
        plt.ylabel(r'$\log(\left| M^{-1}(\varepsilon) \right|)$')
        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 '''
    print('Исследование скорости сходимости и устойчивости от числа узлов сетки')
    delta = 0.001
    for width in [11, 21]:
        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.xlabel('итерации')
    plt.ylabel(r'$\log(\left| M^{-1}(\varepsilon) \right|)$')
    plt.savefig('pics/research_width.png', dpi=200)
    plt.clf()

def show_convergence(N, width, delta):
    print(N, width, delta)
    l3 = Lab3(N, width, delta)
    y = l3.Fedorov_algorithm(True)
    plt.plot(y)
    plt.title('Сходимость метода Фёдорова')
    plt.text(24, 6, 'сетка: {}x{}\nN: {}\ndelta: {:.3f}'.format(width, width, N,
    delta))
    plt.xticks(t)
```

```
    plt.xlabel('итерации')
    plt.ylabel(r'$\log(\left| M^{-1}(\varepsilon) \right|)$')
    plt.savefig('pics/convergence_Fedorov_{}_{}_{:.3f}.png'.format(N, width,
    delta), dpi=200)
    plt.clf()

#-----------------------------------------------------------------------------
#-----------------------------------------------------------------------------
#-----------------------------------------------------------------------------
research_delta()
research_N()
research_width()
show_convergence(30, 11, 0.01)
show_convergence(30, 21, 0.01)
```