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, 16 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 DiagonalQuadraticFunction, 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 функции, что мы и делали.

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

Пример того, как выглядели наши сгенерированные функции:

```
1 38.0198 208.636 276.712 419.618 517.318 549.321 565.029 598.464 641
1 56.0696 86.0772 94.8904 129.966 133.73 151.615 295.072 304.457 330.866
1 121.07 250.754 316.186 452.644 463.517 492.598 526.129 690.467 706.2
```

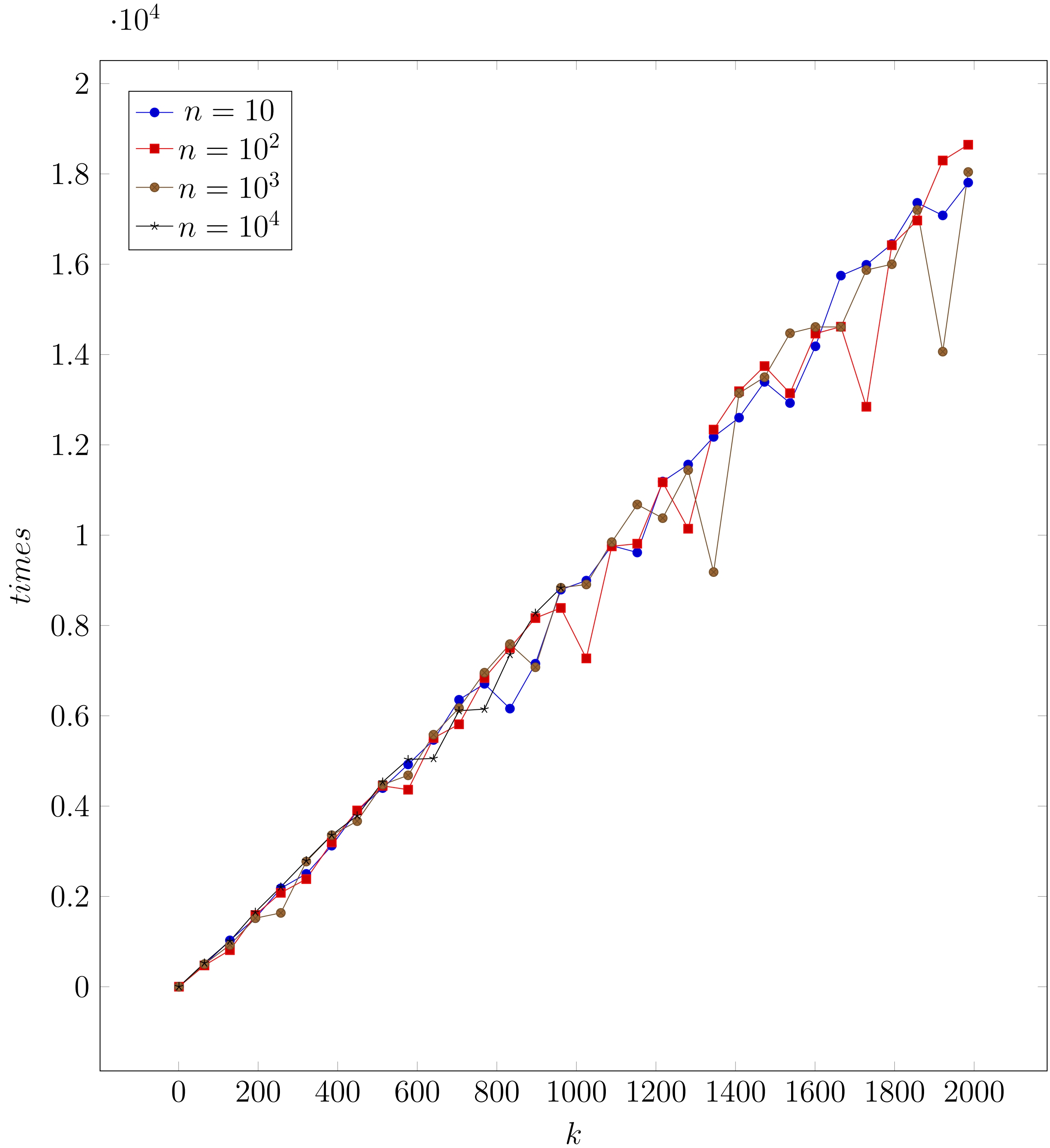
Первая строчка это диагональная матрица A. В данном случае с числом обусловленноости 641. Вторая строчка это вектор b. Прибавление константы мы решили не генерировать, так как на поиск точки минимума она не влияет. Третья строка это начальная точка x_0.

Все сгенерированные функции мы сохраняли в файлы, и благодаря этому не приходилось заново генерировать функции для каждого запуска программы и можно было в несколько заходов протестировать все алгоритмы.

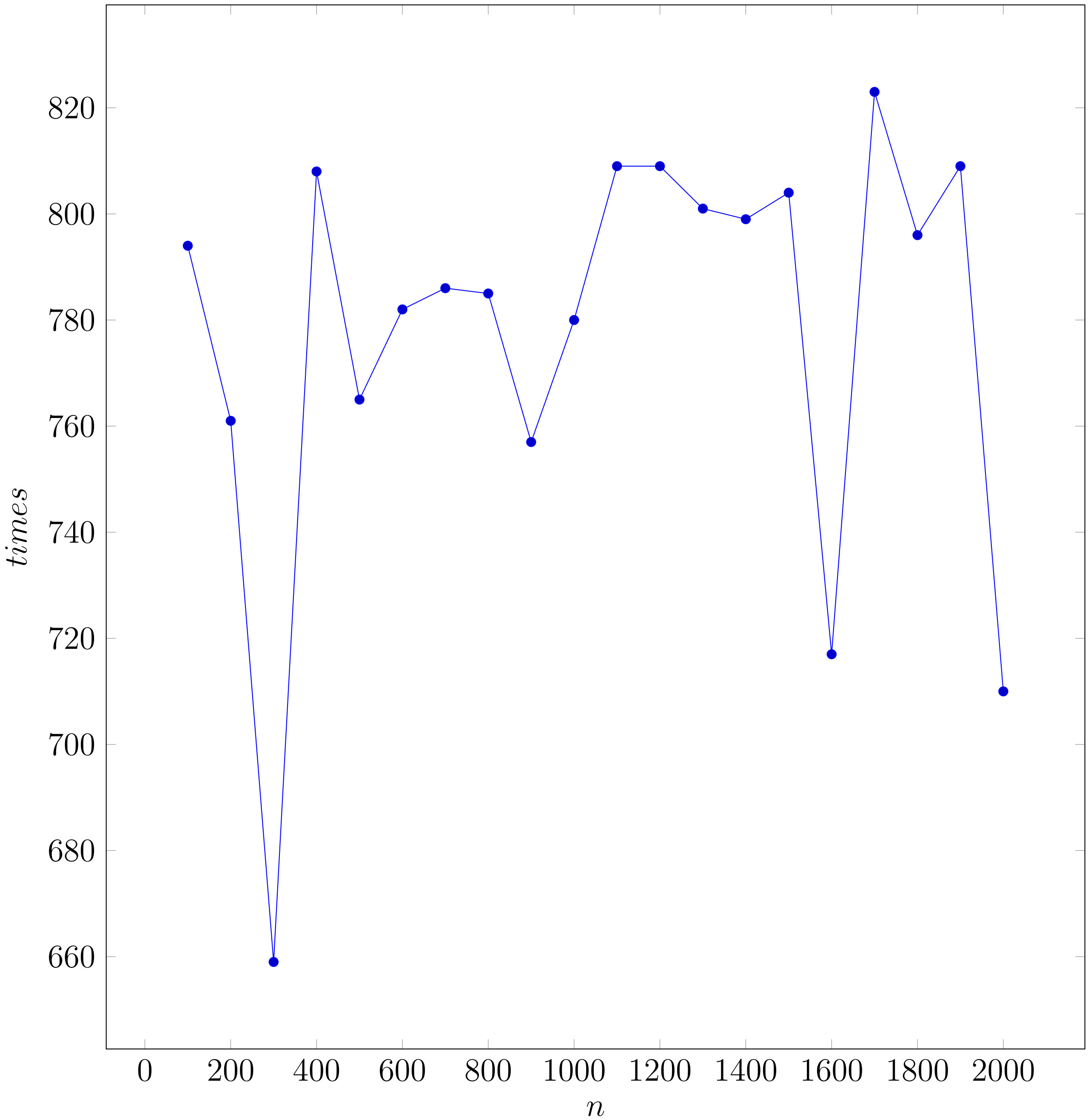
Точность для алгоритмов мы решили задать всего лишь 0.1, так как при тестировании не хотелось ждать по 30 минут, пока алгоритмы найдут необходимый минимум для всех сгенерированных функций. Также при вычислениях минимума у функции размерности $n = 10^4$ пришлось ограничится числом обусловленности $k = 1000$, так как нам не хватало оперативной памяти на компютере для хранения истории всех вычислений функции (При числе обусловленности $k = 2000$, как мы увидим ниже, происходит по $1.7 * 10^4$ итераций в методе градиентного спуска, на каждой из них нужно сохранить три вектора размера 10^4 – точка вычисления функции, точку для градиента и сам градиент. И того $1.7 * 10^4 * 3 * 10^4 = 5.1 * 10^8$ чисел типа long double, т. е. $5.1 * 10^8 * 8 = 408 * 10^7$ байт = 4.08 гигабайт и ни у кого из нас нет столько оперативной памяти на компютере).

1.4. Метод градиентного спуска

Заметим, что в методе градиентного спуска константа линейной скорости сходимости $q = 2/(l + L)$ не зависит от размерности пространства n, а только от собственных чисел матрицы A квадратичной формы, а следовательно для всех размерностей должны получится схожие результаты, что мы как раз таки видим на графике ниже. Но есть один минимум, который выбиваетсятся из общецм массы. Скорее всего это из-за того, что сгенерированная точка попала в многомерный овраг и из-за этого не происходило сильных биений. По сгенерированному тесту это очень сложно понять.

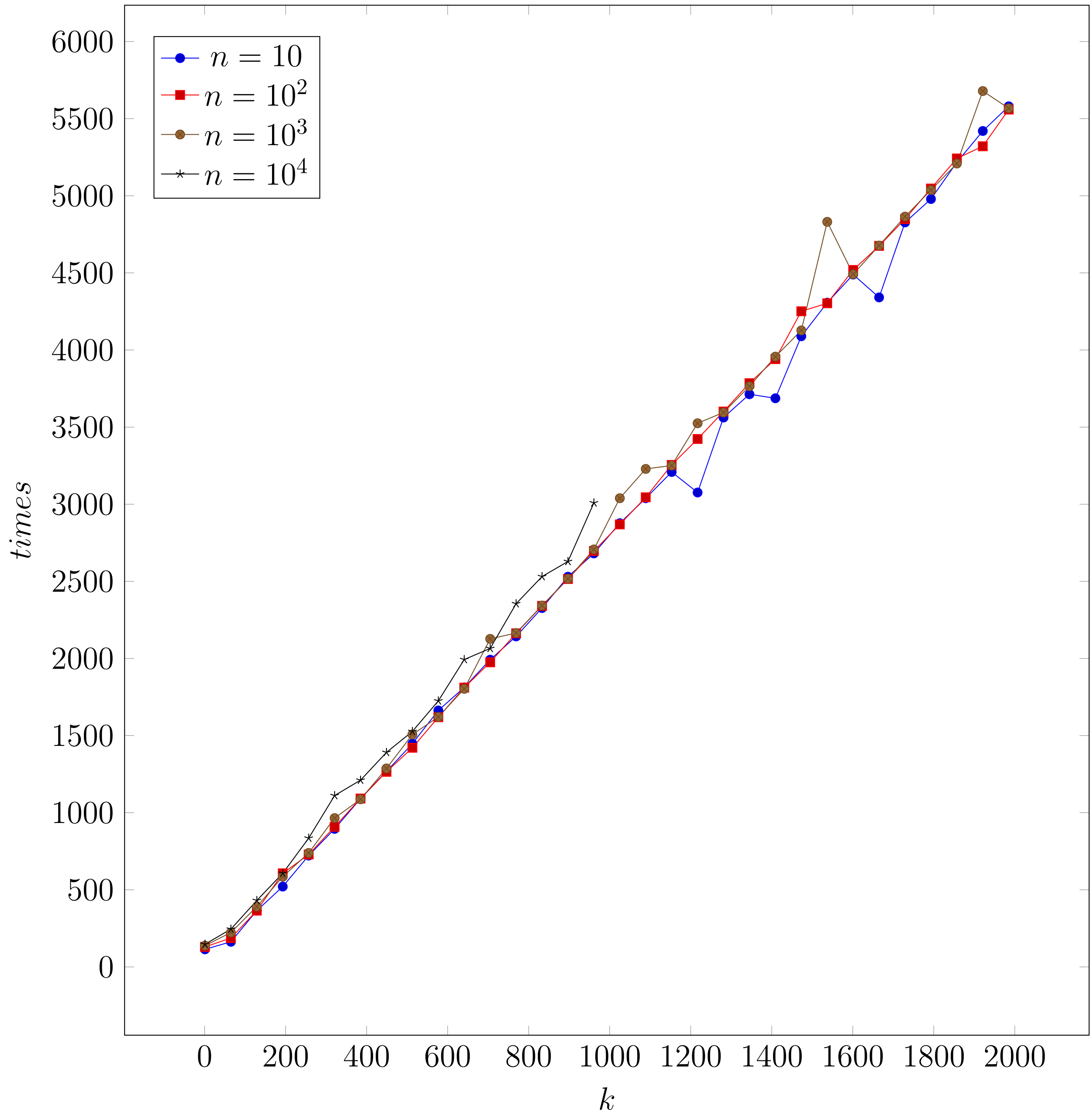


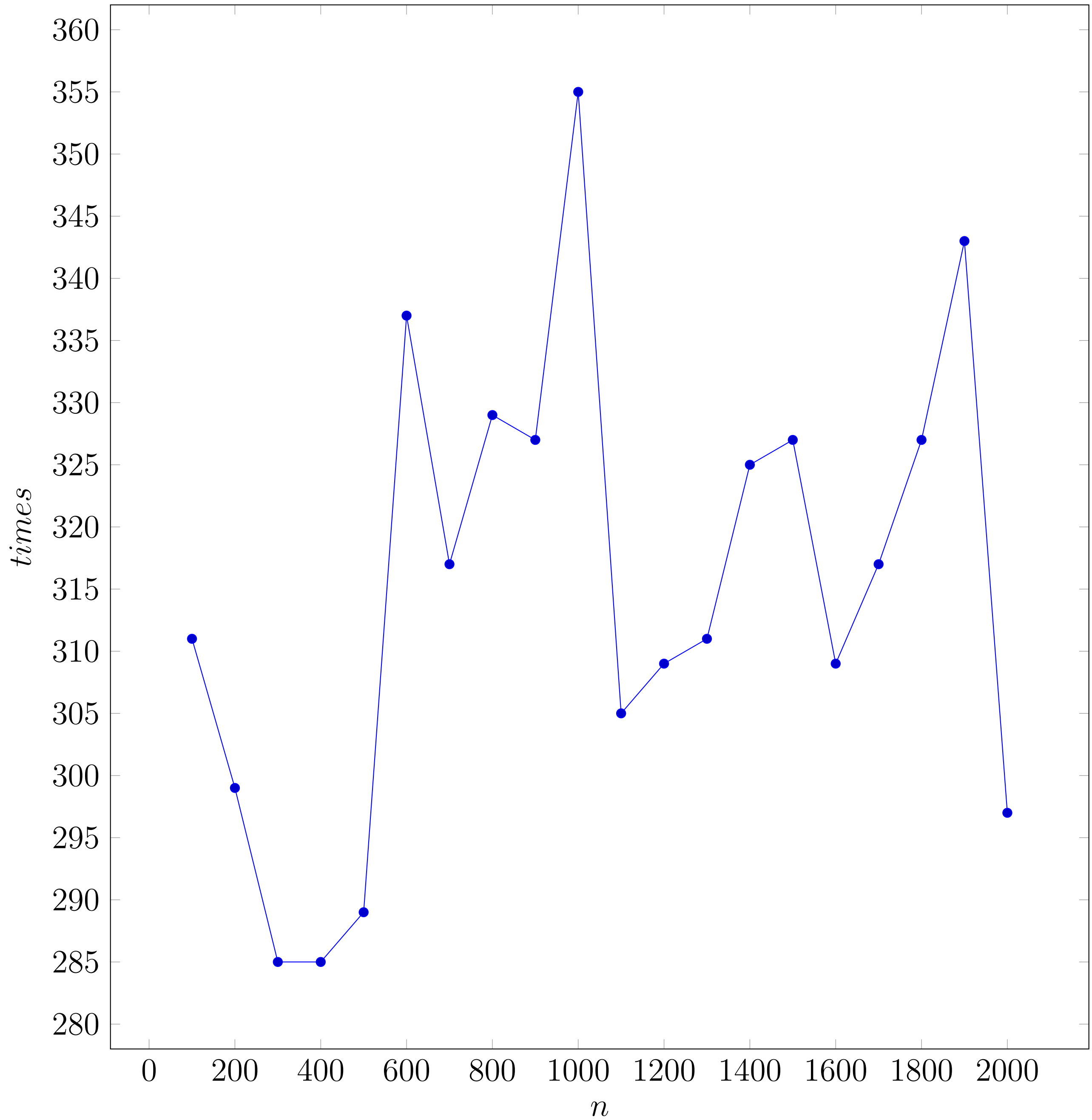
$\cdot 10^4$
$n = 10$
$n = 10^2$
$n = 10^3$
$n = 10^4$
$times$
k
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
0
200
400
600
800
1000
1200
1400
1600
1800
2000



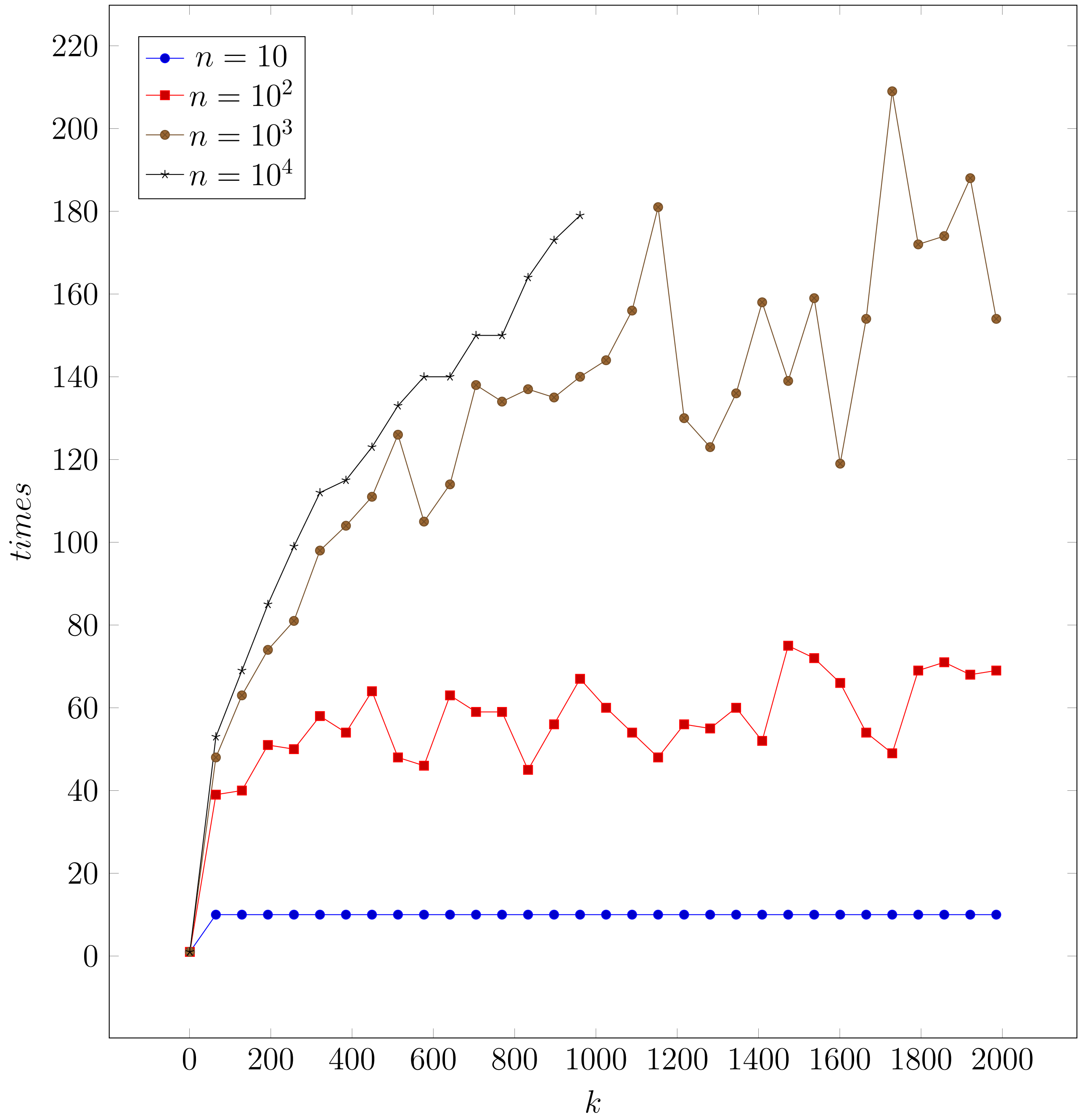
820
800
780
760
740
720
700
680
660
times
0
200
400
600
800
1000
1200
1400
1600
1800
2000
n

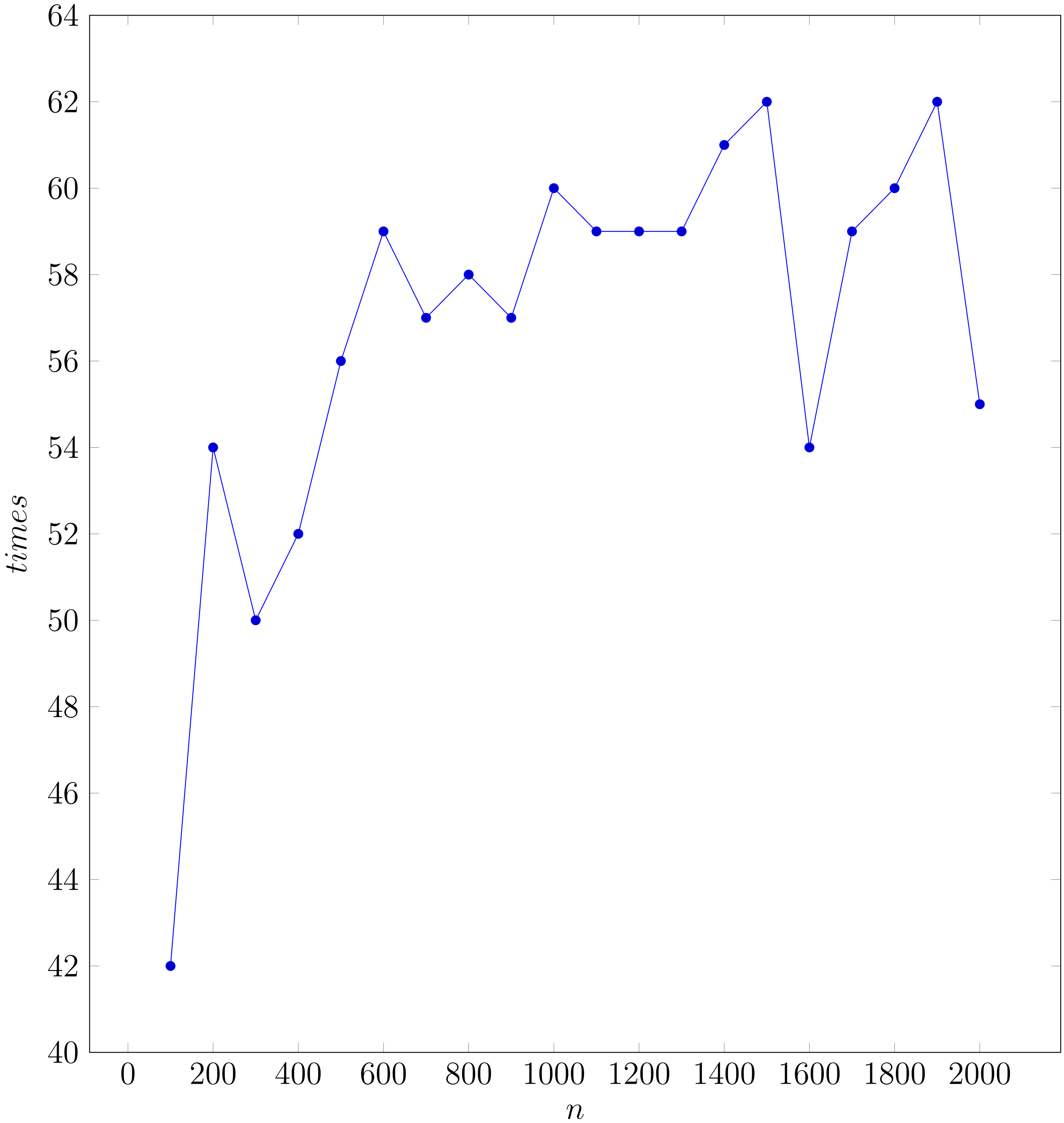
1.5. Метод наискорейшего спуска




360
355
350
345
340
335
330
325
320
315
310
305
300
295
290
285
280
times
0
200
400
600
800
1000
1200
1400
1600
1800
2000
n

1.6. Метод сопряженных градиентов




64
62
60
58
56
54
52
50
48
46
44
42
40
times
0
200
400
600
800
1000
1200
1400
1600
1800
2000
n