**Лабораторная 3**

Моисеев М32001, Муров М32011

## LU - разложение

Для матрицы

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

разложение вышло следующим:

```
    [[1. 0. 0.]
 L=  [2. 1. 0.]
     [3. 2. 1.]]
    [[ 1.  4.  7.]
 U=  [ 0. -3. -6.]
     [ 0.  0.  0.]]
```

## Поиск обратной матрицы:

Для невырожденной матрицы

```
[[1 2 3]
 [4 5 6]
 [7 8 0]]
```

Получаем такую обратную матрицу

```
 [[-1.77  0.88 -0.11]
 [ 1.55 -0.77  0.22]
 [-0.11  0.22 -0.11]]
```

Для проверки перемножим их, получаем единичную матрицу

В качестве итерационного метода был взят метод Зейделя

Для тестирования были выбраны матрицы с различными диагональными преобладаниями. Диагональное преобладание дает итерационному методу схождение к точному решению

```
k\n       5    10    15    20     25
 0   [    7    18    29    41    56]
 1   [  170   392   715  1111  1539]
 2   [    7  2158  3688  5922  8655]
 3   [    6  1225 11363 19824 33905]
```

При этом погрешность возрастает по похожему закону

```
[[0 0 0 0 0]
[1.42e-01 3.02e-01 3.35e-01 3.99e-01 4.76e-01]
[1.55 2.67 3.54 4.20 4.67]
[7.78 2.54e+01 3.54e+01 3.99e+01 4.80e+01]
```

Последнюю строчку с трудом можно назвать удовлетворительным результатом. Для нового алгоритма метода Зейделя максимум удалось получить решение для n = 500, 823 итерации, примерно 4 минуты.

Для функции Гильберта количество шагов и погрешность растет сильно быстрее

начало $n = 5$ ,шаг 5 dx - среднее отклонение от истинного значения.

```
steps=53
dx=0.639717522055063
steps=172
```

```
dx=0.968265045128112
steps=470
dx=0.9200272499839462
steps=439
dx=1.5035776564829042
steps=938
dx=1.3637934856130796
```

Чтобы проверить максимум возможностей самостоятельно написанных разреженных матриц, мы дополнительно создали тест почти диагональных матрицах: Практически во всех недиагональных элементах нули. Это позволяет проверить потолок возможностей алгоритма зейделя и создать матрицу размера $10^6$ Зависимость от времени

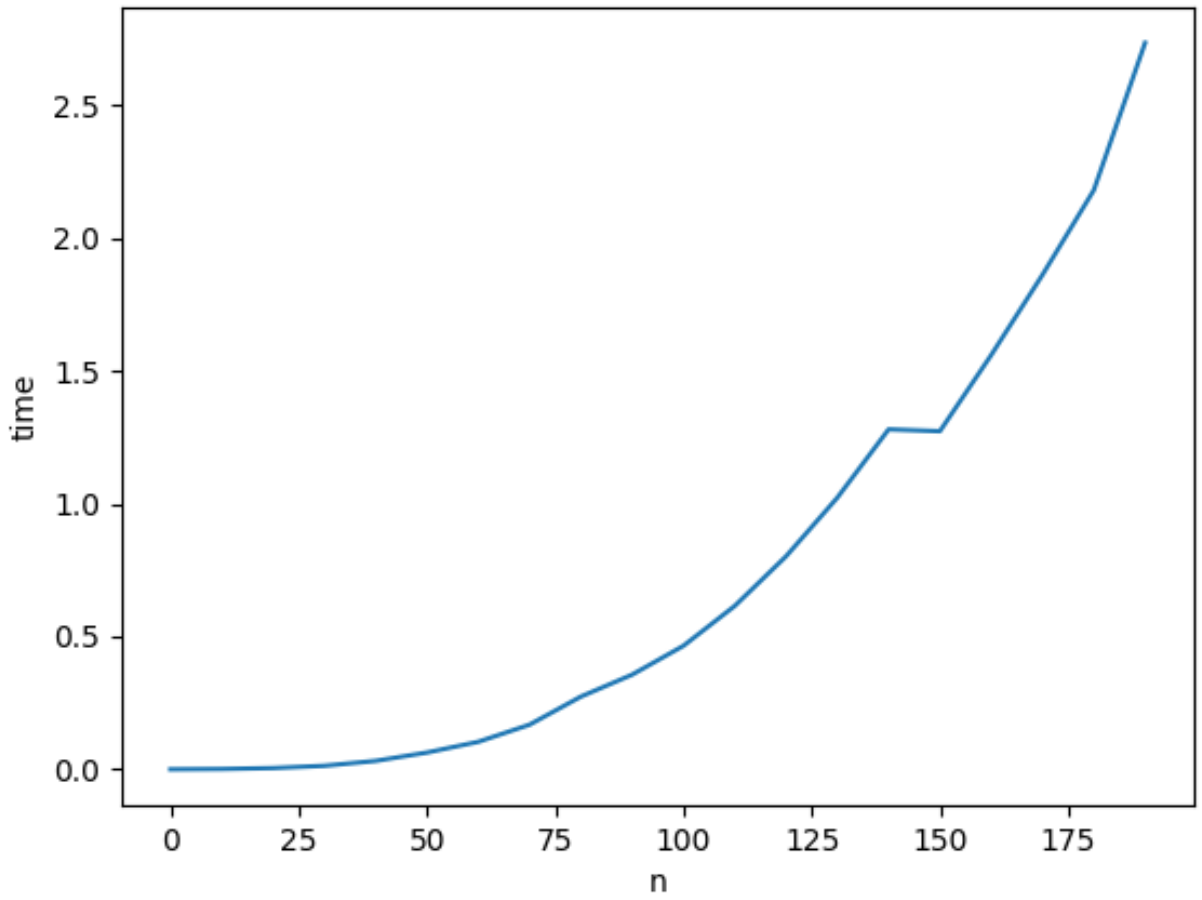


Figure 1: Время работы решения СЛАУ при помощи LU-разложения в зависимости от размера матрицы

Также было рассмотрено время работы относительно числа обусловленности и размера матрицы на функции с переменным значением диагонального преобладания. n - размерность матрицы k - параметр из функции, влияет на диагональное преобладание, чем больше k, тем оно меньше. цветом показано время работы.

Резкое падение времени работы метода Зейделя говорит о том, что вычисления не достигают приемлимой точности. Подсчет происходит быстро и неправильно. Также видно, что время работы сильно зависит от диагонального преобладания.

На матрице Гильберта зависимость выглядит следующим образом:

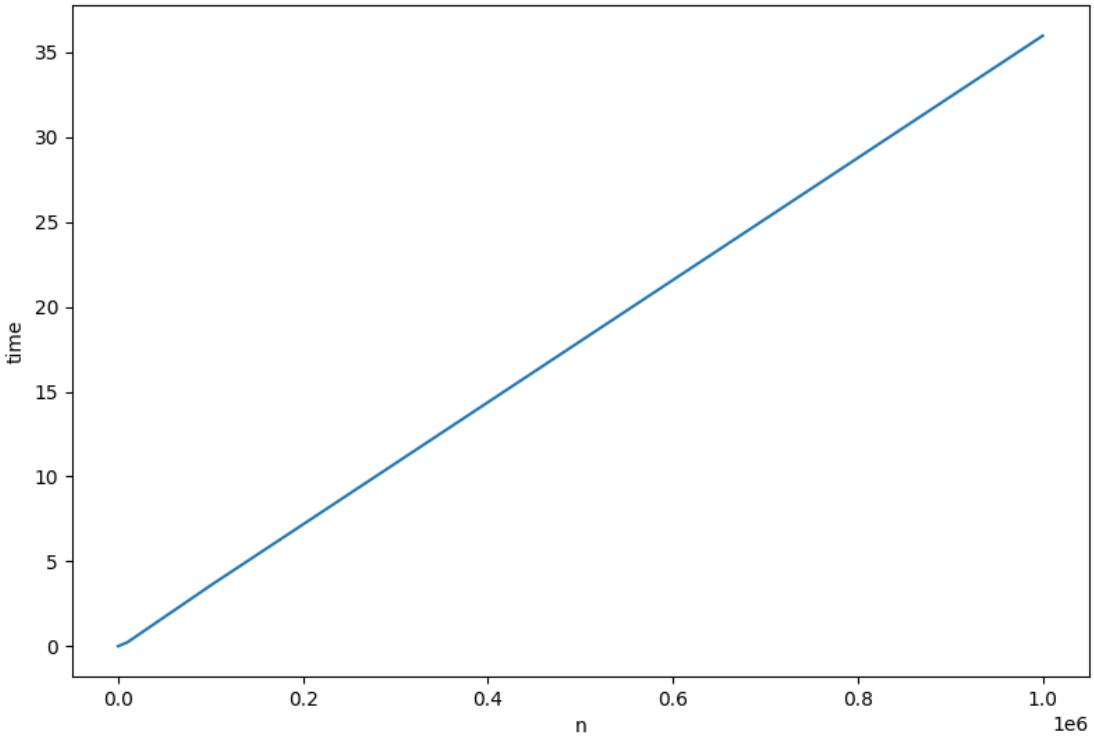


Figure 2: Время работы решения СЛАУ при помощи алгоритма Зейделя в зависимости от размера матрицы

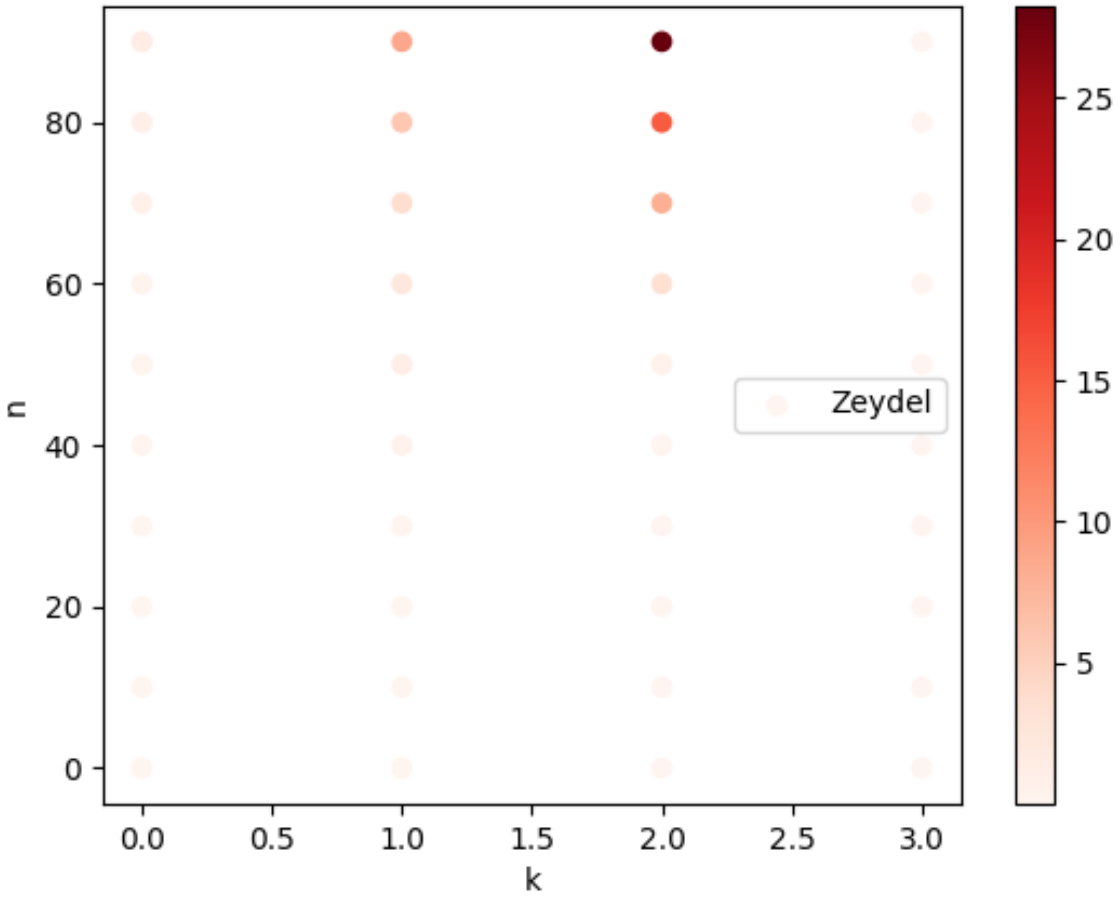


Figure 3: Время работы решения СЛАУ при помощи алгоритма Зейделя.

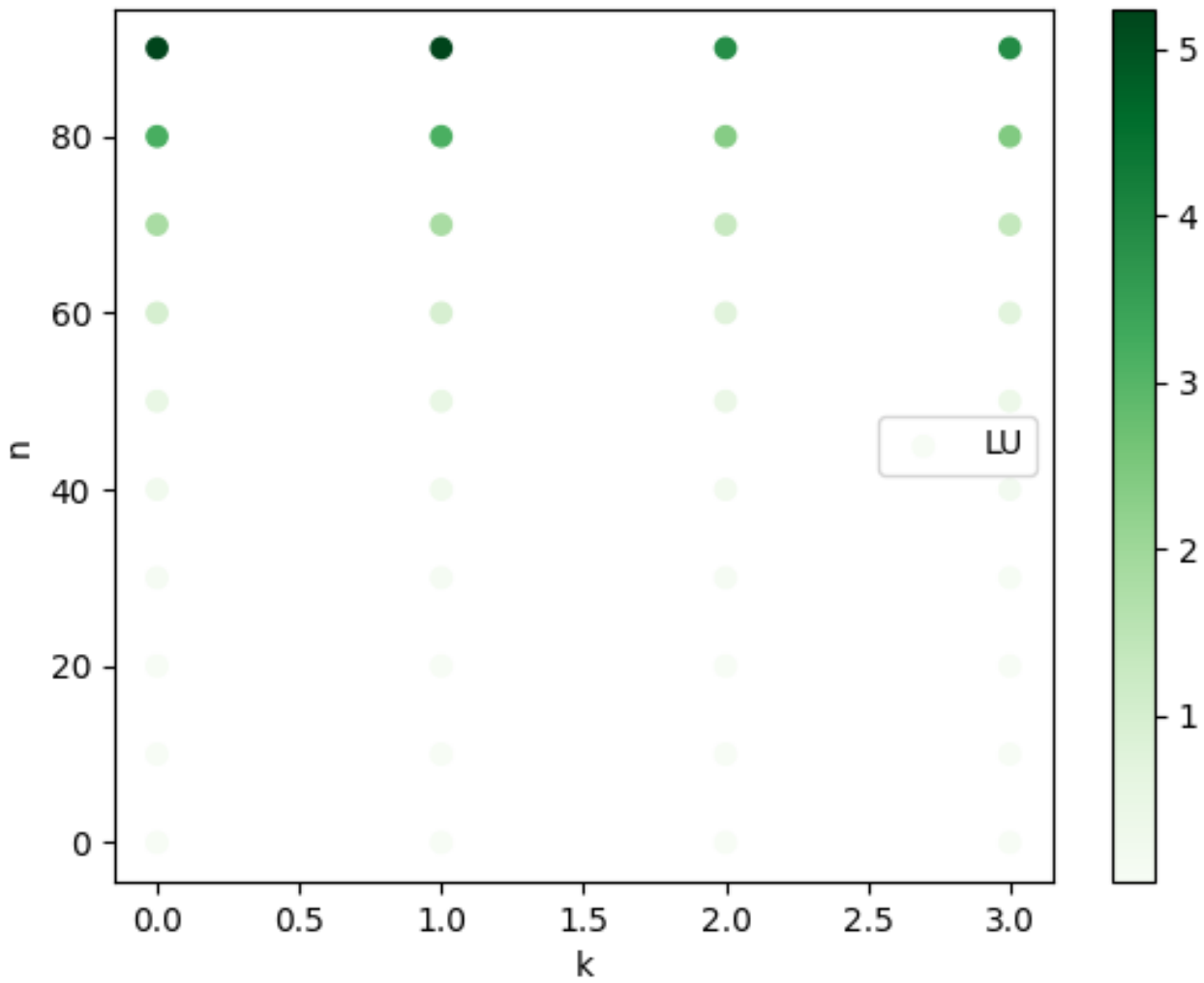


Figure 4: Время работы решения СЛАУ при помощи LU-разложения.

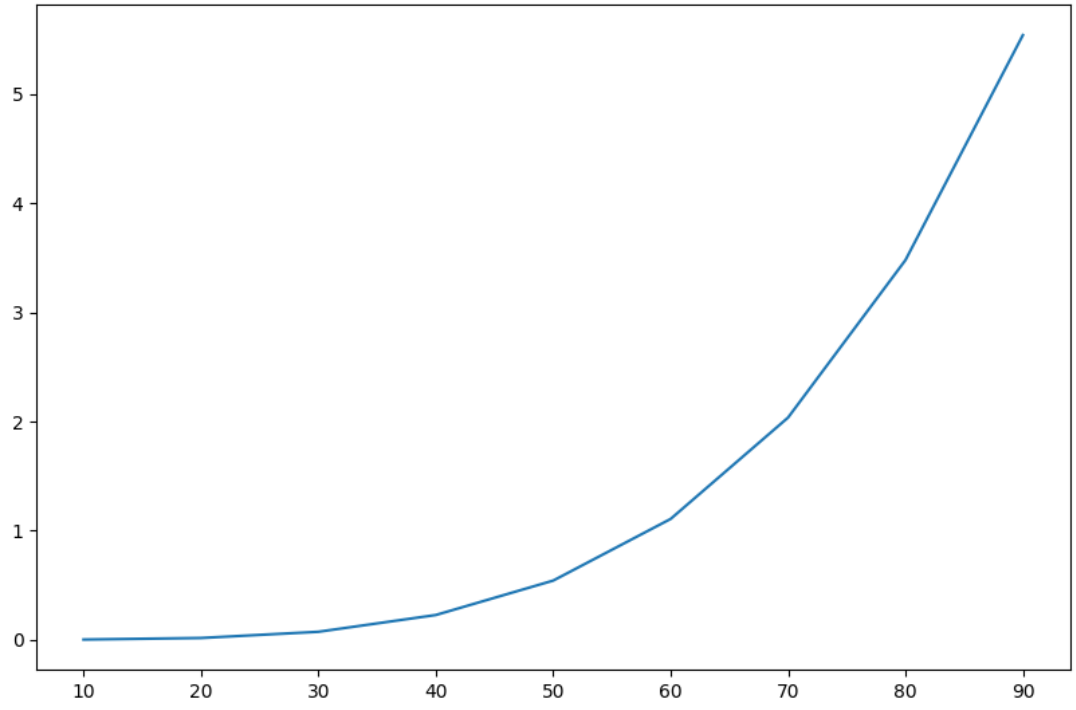


Figure 5: LU

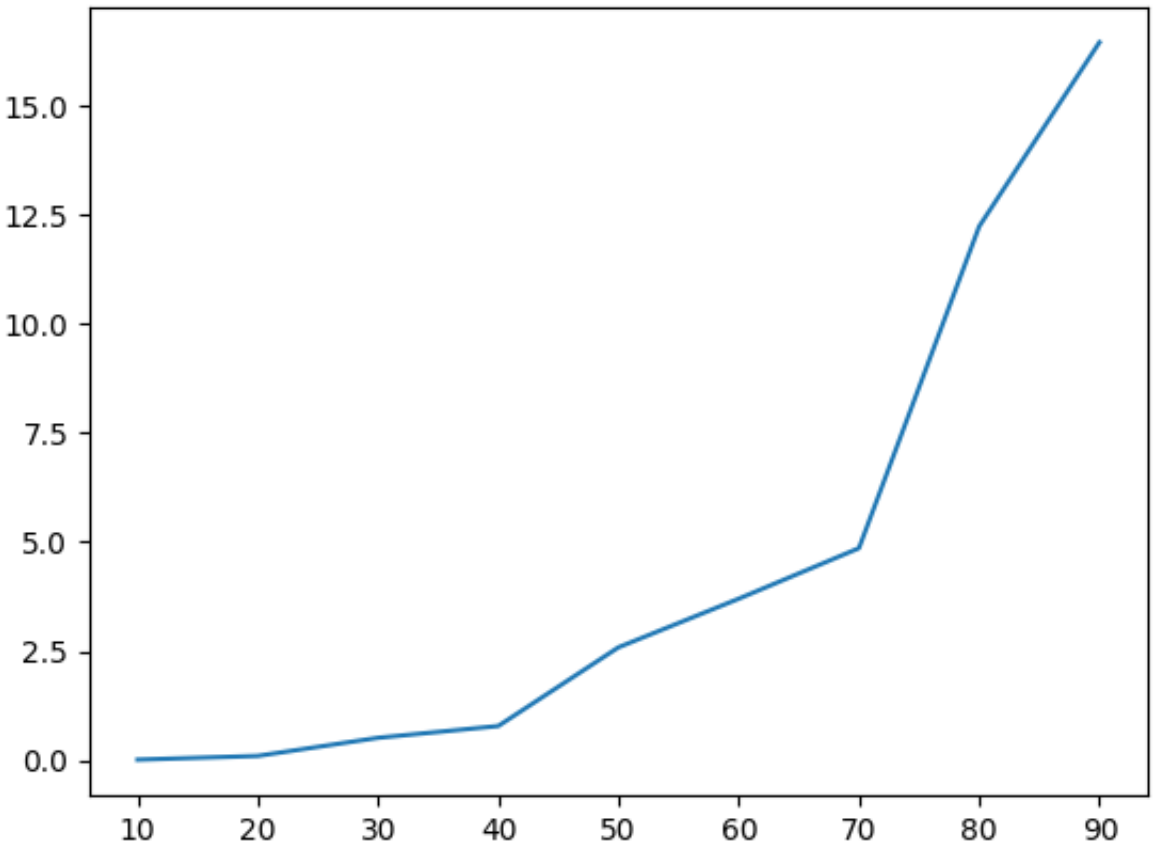


Figure 6: Зейдель

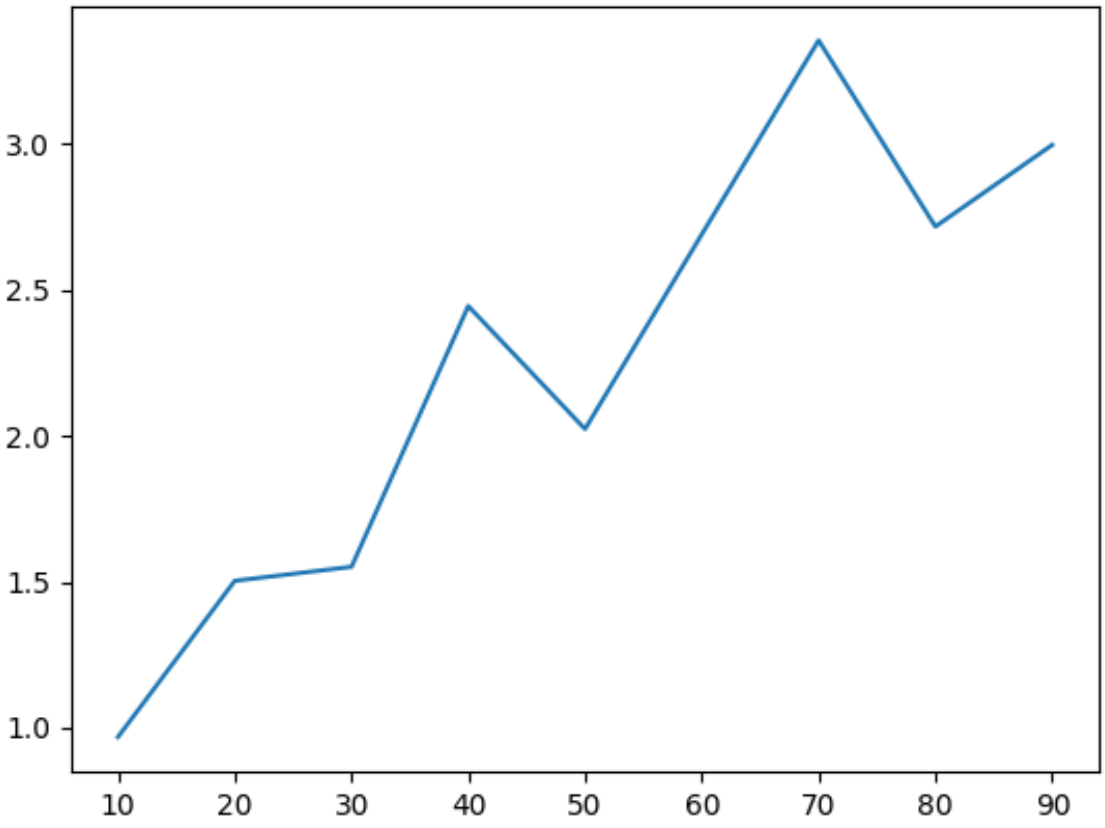


Figure 7: Средняя ошибка при решении методом Зейделя

**Код**

```
from bisect import bisect_left
from itertools import groupby


class Csr:
    def __init__(self, m):
        if isinstance(m, Csr):
            self.v, self.c, self.r, self.m = m.v.copy(), m.c.copy(), m.r.copy(), m.m
            return
        if isinstance(m, tuple):
            self.v, self.c, self.r, self.m = m
            return
        self.v, self.c, self.r, self.m = [], [], [0], len(m[0])
        for i, row in enumerate(m):
            for j, x in enumerate(row):
                if not x:
                    continue
                self.v.append(x)
                self.c.append(j)
            self.r.append(len(self.v))

    @classmethod
    def from_clist(cls, clist, n, m=None):
        m = m or n
        clist.sort()
        v, c, r = [], [], [0]
        j = 0
        for i in range(n):
            while j < len(clist) and clist[j][0] == i:
                v.append(clist[j][2])
                c.append(clist[j][1])
                j += 1
            r.append(len(v))
        return cls((v, c, r, m))

    @property
    def n(self):
        return len(self.r) - 1

    def get_row(self, i):
        start, end = self.r[i : i + 2]
        row = [0] * self.m
        for v, c in zip(self.v[start:end], self.c[start:end]):
            row[c] = v
        return row

    def get_item(self, i, j):
        start, end = self.r[i : i + 2]
        q = bisect_left(self.c, j, start, end)
        if q < len(self.c) and self.c[q] == j:
            return self.v[q]
```

```
        return 0

    def __getitem__(self, item):
        if isinstance(item, int):
            return self.get_row(item)
        return self.get_item(*item)

    def to_mat(self):
        return [self[i] for i in range(self.n)]

    def mul_cool(self, other):
        v, c, r = [], [], [0]
        for i in range(self.n):
            for j in range(self.m):
                ks, es = self.r[i : i + 2]
                ko, eo = other.r[j : j + 2]
                res = 0
                while ks < es and ko < eo:
                    if self.c[ks] == other.c[ko]:
                        res += self.v[ks] * other.v[ko]
                        ks += 1
                        ko += 1
                    elif self.c[ks] < other.c[ko]:
                        ks += 1
                    else:
                        ko += 1
                if res:
                    v.append(res)
                    c.append(j)
            r.append(len(v))
        return Csr((v, c, r, self.m))

    def transpose(self):
        qs = []
        for i in range(self.n):
            start, end = self.r[i : i + 2]
            for vv, j in zip(self.v[start:end], self.c[start:end]):
                qs.append((j, i, vv))
        return self.from_clist(qs, self.n)

    def __str__(self):
        return str((self.v, self.c, self.r))

    def __repr__(self):
        return str(self)
```

```
from primat.lab3.csr import *


def gen_l(a: Csr, n):
    v, c, r = [], [], []
    resv, resr = [], []
    for i in range(a.n):
```

```
        r.append(len(v))
        if i > n:
            q = -a[n, i] / a[n, n]
            if q:
                resv.append(q)
                resr.append(i)
                v.append(resv[-1])
                c.append(n)
        v.append(1)
        c.append(i)
    r.append(len(v))
    return Csr((v, c, r, a.m)), resv, resr


def decomp(a: Csr):
    vs, rs = [], []
    for i in range(a.n):
        li, resv, resr = gen_l(a, i)
        vs.append(resv)
        rs.append(resr)
        a = a.mul_cool(li)
    v, c, r = [], [], [0]
    for i in range(a.n):
        for j in range(i):
            k = bisect_left(rs[j], i)
            if k<len(rs[j]) and rs[j][k] == i:
                v.append(-vs[j][k])
                c.append(j)
        v.append(1)
        c.append(i)
        r.append(len(v))
    return Csr((v, c, r, a.m)), a


def solve(l, u, b):
    u = u.transpose()
    n = l.n
    y = []
    for i in range(n):
        start, end = l.r[i: i + 2]
        yv = b[i]
        for v, j in zip(l.v[start:end], l.c[start:end]):
            if j >= i:
                break
            yv -= y[j] * v
        y.append(yv / l[i, i])

    x = []
    for i in range(n):
        xv = y[n - i - 1]
        start, end = u.r[n - i - 1: n - i + 1]
        for v, j in zip(u.v[start:end][::-1], u.c[start:end][::-1]):
            if n - j - 1 >= len(x):
```

```
                break
            xv -= x[n - j - 1] * v
        x.append(xv / u[n - i - 1, n - i - 1])
        # x.append((y[n - i - 1] - sum(x[j] * u[n - j - 1, n - i - 1] for j in range(i))) / u[n - i -
    return x[::-1]


def inv(a):
    n = a.n
    l, u = decomp(a)
    v, c, r = [], [], [0]
    for j in range(n):
        x = [0] * n
        x[j] = 1
        col = solve(l, u, x)
        for i, vv in enumerate(col):
            if vv:
                v.append(vv)
                c.append(i)
        r.append(len(v))
    return Csr((v, c, r, n))
```

```
import numpy as np

from primat.lab2.grad import norm_sq
from primat.lab3.csr import Csr


def zeydel(a, b, eps):
    counter = 0
    n = a.n
    xp = None
    x = np.array(b, copy=True, dtype=np.longdouble)
    while xp is None or norm_sq(x - xp) > eps**2:
        counter += 1
        xp = x.copy()
        for i in range(n):
            x[i] = b[i]
            start, end = a.r[i : i + 2]
            for v, j in zip(a.v[start:end], a.c[start:end]):
                if i != j:
                    x[i] -= v * x[j]
            x[i] /= a[i, i]
    return x, counter
```

```
from random import choices

import numpy as np

from primat.lab3.csr import *
import math
```

```
def gilbert(n):
    A = [[1 / (i + j + 1) for j in range(n)] for i in range(n)]
    x = [(i + 1) for i in range(n)]
    F = np.matmul(np.array(A), np.array(x))
    return Csr(A), F


def qq(n, k):
    q = list(range(-4, 1))
    v, c, r, f = [], [], [0], []
    for i in range(n):
        row = list(choices(q, k=n - 1))
        row.insert(i, -sum(row) + 10 ** (-k))
        fv = 0
        for j, w in enumerate(row):
            if not w:
                continue
            v.append(w)
            c.append(j)
            fv += w * (j + 1)
        f.append(fv)
        r.append(len(v))
    return Csr((v, c, r, n)), f


def almost_diag(n):
    v, c, r, f = [], [], [0], []
    points = []
    for i in range(n):
        val = math.ceil(np.random.random() * 4 + 1)
        points.append((i,i,val))
        fv = 0
        fv += val * (i + 1)
        if np.random.random() > 0.5:
            points.append((i, (i+1) % n, val-1))
            fv += (val-1) * ((i+1) % n + 1)
        f.append(fv)
        r.append(len(v))
    return Csr.from_clist(points,n,n), f
```