Федеральное государственное автономное
образовательное учреждение высшего образования
«Московский физико-технический институт
(национальный исследовательский университет)»
Физтех-школа прикладной математики и информатики

ЛЕКЦИИ И СЕМИНАРЫ ПО МЕТОДАМ ОПТИМИЗАЦИИ

Авторы:
А. Н. Безносиков, Е. Д. Бородич, Д. М. Двинских, Г. В. Кормаков,
Н. М. Корнилов, И. А. Курузов, А. В. Лобанов, Д. С. Метелев,
А. А. Моложавенко, С. А. Чежегов, Ю. И. Шароватова, Н. Е. Юдин,
Д. В. Ярмошик, А. В. Андреев, А. И. Богданов, Д. А. Былинкин,
Е. В. Рябинин, С. С. Ткаченко, А. А. Шестаков, Ю. И. Шароватова

МОСКВА
ФПМИ МФТИ
2023

Учебное пособие содержит конспект лекций и материалы семинарских занятий по курсу «Методы оптимизации». В таком виде курс был прочитан осенью 2023–2024 учебного года студентам 3-го курса физтех-школы прикладной математики и информатики МФТИ. Важно отметить, что материалы курса опираются на различные курсы методов оптимизации, прочитанные в России и за рубежом. А именно, авторы данного пособия хотели бы поблагодарить А. Бен-Тала, А. Г. Бирюкова, А. В. Гасникова, В. Г. Жадана, А. М. Катруцу, Д. А. Кропотова, А. С. Немировского, Ю. Е. Нестерова, Б. Т. Поляка, Ф. С. Стонякина, М. Такача, Р. Хильдебранда, М. Шмидта, М. Ягги за неоценимый вклад в преподавание и популяризацию численных методов оптимизации во всем мире.

РАБОТА НАД ПОСОБИЕМ ВЕДЕТСЯ В ДАННЫЙ МОМЕНТ! АВТОРЫ ПРИЗНАТЕЛЬНЫ ЗА ЛЮБЫЕ КОММЕНТАРИИ ПО ЕГО УЛУЧШЕНИЮ!

Содержание

1. Семинар X3. CVXPY

Демьян Ярмошик

CVXPY - опенсорсный "modeling language"для ряда задач оптимизации (выпуклые, целочисленно-выпуклые, квазивыпуклые и так далее) на Python, работающий в парадигме задач с известной структурой, то есть не "черный ящик". CVXPY не решает задачу, а позволяет записать её в удобном виде на Python, а потом компилирует и передаёт в солвер.

Замечание 1.1. Для задач не очень большой размерности ($10^3 - 10^4$ переменных) в среднем должен работать в 2-10 раз медленнее солвера, специально заточенного под задачу, но в 10+ раз быстрее программироваться.

Пример 1.2. Покажем, как можно получить решение прямой и двойственной задачи без:
- реализации функций, вычисляющих (суб)градиент/гессиан,
- построения двойственной задачи и выписывания сопряженных функций,
- выбора алгоритма оптимизации:

```
m, n = 15, 20
A, b = np.random.randn(m, n), np.random.randn(m)

x = cp.Variable(n)
f = cp.norm1(x)
constraints = [A @ x == b]
prob = cp.Problem(cp.Minimize(f), constraints)
prob.solve()

```

1.1. Disciplined Convex Programming

Disciplined Convex Programming (DCP) - теоретический фреймворк для описания задач выпуклой минимизации. Задачи, записанные в DCP, можно эффективно равносильными преобразованиями привести к стандартному (коническому: LP, SDP, ...) виду и передать в солвер.

Замечание 1.3. CVXPY требует, чтобы выпуклые задачи удовлетворяли правилам DCP. Преобразования делаются автоматически с использованием достаточных, но не необходимых условий выпуклости. Из-за этого иногда требуется предварительно привести задачу в DCP-форму.

Предложение 1.4. Общее правило выпуклой композиции. $h(f_1(x), ..., f_n(x))$ - выпуклая функция, если h - выпуклая (в совокупности) функция, и для каждого индекса i выполнено какое-то из условий:
• h - возрастающая функция i-го аргумента и f_i выпуклая;
• h - убывающая функция i-го аргумента и f_i вогнутая;
• f_i аффинная ($f_i(x) = a^\top x + b$).

Доказательство. □ Для определенности предположим, что f_i - выпуклые.

$$\vec{f}(\alpha x_1 + (1-\alpha)x_2) \leq \alpha\vec{f}(x_1) + (1-\alpha)\vec{f}(x_2),$$

где неравенство понимается покомпонентно. Воспользуемся монотонностью и выпуклостью h:

$$\begin{aligned} h(\vec{f}(\alpha x_1 + (1-\alpha)x_2)) \leq & h(\alpha\vec{f}(x_1) + (1-\alpha)\vec{f}(x_2)) \\ \leq & \alpha h(\vec{f}(x_1)) + (1-\alpha)h(\vec{f}(x_2)) \end{aligned}$$

Случай, когда f_i вогнутая, а h убывает по i-му аргументу, аналогичен рассмотренному выше. В случае, когда f_i-аффинные, первые два неравенства заменяются на равенства. ■

Замечание 1.5. Для вогнутой комбинации работает то же самое правило, нужно только заменить "выпуклая"на "вогнутая"в формулировке предложения.

Замечание 1.6. Стоит отслеживать знакоопределённость значений функций, так как монотонность может зависеть от знака аргумента: например, $\|x\|_1, \|x\|_2$ возрастают по x_i при $x_i \geq 0$ и убывают при $x_i \leq 0$.

Пример 1.7. Рассмотрим $f(x,y) = \frac{(x-y)^2}{1-\max(x,y)}, \quad x < 1, \ y < 1$ и проведем анализ по DCP:
• x, y, 1 - аффинные функции,
• $\max(x,y)$ - выпуклая функция, $x - y$ - аффинная,
• u^2/v - выпуклая функция, монотонно убывающая по v при $v > 0$, поэтому выпукла для $u = (x-y)$, $v = 1 - \max(x,y)$.
Таким образом, эта функция выпуклая.

Теперь мы готовы дать формальное определение:

Определение 1.8. Задача удовлетворяет правилам DCP, если она:
- имеет не более одной цели вида min{скалярное выпуклое выражение} или max{скалярное выпуклое выражение};
- имеет 0 или более ограничений вида {выпуклое выражение} $\leq$ {вогнутое выражение}, {вогнутое выражение} $\geq$ {выпуклое выражение} или {аффинное выражение} $=$ {аффинное выражение};
- выпуклость/вогнутость/аффинность может быть установлена по общему правилу выпуклой композиции.

Пример 1.9. $f(x) = \sqrt{1+x^2}$ - выпуклая функция, однако `CVXPY` выдаст ошибку, если мы передадим ее в таком виде. Квадратный корень - вогнутая возрастающая функция, $1+x^2$ - выпуклая функция. Применить общее правило выпуклой композиции не получается, поэтому $f(x)$ не удовлетворяет правилам DCP. Вместо этого, передадим $f(x) = \left\|\begin{pmatrix}1\\x\end{pmatrix}\right\|_2$ – композиция аффинной и выпуклой функции. Теперь $f(x)$ удовлетворяет DCP. Ниже приведен код решения задачи.

```
x = cp.Variable()
f = cp.norm(cp.hstack((1, x)))
cp.Problem(cp.Minimize(f)).solve()

```

Пример 1.10.

$$
\begin{aligned}
&\min_{x\in\mathbb{R}} x+2,\\
&\text{s.t. } 5 = 2/x,\\
&x > 0.
\end{aligned}
$$

В ограничение вида равенства входит неафинная функция, поэтому задача не удовлетворяет DCP. Введем $t = 1/x > 0$. Поскольку $x = 1/t$ – функция выпуклая при $t > 0$, получим задачу минимизации выпуклой функции с аффинными ограничениями.

```
t = cp.Variable()
prob = cp.Problem(cp.Minimize(cp.inv_pos(t) + 2), [5 ==
2 * t])
prob.solve()

```

Пример 1.11.

$$
\min_{x\in\mathbb{R}} x+2,
$$

$$\text{s.t. } 5 \leq 2/x^2.$$

Выпуклая функция записана справа от знака $\leq$ – задача не удовлетворяет DCP. Предлагается обе части неравенства умножить на x^2.

```
        x = cp.Variable()
        prob = cp.Problem(cp.Minimize(x + 2), [5 * x**2 <= 2])
        prob.solve()

```

1.2. Преобразование задач в CVXPY

Для начала – общая схема преобразования задач: • Для каждой функции-атома $f(x)$ из библиотеки (полный список) известно представление этой функции в виде решения некоторой конической задачи;
• CVXPY идёт по дереву арифметического выражения и заменяет функции на их конические представления;
• в результате получается задача из стандартного класса конических задач (LP, SDP, ...);
• в зависимости от того, к какому классу преобразовалась исходная задача, подбирается подходящий солвер и производятся финальное преобразование задачи к форме, определяемой API солвера;
• DCP гарантирует, что получившаяся задача эквивалентна исходной.

Для функций-атомов $f(x)$ в библиотеке имеется их коническое представление

$$\min_y c^\top x + d^\top y + e,$$
$$\text{s.t. } A\begin{pmatrix} x \\ y \end{pmatrix} = b, \quad \begin{pmatrix} x \\ y \end{pmatrix} \in K,$$

где K - один из стандартных конусов: $\mathbb{R}^n_+$ (в случае, если сводится к LP), конус второго порядка, экспоненциальный и так далее. Функции в задаче заменятся коническими представлениями: • Для каждого атома $f(x)$ добавляется переменная y,
• вхождения $f(x)$ в задачу заменяются на $c^\top x + d^\top y + e$,
• к текущему набору ограничений добавляются ограничения из конического представления $f(x)$.
В итоге получается задача с линейным функционалом и коническими и аффинными ограничениями.

Пример 1.12.

$$\min_{x\in\mathbb{R}^d} \max\{\|x\|_1, \|x\|_2^2\},$$
$$\text{s.t. } Ax = b.$$

Запишем конические представления атомов:

$$\max_{i=1,\dots,n} x_i = \min\{y \mid x_i \le y,\ i = 1,\dots,n,\ y \in \mathbb{R}\} \quad (\text{LP}),$$

$$\|x\|_1 = \sum_{i=1}^{d} |x_i|, \quad |x_i| = \max\{x_i, -x_i\} \quad (\text{LP}),$$

$$\|x\|_2^2 = \min\left\{t \mid \|(2x, t-1)^\top\|_2 \le t+1,\ t \ge 0\right\} \quad (\text{SOCP}).$$

Рекурсивно заменяя максимум, первую и вторую норму, получим:

$$\max\{\|x\|_1, \|x\|_2^2\} \to \quad \begin{array}{l} \min_{p,t,y} p \\ \quad \text{s.t. } \sum_{i=1}^d y_i \le p \\ x \le y \\ -x \le y \\ t \le p \\ \|(2x, t-1)^\top\|_2 \le t+1, \\ t \ge 0 \end{array} \quad .$$

Осталось добавить $\min_x$ и исходное аффинное ограничение $Ax = b$ - получим равносильное преобразование задачи в задачу из класса SOCP. Её можно далее привести к стандартному виду, как это делалось на семинаре про классы выпуклых задач.

```
m, n = 15, 20
A, b = np.random.randn(m, n), np.random.randn(m)

x = cp.Variable(n)
f = cp.maximum(cp.norm1(x), cp.sum_squares(x))
constraints = [A @ x == b]
prob = cp.Problem(cp.Minimize(f), constraints)
prob.solve()

```

Если посмотреть на код, можно заметить, что все обозначенные выше преобразования, которые мы делали руками, `CVXPY` делает автоматически.

В заключении еще раз отметим, что вместо знания редукций и ручной работы по их применению пользователю нужно заботиться только о приведении задачи к DCP-виду, которое, по сути, заключается в одном правиле выпуклой композиции.

1.3. Интерфейс CVXPY

Отметим некоторые ключевые аспекты интерфейса:
• Переменные могут быть скалярными, векторными или матричными. Более высокие размерности пока не поддерживаются, их нужно обрабатывать вручную:

```
    a = cp.Variable()

    x = cp.Variable(5)

    X = cp.Variable((5, 1))
```

• Параметры можно использовать вместо констант, если планируется решать задачу несколько раз при разных значениях параметров. При этом задача не будет перекомпилироваться. Также можно вычислять производные решения по значениям параметров.

```
    m = cp.Parameter(nonneg=True)

    c = cp.Parameter(5)

    G = cp.Parameter((4, 7), nonpos=True)

    G.value = -np.ones((4, 7))
```

• Ограничения определяются с помощью операторов $\leq$, $\geq$, $=$. Неравенства между векторами и матрицами понимаются покомпонентно. Нельзя использовать строгие неравенства (это нестрашно - на практике в них мало смысла). Сравнения по цепочке не поддерживаются.

• Чтобы решать задачи с дискретными переменными, достаточно выставить флаги `boolean` или `integer` при создании переменных:

```
    x = cp.Variable(10, boolean=True)
    Z = cp.Variable((5, 7), integer=True)
```

Напомним, CVXPY не решает задачу сам, а вызывает внешние солверы. Представим некоторые из них в таблице:

Solver	LP	QP	SOCP	SDP	EXP	POW	MIP
CLARABEL	+	+	+	+	+	+	
GLPK	+						
GLPK_MI	+						+
OSQP	+	+					
CPLEX	+	+	+				+
ECOS	+	+	+		+		
GUROBI	+	+	+				+
MOSEK	+	+	+	+	+	+	+
CVXOPT	+	+	+	+			
SCS	+	+	+	+	+	+	
SCIP	+	+	+				+
SCIPY	+						+

Солвер, вызываемый `CVXPY`, можно изменить, используя параметр `solver` метода `problem.solve`.

Замечание 1.13. По умолчанию `CVXPY` вызывает солвер, наиболее специализированный для данного типа задачи. Например, для SOCP вызывается ECOS. SCS может решать все задачи (кроме смешанных целочисленных программ). Для QP `CVXPY` использует OSQP. Дополнительные солверы поддерживаются, но должны устанавливаться отдельно.

Замечание 1.14. У разных солверов параметры (например, максимальное число итераций) называются по-разному.

Замечание 1.15. Если в процессе `problem.solve()` выяснится, что задача несовместна или неограниченна, в поле `problem.status` выставятся значения `infeasible` или `unbounded` соответственно.

1.4. Задача 1. Поиск минимального эллипсоида

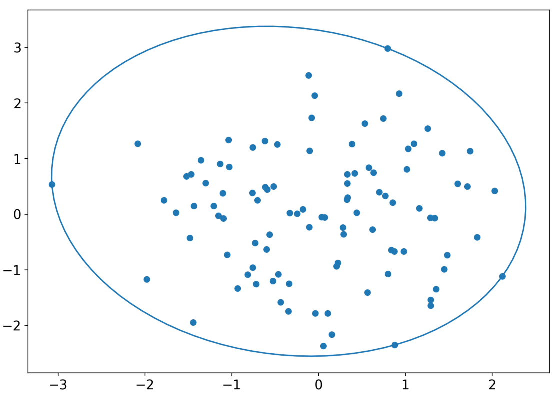
Рассмотрим эллипсоид, заданный при помощи матрицы $X \in \mathbb{S}^n_{++}$ как $E = \left\{a \in \mathbb{R}^n \middle| \|Xa + b\| \leq 1\right\}$. Эквивалентно, его можно определить как прообраз шара под действием отображения, задаваемого матрицей X: $E = \left\{a \middle| Xa \in B_1(0)\right\}$. В таком случае мы можем вычислить его объем:

$$V_E = \iiint\limits_{x \in E} dx = \iiint\limits_{x | Xx \in B_1(0)} dx = \iiint\limits_{y \in B_1(0)} \det \frac{\partial X^{-1} y}{\partial y} dy = \det(X^{-1}) V_{B_1(0)}.$$

Теперь рассмотрим задачу построения минимального по объему эллипсоида, такого, что точки $\{a_i\}_{i=1}^m$ лежат в нем, т.е. $\|Xa_i\| \leq 1$. Минимизация объема V_E эквивалентна минимизации $\log V_E = \log\det(X^{-1}) + \log V_{B_1(0)} = -\log\det X + c$. Тогда, опуская аддитивную константу, получаем задачу минимизации:

$$\min_{X \in \mathbb{S}^n_{++}} \; -\log\det X,$$
$$\text{s.t. } \|Xa_i + b\| \leq 1, i = \overline{1, m}.$$

```
n, m = 2, 100
A = np.random.randn(m, n)

X = cp.Variable((n, n))
b = cp.Variable(n)

obj = cp.Minimize(-cp.log_det(X))

constraints = [cp.norm(X @ a + b) <= 1 for a in A]

problem = cp.Problem(obj, constraints)
_ = problem.solve(verbose=True)

L = np.linalg.inv(X.value)
plt.figure(figsize=(10, 7))
plt.scatter(A[:, 0], A[:, 1])
phi = np.linspace(0, 2 * np.pi, num=100)
xy = np.vstack((np.cos(phi), np.sin(phi))) - b.value.reshape(-1, 1)
ellips = L.dot(xy)
plt.plot(ellips[0, :], ellips[1, :])
```



1.5. Задача 2. Встреча двух тел

Два тела двигаются с трением на плоскости. Даны их начальные координаты и скорости, можно управлять их ускорениями. Нужно найти такие ускорения для обоих тел, чтобы их конечные состояния были одинаковыми и было потрачено суммарно минимум энергии. Дискретизуем уравнения динамики линейной системы для одного тела:

$$
\begin{aligned}
&x_{t+1} = Ax_t + Bu_t,\\
&m\frac{v_{t+1} - v_t}{\tau} \approx -\eta v_t + u_t, \quad \frac{p_{t+1} - p_t}{\tau} \approx v_t,
\end{aligned}
$$

где x_t - общий вектор состояния, состоящий из пространственных координат $p_t = (p_t^x, p_t^y)$, вектора скорости $v_t = (v_t^x, v_t^y)$, u_t - вектор ускорения, η - коэффициент трения, τ - квант времени. Приближённо имеем следующую систему:

$$
\begin{cases}
v_{t+1} = \left(1 - \frac{\tau}{m}\eta\right) v_t + \frac{\tau}{m} u_t,\\
p_{t+1} = p_t + \tau v_t.
\end{cases}
$$

Перепишем в стандартной форме:

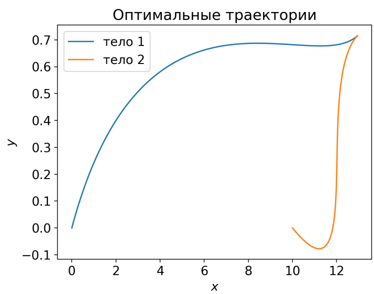
$$
x_t = \begin{bmatrix} p_t^x \\ p_t^y \\ v_t^x \\ v_t^y \end{bmatrix}, \quad A = \begin{bmatrix} 1 & 0 & \tau & 0 \\ 0 & 1 & 0 & \tau \\ 0 & 0 & 1 - \frac{\tau}{m}\eta & 0 \\ 0 & 0 & 0 & 1 - \frac{\tau}{m}\eta \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \frac{\tau}{m} & 0 \\ 0 & \frac{\tau}{m} \end{bmatrix}.
$$

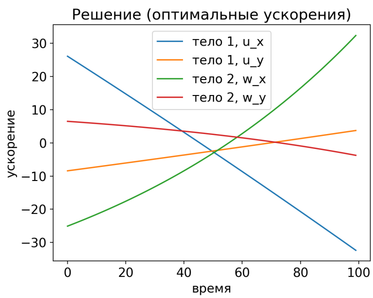
Будем считать, что затраты энергии пропорциональны сумме квадратов ускорений. Тогда задача встречи двух тел формулируется следующим образом:

$$
\begin{aligned}
&\min_{\substack{u, w \in \mathbb{R}^{2\times T} \\ x, z \in \mathbb{R}^{2\times(T+1)}}} \sum_{i=1}^{T} \|u_i\|_2^2 + \|w_i\|_2^2,\\
&\text{s.t. } x_{t+1} = Ax_t + Bu_t,\ t = 1, \ldots, T,\\
&z_{t+1} = Cz_t + Dw_t,\ t = 1, \ldots, T,\\
&x_T = z_T.
\end{aligned}
$$

```
x, z = cp.Variable((4, T + 1)), cp.Variable((4, T + 1))
u, w = cp.Variable((2, T)), cp.Variable((2, T))
```

```
    cost = cp.sum_squares(u) + cp.sum_squares(w)
    constraints = [x[:, 1:] == A @ x[:, :-1] + B @ u, z[:, 1:] == C @ z[:, :-1] + D @ w]
    constraints += [x[:, T] == z[:, T], x[:, 0] == x0, z[:, 0] == z0]

    prob = cp.Problem(cp.Minimize(cost), constraints)
    prob.solve(verbose=0)
```





Литература

1. Жадан В. Г. Методы оптимизации. Часть I. Введение в выпуклый анализ и теорию оптимизации. Москва : МФТИ, 2014. 271 с.
2. Иванов Г.Е. Лекции по математическому анализу. Часть 1. Москва : МФТИ, 2017. 340 с.
3. Иванов Г.Е. Лекции по математическому анализу. Часть 2. Москва : МФТИ, 2016. 191 с.
4. Нестеров Ю.Е. Методы выпуклой оптимизации. Москва : Изд-во МЦНМО, 2010. 281 с.
5. Boyd S., Vandenberghe L. Convex Optimization. Cambridge : Cambridge University Press, 2004. 716 с.
6. Cauchy A. Méthode générale pour la résolution des systémes d'équations simultanées. Paris : Comptes rendus hebdomadaires des séances de l'Académie des sciences, vol. 55, 1847. 536–538 с.