Университет ИТМО

Факультет информационных технологий и программирования
Направление прикладной математики и информатики

ОТЧЁТ ПО ЛАБОРАТОРНЫМ РАБОТАМ

курса «Методы оптимизации»

Выполнили студенты:

Мозжевилов Данил, Кучма Андрей
Группы: M3238, M3239

Санкт-Петербург, 15 апреля 2021 г.

Содержание

Лабораторная работа 1

Методы многомерной оптимизации

1.1. Постановка задачи и цель работы

1. Реализовать алгоритмы:

   - Метод градиентного спуска
   - Метод наискорейшего спуска
   - Метод сопряженных градиентов

   Оценить как меняется скорость сходимости, если для поиска величины шага используются различные методы одномерного поиска.

2. Проанализировать траектории методов для нескольких квадратичных функций: придумайте две-три квадратичные двумерные функции, на которых работа каждого из методов будет отличаться. Нарисовать графики с линиями уровня функций и траекториями методов.

3. Исследовать, как зависит число итераций, необходимое методам для сходимости, от следующих двух параметров:

   - числа обусловленности $k \geq 1$ оптимизируемой функции
   - размерности пространства n оптимизируемых переменных

   Сгенерировать от заданных параметров k и n квадратичную задачу размерности n с числом обусловленности k и запустить на ней методы многомерной оптимизации с некоторой заданной точностью. Замерить число итераций $T(n, k)$, которое потребовалось сделать методу до сходимости.

1.2. Иллуюстрации работы градиентных методов на двумерных квадратичных функцкиях

Рассмотрим функцию $f(x, y) = x^2 - xy + 4y^2 + 2x + y$. В матричном виде ее вид $f(x) = 1/2 * (Ax, x) + b * x$, где $A = \begin{pmatrix} 2 & -1 \\ -1 & 8 \end{pmatrix}$ и $b = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$.

$det(A - \lambda E) = \begin{vmatrix} 2-\lambda & -1 \\ -1 & 8-\lambda \end{vmatrix} = (2-\lambda) * (8-\lambda) - 1 = 15 - 10 * \lambda + \lambda^2 = (5 + \sqrt{10} - \lambda) * (5 + \sqrt{10} - \lambda)$. Собственные значение матрицы A положительны, следовательно квадратичная форма f положительно определенная, а следовательно выпукла вниз. Таким образом к этой квадратчной форме можно применить алгоритмы минимизации. Для начала найдем точку минимума функции аналитически.

Надем точку, в которой градиент данной функции обращается в ноль. Это и будет точка минимума функции. $grad\ f = \left(2 * x - y + 2 \quad -x + 8y + 1\right)^T = (0\ 0)^T$ Решив систему линейных уравнений, получаем $x = -17/15, y = -4/15$ и $min(f(x, y)) = -19/15$

1.3. Общая схема того, как мы реализовывали алгоритмы

В начале мы создали классы Matrix, DiagonalMatix и Vector и для них перегрузили операторы $'+'$, $'-'$, $'*'$ и $'[\,]'$ (класс DiagonalMatix появился только под конец, когда мы уже начали тестировать и узнали, что для тестов нужны только диагональные матрицы и оказалось, что в коде для матриц испльзовался только оператор $'*'$, поэтому мы не стали реализовывать остальные перегрузки для этого класса).

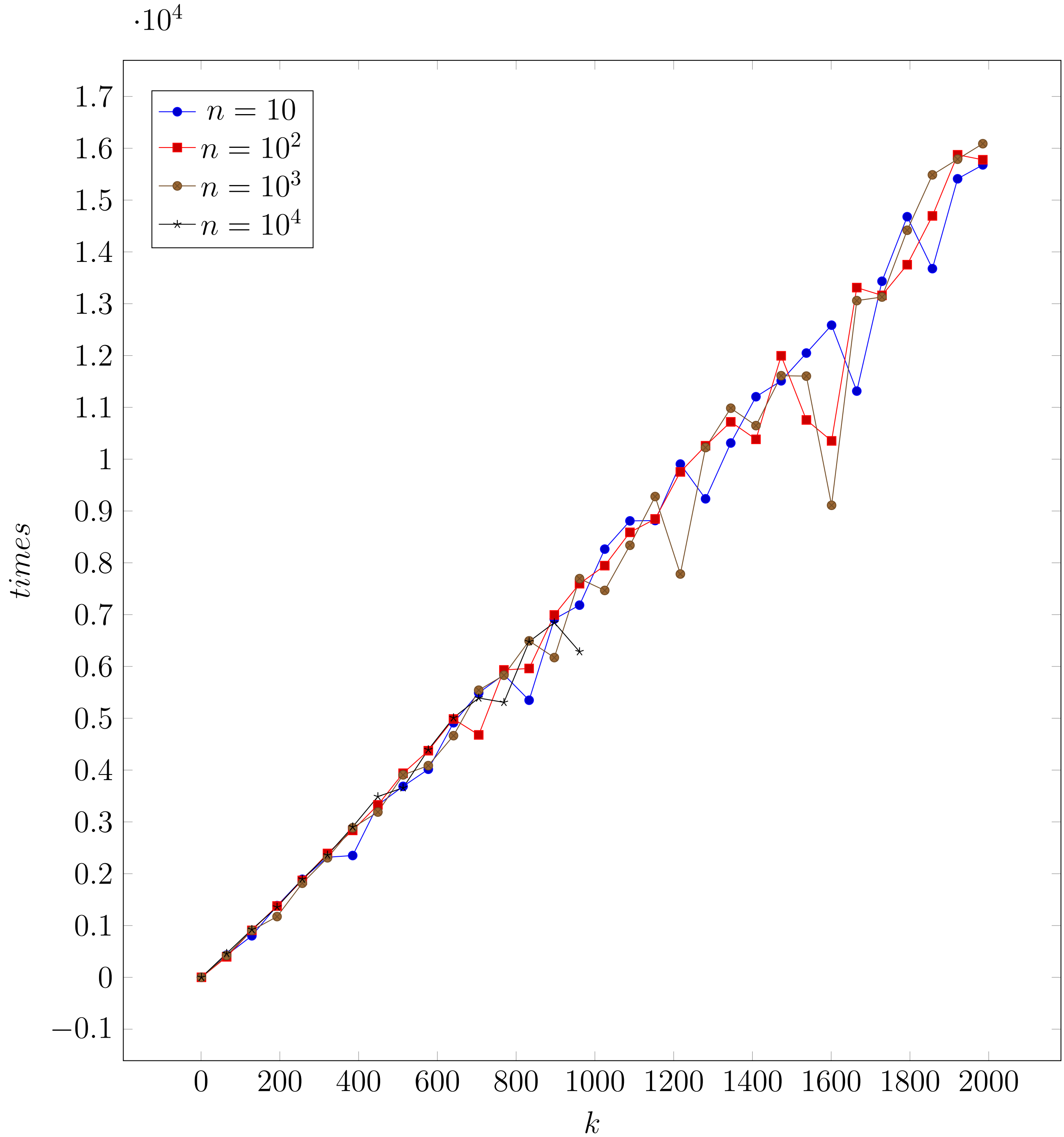
Далее мы решили не использовать лямда-функции для задания квадратичных форм, а сделать отдельные классы QuadraticFunction и DiagonalQuadraticFunctio в которых храниться матрица A, вектор b и число c, и просто передавать их в качестве параметров в реализуемые алгоритмы, к тому же в классе можно хранить всю историю обращения к ней.

Также мы создали класс GeneratorQudraticFunction, который генерировал рандомые вектора по заданной размерности и числу обусловленности.

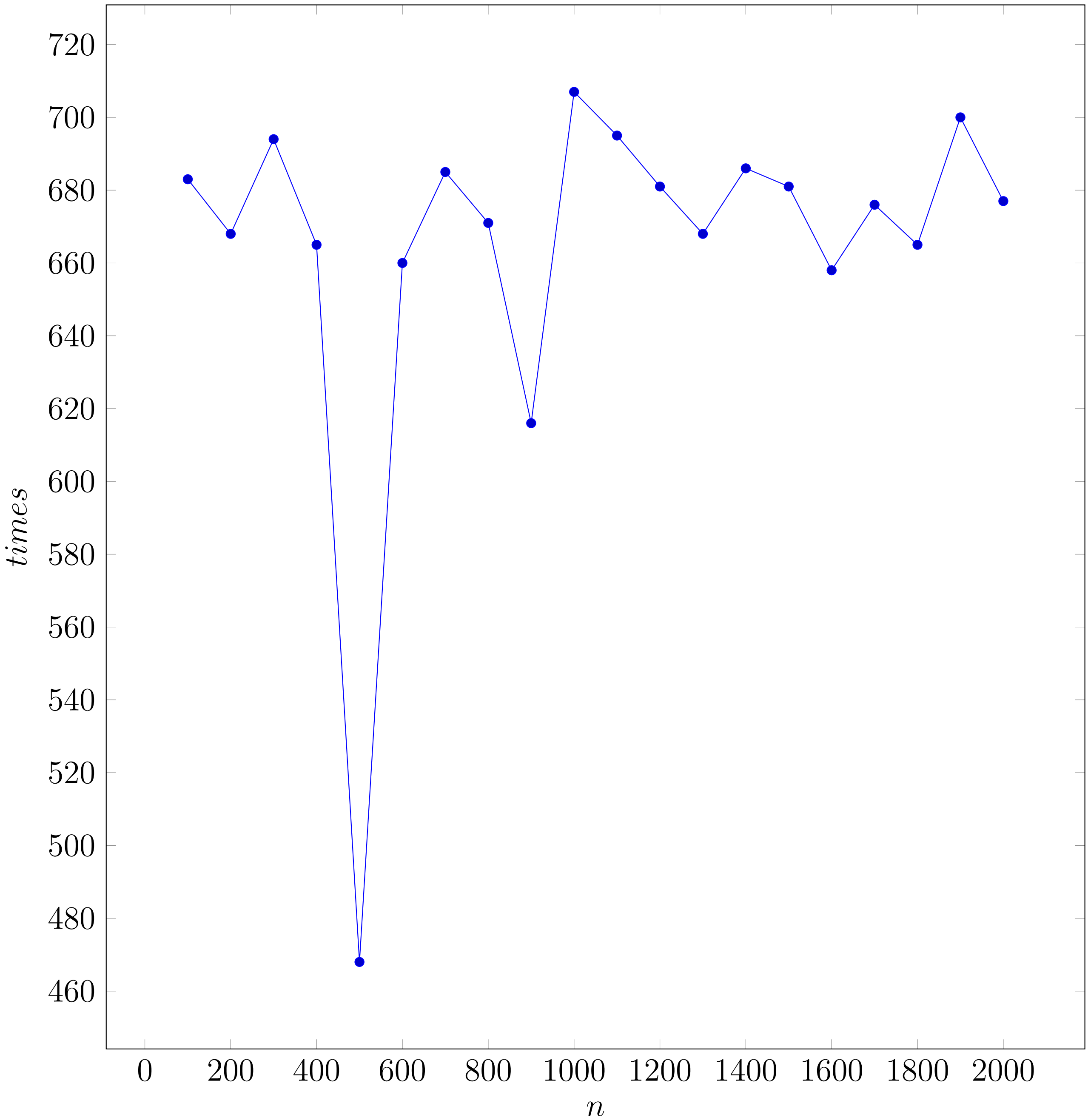
Точность для алгоритмов мы решили задать всего лишь 0.1, так как при тестировании не хотелось ждать по 30 минут, пока алгоритмы найдут необходимый минимум для всех сгенерированных функций.

1.4. Метод градиентного спуска

Заметим, что в методе градиентного спуска константа линейной скорости сходимости не зависит от размерности пространства, а только от собственных чисел матрицы квадратичной формы, а следовательно для всех размерностей должны получится похожие результаты, что мы как раз таки видим на графике ниже.

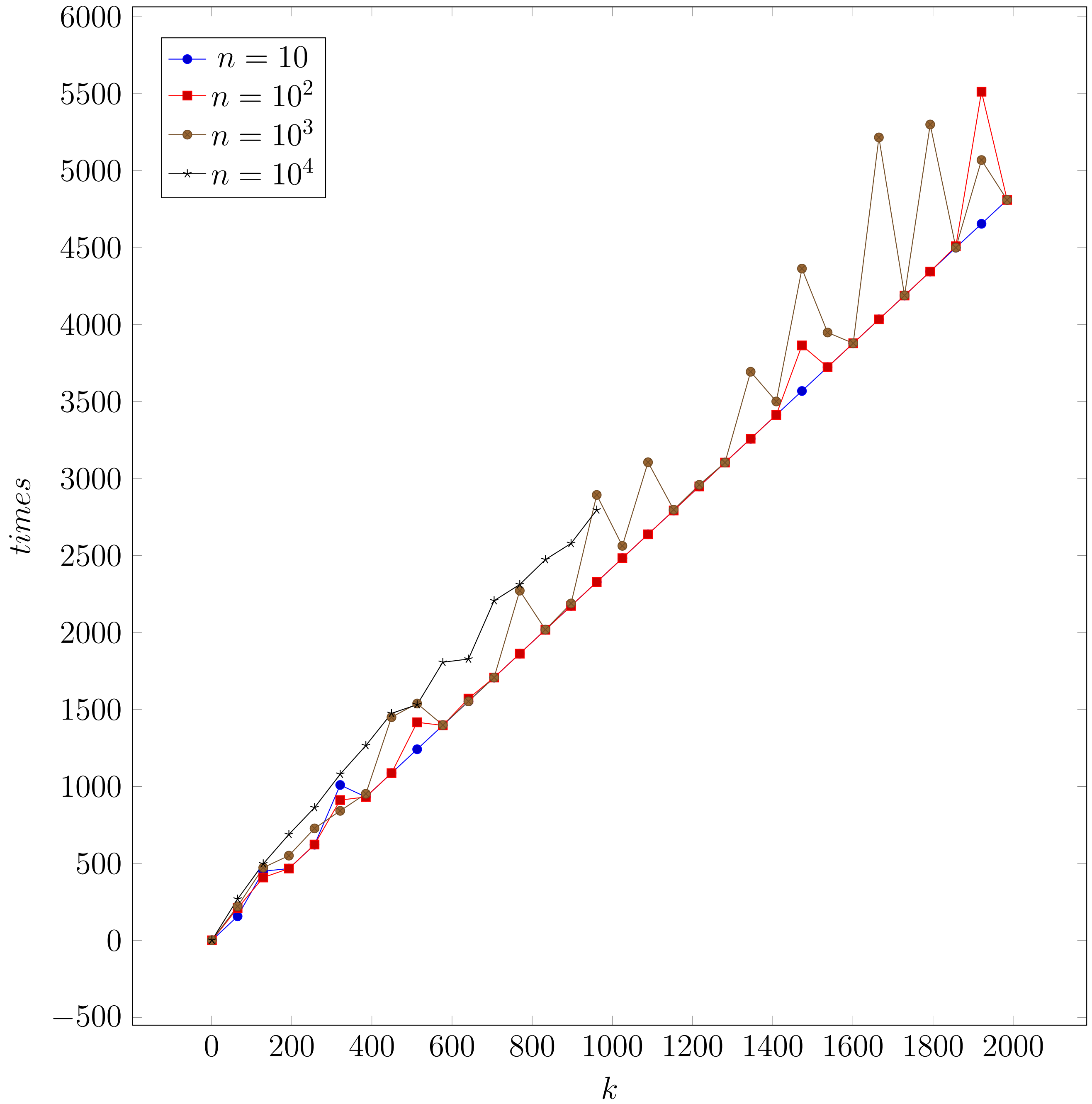


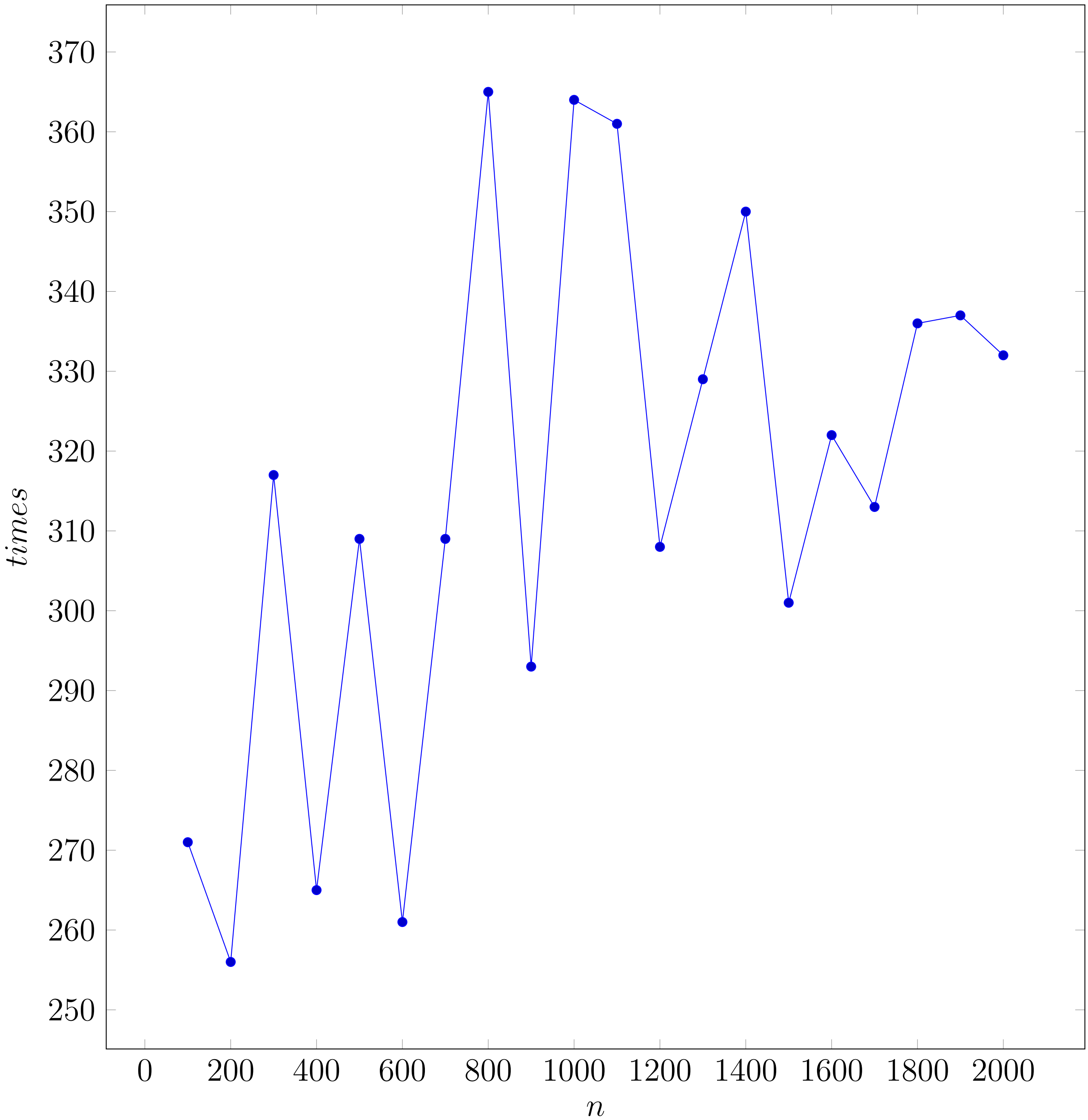
$\cdot 10^4$
1.7
1.6
1.5
1.4
1.3
1.2
1.1
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
−0.1
times
0
200
400
600
800
1000
1200
1400
1600
1800
2000
k
$n = 10$
$n = 10^2$
$n = 10^3$
$n = 10^4$



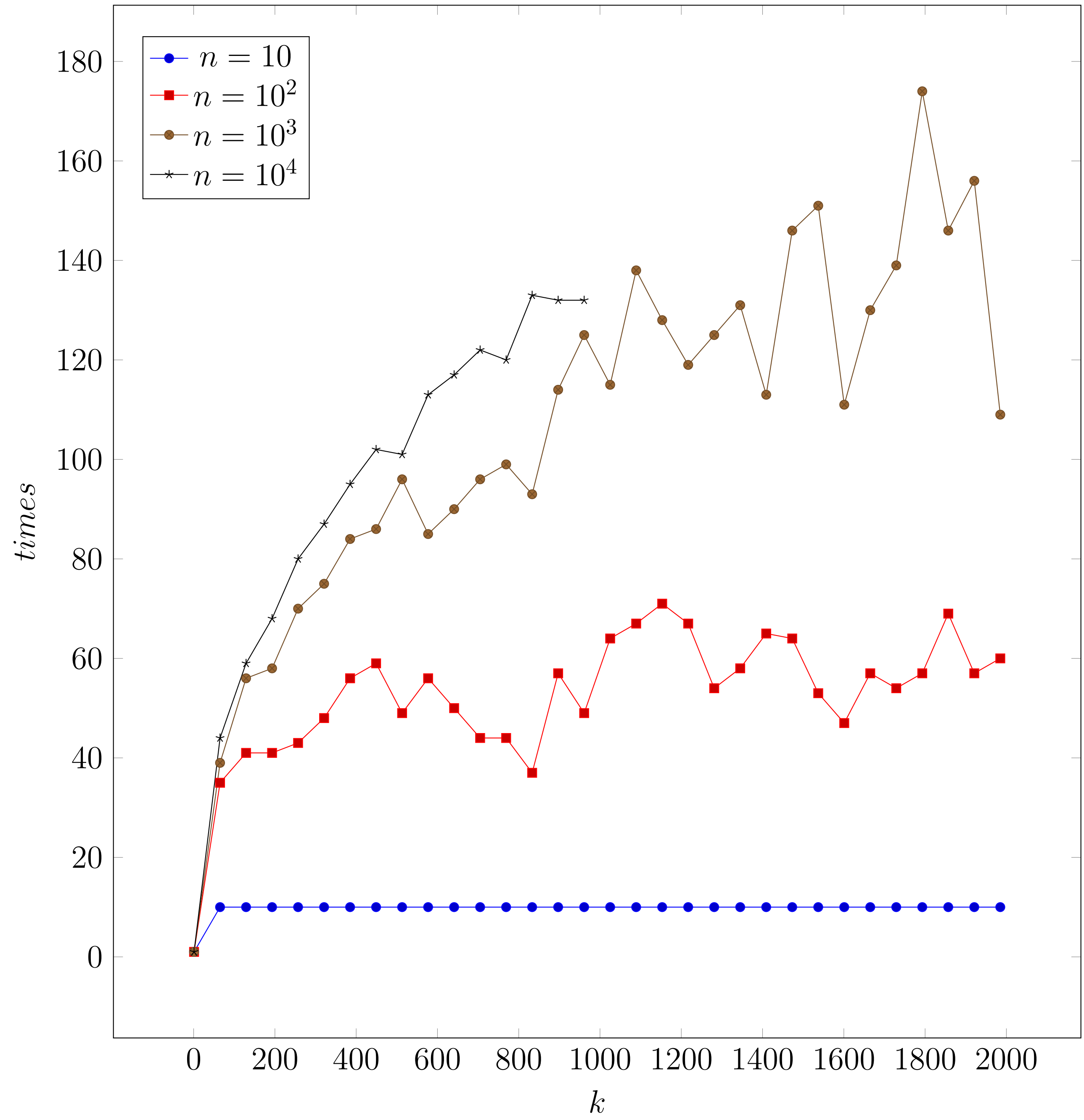
times
720
700
680
660
640
620
600
580
560
540
520
500
480
460
0
200
400
600
800
1000
1200
1400
1600
1800
2000
n

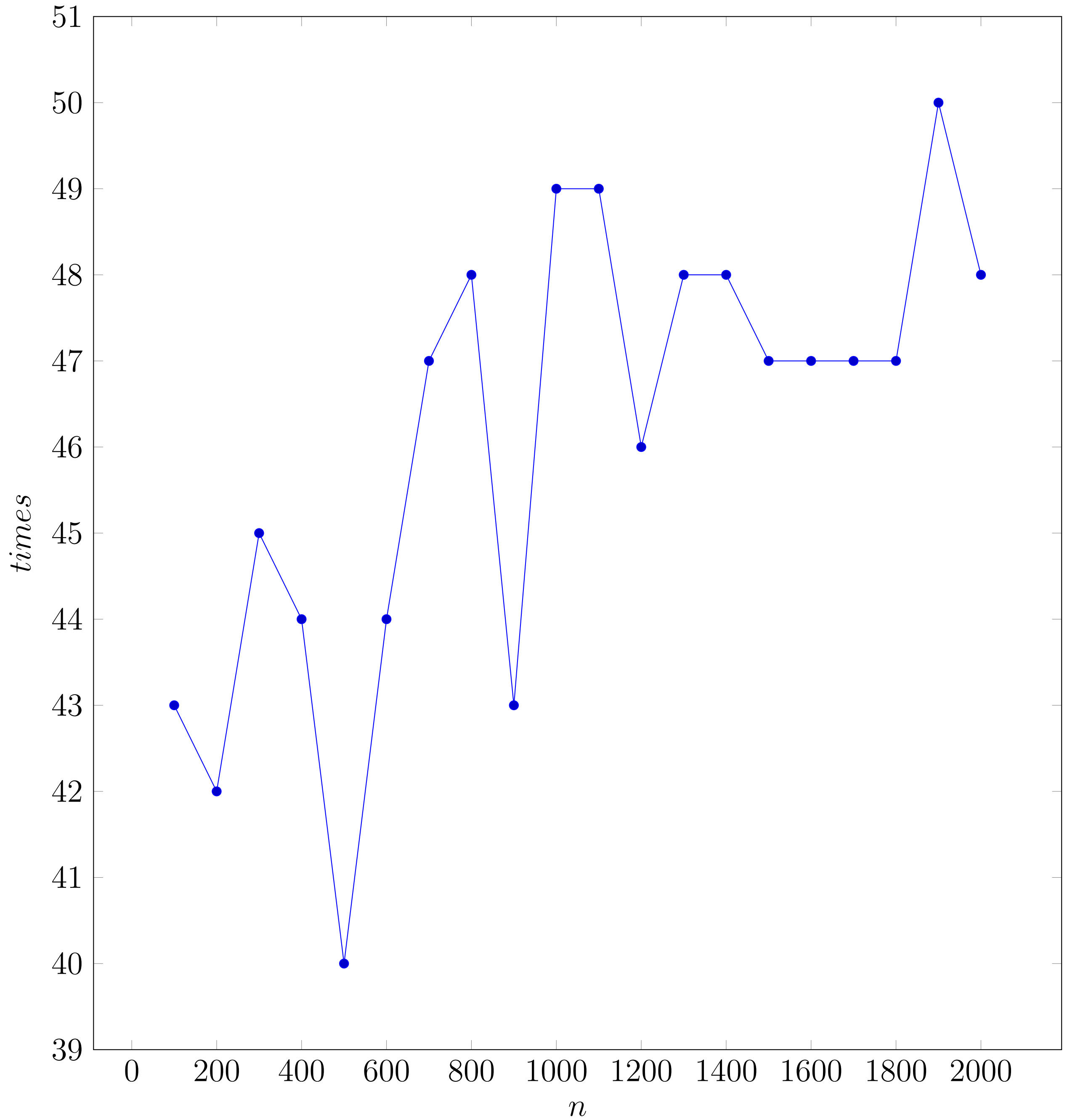
1.5. Метод наискорейшего спуска




370
360
350
340
330
320
310
300
290
280
270
260
250
times
0
200
400
600
800
1000
1200
1400
1600
1800
2000
n

1.6. Метод сопряженных градиентов




51
50
49
48
47
46
45
44
43
42
41
40
39
times
0
200
400
600
800
1000
1200
1400
1600
1800
2000
n