

УНИВЕРСИТЕТ ИТМО

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ПРОГРАММИРОВАНИЯ
НАПРАВЛЕНИЕ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2
курса «Методы оптимизации»

Выполнили студенты:

Мозжевилов Данил, Кучма Андрей

Группы: М3238, М3239

Санкт-Петербург, 16 апреля 2021 г.

Оглавление

1. Методы многомерной оптимизации	2
1.1. Постановка задачи и цель работы	2
1.2. Общая схема того, как мы реализовывали алгоритмы	2
1.3. Иллюстрации работы градиентных методов на двумерных квадратичных функциях	3
1.3.1. Первая функция	3
1.3.2. Вторая функция	4
1.3.3. Третья функция	4
1.4. Метод градиентного спуска	4
1.5. Метод наискорейшего спуска	7
1.6. Метод сопряженных градиентов	9

Глава 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. 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 использовался 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 случайные 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 строка это диагональная матрица A . В данном случае с числом обусловленности 641. Вторая строка это вектор b . Прибавление константы мы решили не генерировать, так как на поиск точки минимума она не влияет. Третья строка это начальное приближение x_0 .

Все сгенерированные функции мы сохраняли в файлы, и благодаря этому не приходилось заново генерировать функции для каждого запуска программы, а также была возможность запускать тестирование на каждом методе отдельно.

Точность для алгоритмов мы решили задать всего лишь 0.01, так как при тестировании не хотелось ждать по 30 минут, пока алгоритмы найдут необходимый минимум для всех сгенерированных функций. Да и это не требовалось, так как судя по данным, которые мы получали, этой точности хватало, чтобы получать приближение до пяти знаков после запятой. Также при вычислении минимума у функции размерности $n = 10^4$ пришлось ограничиться числом обусловленности до $k = 1000$, так как наши алгоритмы работали очень долго.

1.3. Иллюстрации работы градиентных методов на двумерных квадратичных функциях

1.3.1. Первая функция

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 значения матрицы A положительны, следовательно квадратичная форма f положительно определенная, а следова-

тельно выпукла вниз. Таким образом к этой квадратичной 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. $\text{grad } f = (2 * x - y + 2 \quad -x + 8y + 1)^T = (0 \ 0)^T$. Решив систему линейных уравнений, получаем

$$x_m = -\frac{17}{15}, \quad y_m = -\frac{4}{15}$$

$$f(x_m, y_m) = -\frac{19}{15}$$

1.3.2. Вторая функция

Рассмотрим вторую функцию для исследования

$$f(x, y) = 8x^2 + 5y^2 + 10xy + 5x + 6y$$

. Аналогично предыдущей функции находим собственные значения, убедимся, что они положительны, находим точку минимума и само значение минимума.

$$x_m = \frac{1}{6}, \quad y_m = -\frac{23}{30}$$

$$f(x_m, y_m) = -\frac{133}{60}$$

1.3.3. Третья функция

Ну и наконец третья функция

$$f(x, y) = 22x^2 + 2y^2 + xy + x + y$$

. Аналогично с предыдущими

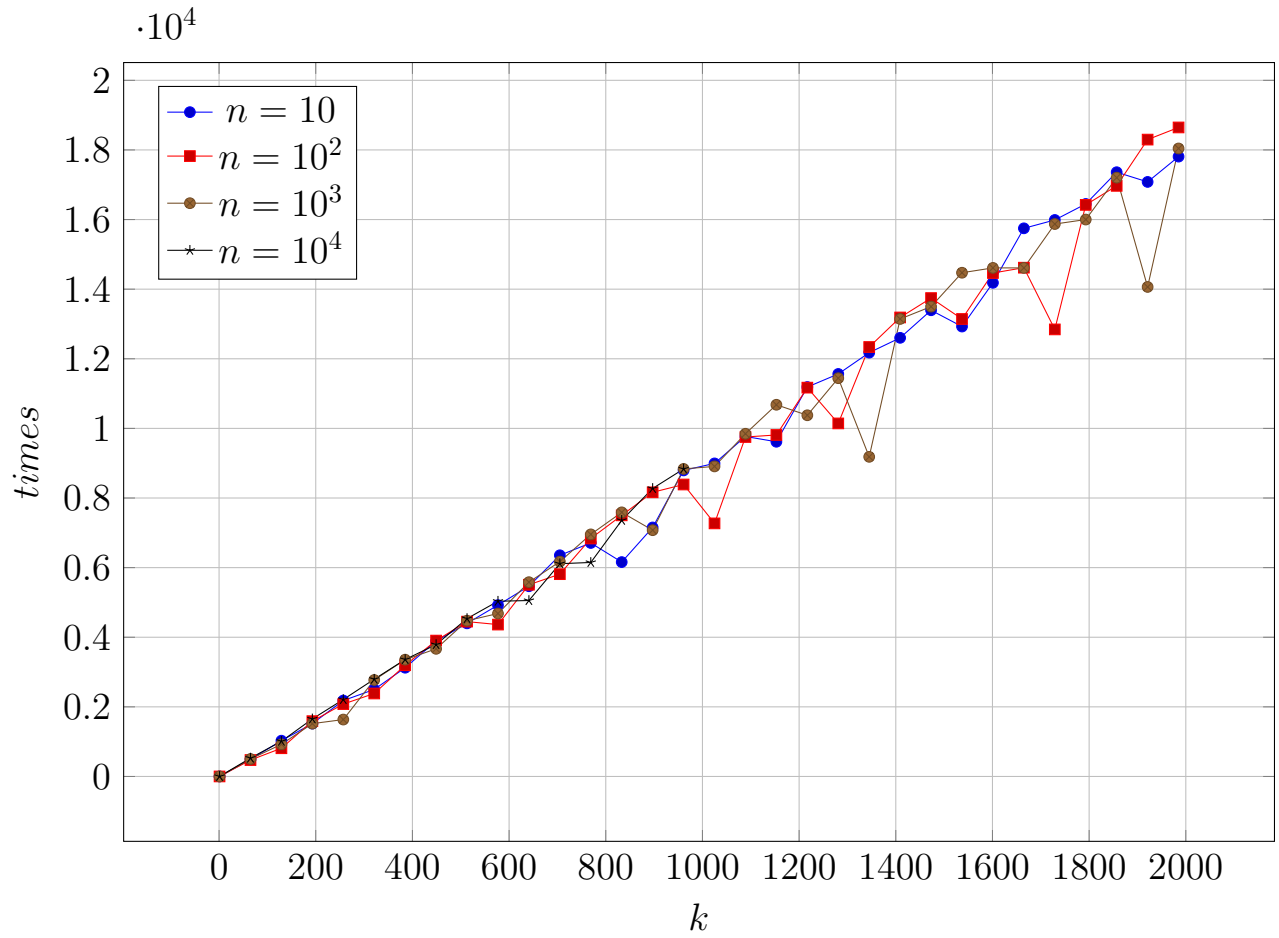
$$x_m = -\frac{3}{175}, \quad y_m = -\frac{43}{175}$$

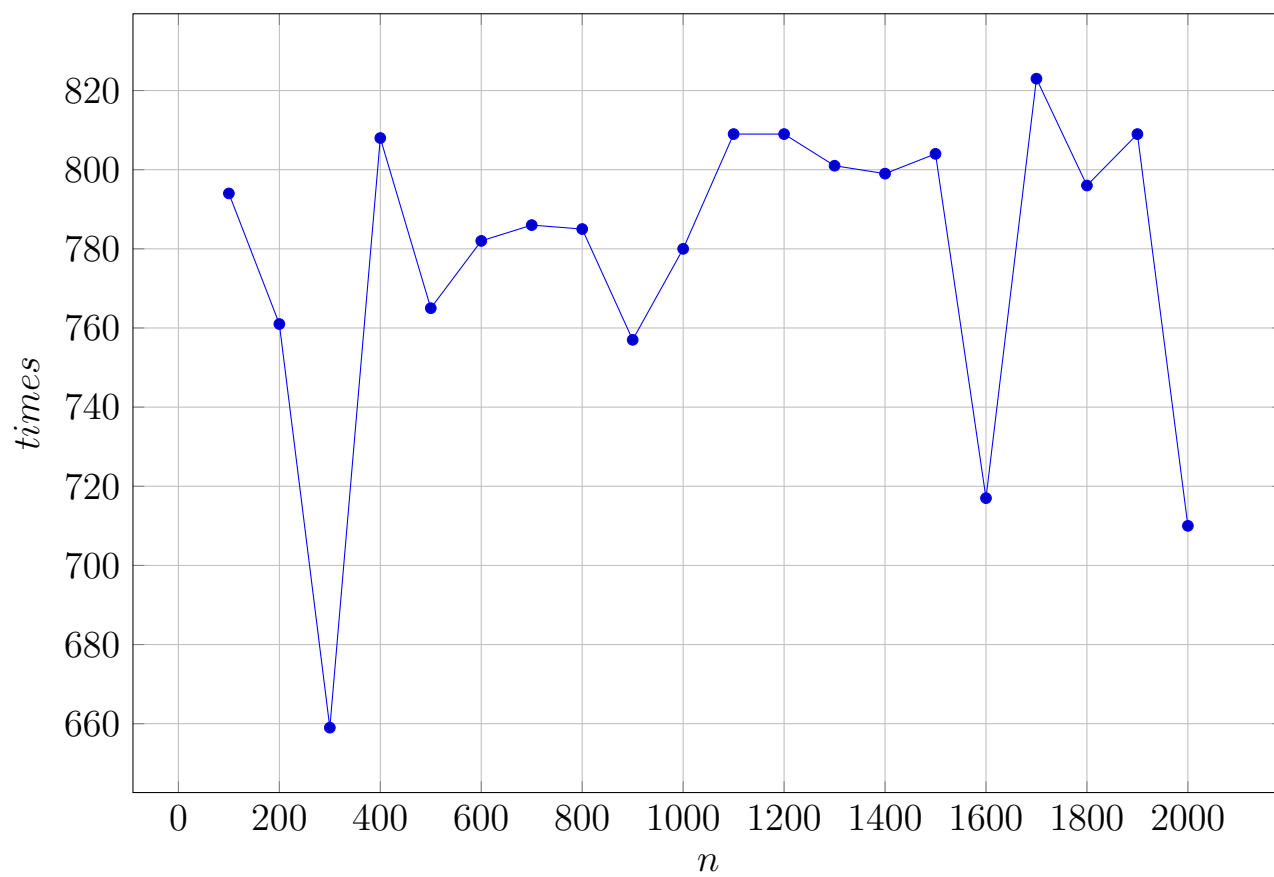
$$f(x_m, y_m) = -\frac{23}{175}$$

1.4. Метод градиентного спуска

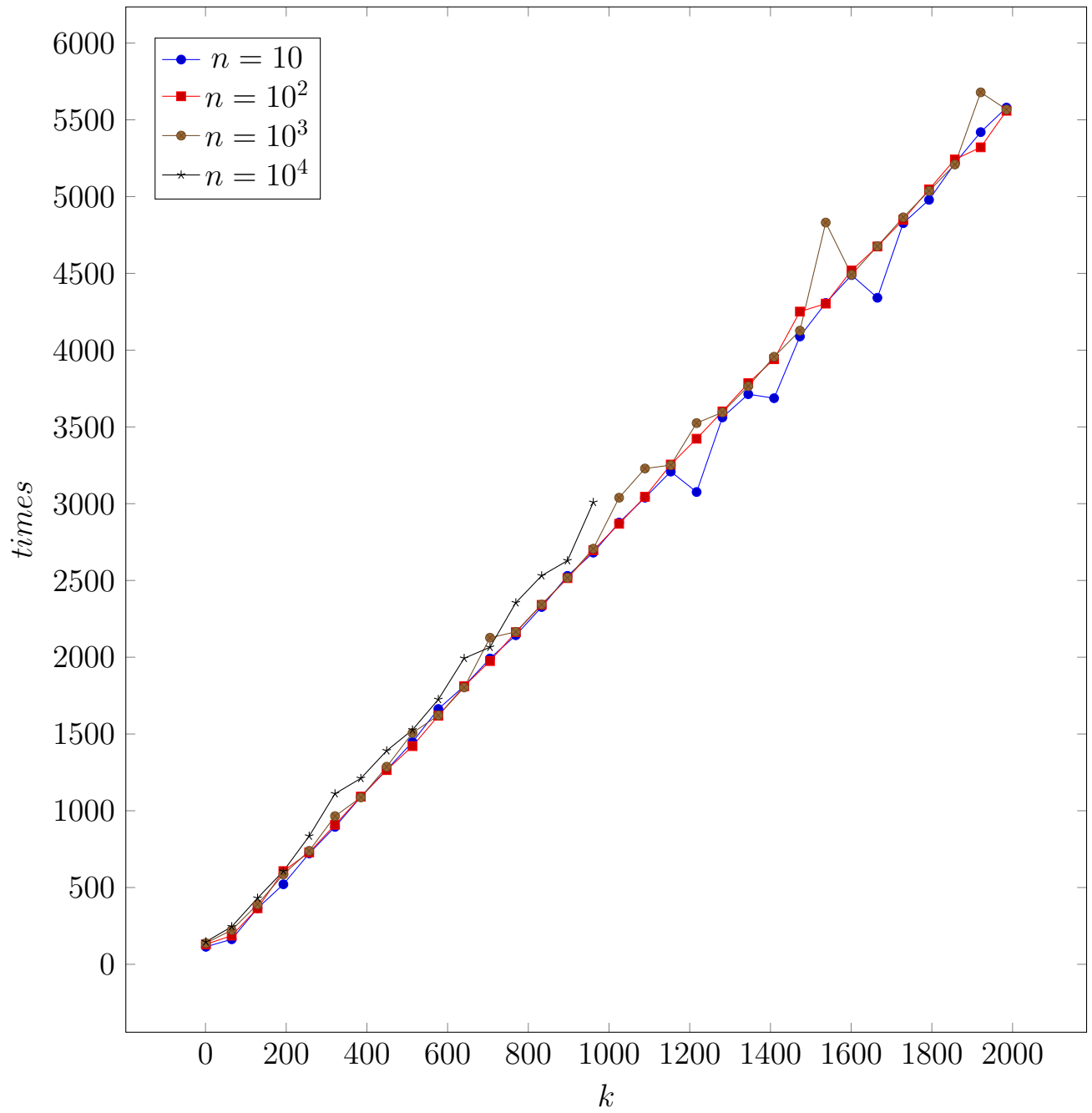
Заметим, что в методе градиентного спуска константа линейной скорости сходимости $q = 2/(l + L)$ не зависит от размерности пространства n , а только от собственных чисел матрицы A квадратичной формы, а следовательно для всех размерностей должны получиться схожие результаты, что мы как раз таки

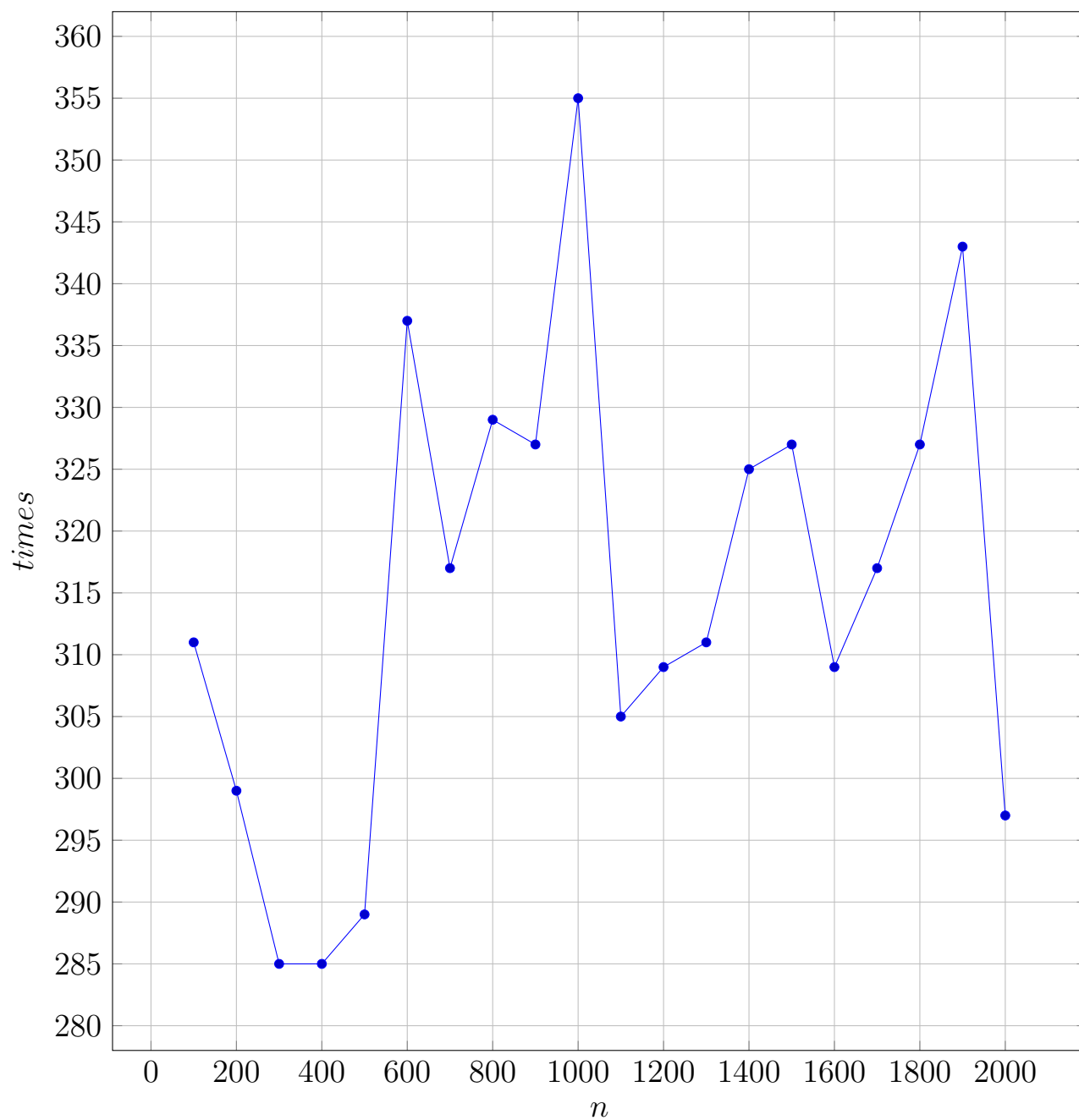
видим на графике ниже. Но есть несколько минимумов, которые выбиваются из общей массы. Скорее всего это из-за того, что сгенерированная точка попала в многомерный овраг и из-за этого не происходило сильных биений. По сгенерированным тестам это сложно понять.





1.5. Метод наискорейшего спуска





1.6. Метод сопряженных градиентов

