In [15]:
```
import functools

import cvxpy as cvx
import numpy as np
import cvxopt

import matplotlib.pyplot as plt
from matplotlib import collections  as mc
import seaborn as sns

plt.rc("text", usetex=True)

%matplotlib notebook
%autosave 15
```

```
Autosaving every 15 seconds
```

Задача 1

Для следующей задачи

$$\begin{aligned} & \min_{\mathbf{x} \in \mathbb{R}^n} \frac{1}{2}\mathbf{x}^\top \mathbf{x} + \mathbf{c}^\top \mathbf{x} \\ & \text{s.t. } \mathbf{A}\mathbf{x} \leq \mathbf{b} \end{aligned}$$

где $\mathbf{A} \in \mathbb{R}^{m \times n}$ — плотная матрица, предложить эффективный метод решения системы в методе Ньютона на шаге барьерного метода.
Рассмотреть два случая: $m \ll n$ и $n \ll n$.

Решение: Вначале вспомним, где, как и почему в барьерном методе возникает необходимость использовать метод Ньютона.
Имеем дело с общей задачей выпуклой оптимизации:

$$\begin{aligned} &\min_{x\in\mathbb{R}^n} f_0(x) \\ \text{s.t. } &f_i(x) \leq 0 \qquad i = 1, \ldots, m \end{aligned}$$

Где $f_i(x) = A_{(i,\,:)^\top} x$ — скалярное произведение i-ой строки матрицы A на вектор x.
Избавимся от ограничений типа неравенств с помощью индикаторов, переходя к задаче неограниченной оптимизации:

$$\begin{aligned} \min\ \Psi(x) &:= f_0(x) + \sum_{i=1}^{m} -t\log(-f_i(x)) \\ \text{s.t. } Ax = b, \end{aligned}$$

- Задача по-прежнему **выпуклая**
- Функция

$$\phi(x) = -\sum_{i=1}^{m} \log(-f_i(x)) = -\sum_{i=1}^{m} \log(-(Ax-b))_i$$

называется *логарифмическим барьером*. Её область определения - множество точек, где ограничения типа неравенств выполняются строго.

Вспомним, какой вид имеют градиент и гессиан барьерной функции:

$$\nabla\phi(x) = -\sum_{i=1}^{m} \frac{\nabla f_i(x)}{f_i(x)} \quad \nabla^2\phi(x) = \sum_{i=1}^{m} \frac{\nabla f_i(x)\nabla f_i(x)^\top}{f_i(x)^2} - \frac{\nabla^2 f_i(x)}{f_i(x)}$$

$$\nabla f_i(x) = A_{(i,:)} \quad \nabla^2 f_i(x) = 0$$

$$\nabla\phi(x) = \sum_{i=1}^{m} \frac{A_{(i,:)}}{(b-Ax)_i} = A^\top \text{diag}^{-1}(b-Ax)\cdot \mathbf{1}^m$$

$$\begin{aligned} \nabla^2\phi(x) = \sum_{i=1}^{m} \frac{A^\top_{(i,:)}}{(b-Ax)_i}\frac{A_{(i,:)}}{(b-Ax)_i} &= (A^\top \text{diag}^{-1}(b-Ax))(A^\top \text{diag}^{-1}(b-Ax))^\top \\ &= A^\top(\text{diag}^{-2}(b-Ax))A \end{aligned}$$

Я постарался максимально векторизовать формулы, но пока непонятно, станет ли от этого легче.
Т.к. целевая функция выпуклая, будем искать её минимум среди стационарных точек, которые найдём методом Ньютона:

$$\begin{aligned}
&\Psi'(x^*(t)) = f_0'(x^*(t)) + t\phi'(x^*(t)) = \\
&= x^*(t) + \mathrm{c} + t\sum_{i=1}^{m}\frac{A_{(i,:)}}{(b-Ax)_i} = \\
&= x^*(t) + \mathrm{c} + t\cdot A^\top \mathrm{diag}^{-1}(b-Ax)\cdot \mathbf{1}^m
\end{aligned}$$

$$\begin{aligned}
&\Psi''(x^*(t)) = f_0''(x^*(t)) + t\phi''(x^*(t)) = \\
&= I_n + t\sum_{i=1}^{m}\frac{A_{(i,:)}A_{(i,:)}^\top}{(b-Ax)_i^2} = \\
&= I_n + t\ \cdot A^\top(\mathrm{diag}(b-Ax))^{-2}A
\end{aligned}$$

Шаг метода Ньютона для градиента запишется так:

$$\begin{aligned}
&x_{n+1}(t) = x_n(t) - (\Psi''(x(t)))^{-1}\Psi'(x(t)) = \\
&= x_n(t) - \left(I_n + t\ \cdot A^\top(\mathrm{diag}(b-Ax_n(t)))^{-2}A\right)^{-1}\cdot\left(x_n(t) + \mathrm{c} + t\cdot A^\top \mathrm{diag}^{-1}(b-Ax_n(t))\cdot\mathbf{1}^m\right)
\end{aligned}$$

Теперь его вычисление нужно как-то разумно оптимизировать в зависимости от структуры матрицы A.
Вид гессиана кажется подозрительно знакомым, и это неспроста: для его обращения можно применить формулу Шермана-Моррисона-Вудберри, которую мы доказывали в домашнем задании!

Для краткости введём обозначение $\Lambda := \mathrm{diag}(\mathbf{b} - \mathbf{A}\mathbf{x}_n(t))$.

$$\begin{gathered}
(\mathbf{A}+\mathbf{UCV})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{U}(\mathbf{C}^{-1}+\mathbf{V}\mathbf{A}^{-1}\mathbf{U})^{-1}\mathbf{V}\mathbf{A}^{-1} \\
\mathbf{A}\in\mathbb{R}^{n\times n},\ \mathbf{U}\in\mathbb{R}^{n\times m},\ \mathbf{C}\in\mathbb{R}^{m\times m},\ \mathbf{V}\in\mathbb{R}^{m\times n} \\
\mathbf{A} = \mathbf{I}_n,\ \mathbf{U} = \mathbf{A}^\top,\ \mathbf{C} = t\cdot\Lambda^{-2},\ \mathbf{V} = \mathbf{A} \\
\left(I_n + \ \cdot A^\top t\cdot\Lambda^{-1}A\right)^{-1} = \mathbf{I}_n - \mathbf{A}^\top(t^{-1}\Lambda^2 + \mathbf{A}\mathbf{A}^\top)^{-1}\mathbf{A}
\end{gathered}$$

Всегда ли она позволяет улучшить асимптотику по сравнению с обычным методом Гаусса?
Рассмотрим два случая и в каждом явно вычислим асимптотику каждого шага:

1. $m \ll n$:

- $t^{-1}\Lambda^2 — O(m)$
- $\mathbf{A}\mathbf{A}^\top — O(m^2 n)$
- $t^{-1} \cdot \Lambda^2 + \mathbf{A}\mathbf{A}^\top — O(m^2)$
- $(t^{-1}\Lambda^2 + \mathbf{A}\mathbf{A}^\top)^{-1} — O(m^3)$
- $\mathbf{I}_n + \mathbf{A}^\top(t^{-1}\Lambda^2 + \mathbf{A}\mathbf{A}^\top)^{-1}\mathbf{A} — O(nm^2 + n^2 m) = O(n^2 + n^2 m)$
- $\left(\mathbf{x}_n(t) + \mathbf{c} + t \cdot \mathbf{A}^\top \Lambda^{-1} \cdot \mathbf{1}^m\right) — O(nm)$
- $\left(\mathbf{I}_n + \mathbf{A}^\top(t^{-1}\Lambda^2 + \mathbf{A}\mathbf{A}^\top)^{-1}\mathbf{A}\right)\left(\mathbf{x}_n(t) + \mathbf{c} + t \cdot \mathbf{A}^\top \Lambda^{-1} \cdot \mathbf{1}^m\right) — O(n^2)$

Итоговая асимптотика — $O(n^2 m)$

2. $n \ll m$

- Формула Шермана-Моррисона-Вудберри даёт асимптотику $O(m^3)$. Попробуем обратить напрямую.
- $A^\top t \cdot \Lambda^{-1} A — O(nm^2 + m^2 n)$
- $I_n + \ \cdot A^\top t \cdot \Lambda^{-1} A — O(n^2)$
- $\left(I_n + \ \cdot A^\top t \cdot \Lambda^{-1} A\right)^{-1} — O(n^3)$

Итоговая асимптотика — $O(m^2 n)$

Вывод: один шаг метода Ньютона осуществим за $O(H^2 L)$, где $H := \max\{n, m\},\ L := \min\{n, m\}$.
Подбирать метод следует на основе структуры матрицы A.

In [125]:
```
%%time
m = 7
n = 11
for _ in range(10000):
    A = np.random.rand(m, n)
    b = np.random.rand(m, 1)
    x = np.random.rand(n, 1)
    r = b - A @ x
    assert np.allclose(np.sum(np.column_stack([A[i, :] / r[i] for i in range(m)]), axis=1),
                       np.ravel(A.T @ np.linalg.inv(np.diagflat(r)) @ np.ones((m, 1))))
    assert np.allclose(np.sum(np.dstack([A[i, :].reshape(n, 1) @ A[i, :].reshape(1, n) / (r[i]**2)
                                         for i in range(m)]), axis=-1),
                       (A.T @ np.linalg.inv(np.diagflat(r))) @ ((A.T @ np.linalg.inv(np.diagflat(r)))).T)
```

```
CPU times: user 10.9 s, sys: 18 s, total: 28.8 s
Wall time: 4.26 s
```

Задача 2

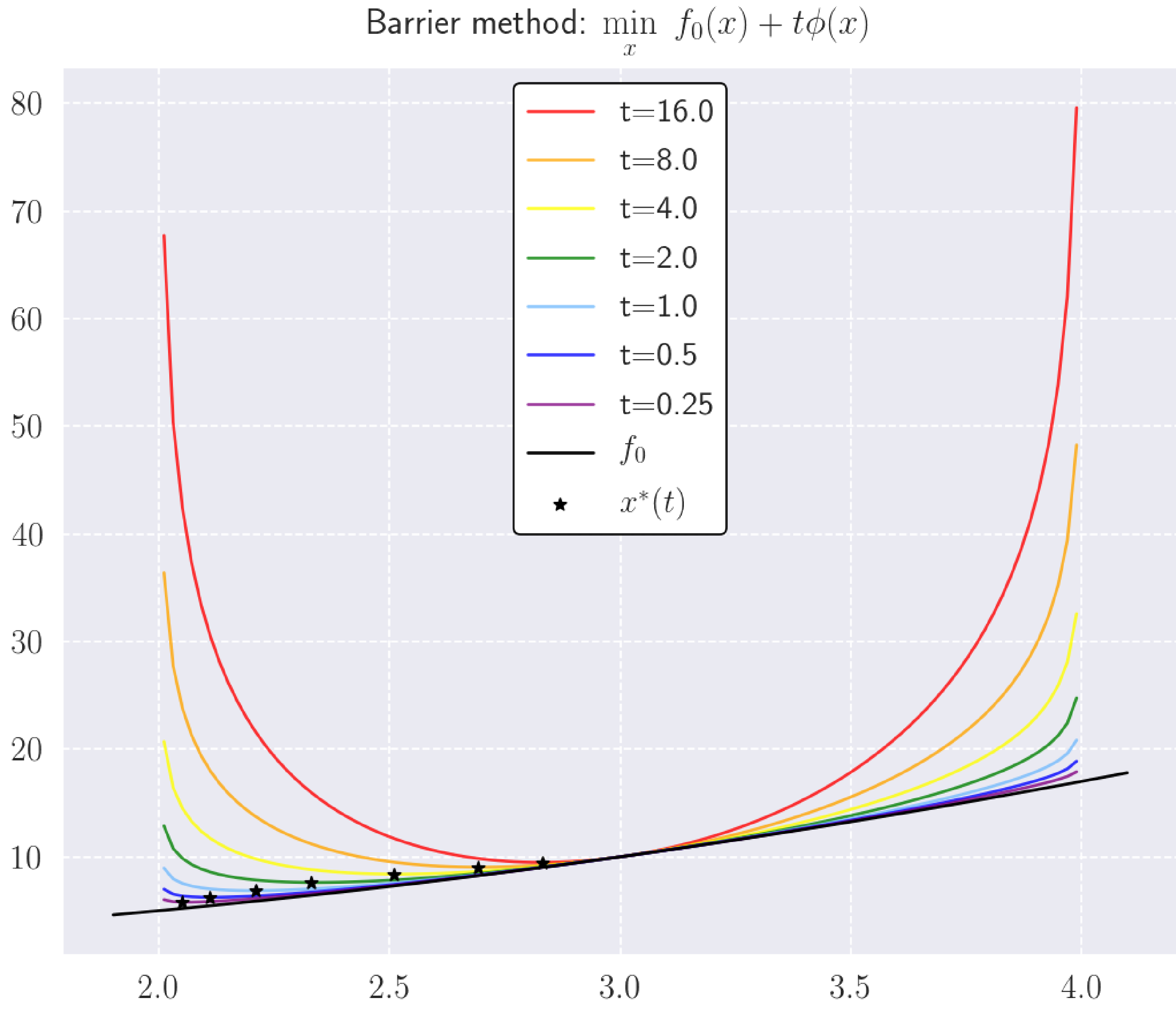
Для задачи

$$\min_{\mathbf{x}\in\mathbb{R}} x^2 + 1$$
$$\text{s.t. } 2 \leq x \leq 4$$

найти аналитически решение x^*. Нарисовать на одном графике $f_0(x)$ и $f_0(x) + t\phi(x)$ для 4 значений t > 0, которые монотонно убывают к 0. Покажите на графике $x^*(t)$ для каждого значения t.

Задача выпуклая, стационарная точка единственная и лежит в допустимой области: $x^* = 2$.
Визуализация:

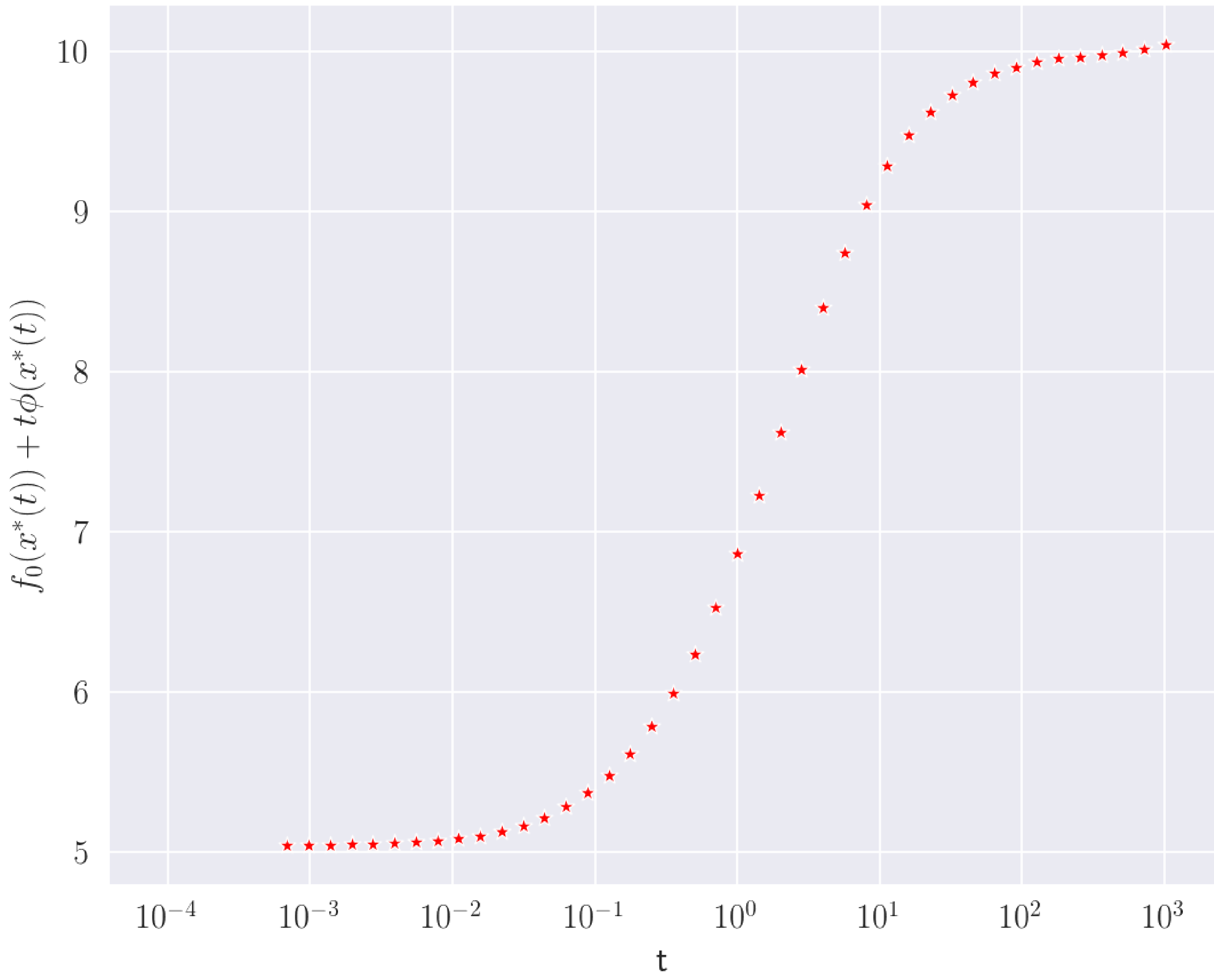
In [55]:
```
def objective(x, t):
    return x**2 + 1 - t * (np.log(x - 2) + np.log(4 - x))

sns.set(font_scale=1.5, style="darkgrid")
plt.figure(figsize=(10, 8))

stars_x = []
stars_f = []

t_list = 2. ** np.arange(4, -3, -1)
colors = ['red', 'orange', 'yellow', 'green',
          'xkcd:sky blue', "blue", "purple"]
x_ast_t = []
grid = np.arange(2.01, 3.99, 0.02)

for t, c in zip(t_list, colors):
    vals = functools.partial(f, t=t)(grid)
    argmin = np.argmin(vals)
    x_ast_t.append([grid[argmin], vals[argmin]])
    plt.plot(grid, vals, color=c, label='t={}'.format(t), alpha=0.8, zorder=1)

grid = np.linspace(1.9, 4.1, 1000)
x_ast_t = np.vstack(x_ast_t)

plt.plot(grid, grid**2 + 1, color = 'black', alpha=1, label=r'$f_0$')
plt.scatter(x_ast_t[:, 0], x_ast_t[:, 1], label=r"$x^{*}(t)$",
            marker='*', c='black', s=40, zorder=2)
plt.grid(linestyle='--')

legend = plt.legend()
frame = legend.get_frame()
frame.set_facecolor("white")
frame.set_edgecolor("black")
frame.set_alpha(1)
plt.title(r'Barrier method: $\min\limits_{x}\ f_0(x) + t\phi(x)$')
plt.show()
```



Barrier method: $\min_x \; f_0(x) + t\phi(x)$
t=16.0
t=8.0
t=4.0
t=2.0
t=1.0
t=0.5
t=0.25
f_0
$x^*(t)$
80
70
60
50
40
30
20
10
2.0
2.5
3.0
3.5
4.0

In [95]:

```
grid = np.arange(2.01, 3.99, 0.02)
t = 2. ** np.arange(10, -11, -0.5)
vals = f(grid, t.reshape(-1, 1))
argmins = np.argmin(vals, axis=1)
plt.figure(figsize=(10, 8))
sns.scatterplot(t, [vals[i, argmins[i]] for i in range(vals.shape[0])],
                marker="*", s=80, color="red")
plt.xscale("log")
plt.xlabel("t");
plt.ylabel(r"$f_{0}(x^*(t)) + t \phi(x^*(t))$");
```



10
9
8
7
6
5
$f_0(x^*(t)) + t\phi(x^*(t))$
10^{-4}
10^{-3}
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}
10^{3}
t

Задача 3

Вместо логарифмического барьера можно использовать другие функции, аппроксимирующие $I_-(x)$. Предлагается проанализировать использование барьерной функции вида $h(u) = -\frac{1}{u}$

1. Проверьте, будет ли выпукла «ограниченная» задача для барьера $h(u)$
2. Получите выражения для двойственных переменных $\lambda^*(t)$ и $\mu^*(t)$ и зазора двойственности при использовании барьера $h(u)$
3. Какими свойствами должна обладать барьерная функция, чтобы зазор двойственности зависел только от m и t?

1. Барьер $h(u) = -\frac{1}{u}, u \leqslant 0 \downarrow$ и выпуклый $\implies$ $h(f_i)$ выпукла.

$$f_0(x) + t \cdot \phi(x) = f_0(x) + t \sum_{i=1}^{m} h(f_i(x)) = f_0(x) - t \sum_{i=1}^{m} \frac{1}{f_i(x)}$$

выпукла как сумма выпуклых.
Все нелинейные ограничения уходят в целевую функцию, допустимое множество в ограниченной задаче выпукло (т.к. линейно).

1. Пусть $x^*(t)$ — оптимальное решение исходной задачи. Разрешимость прямой задачи:

$$Ax^*(t) = b,\ f_i(x^*) < 0,\ i = 1, \ldots, m$$

Стационарность лагранжиана

$$\begin{aligned}
\mathcal{L}(x(t), \mu) &= \nabla f_0(x) + t \cdot \phi(x) + \mu^\top (Ax - b) \\
\exists \widehat{\mu} > 0 : \nabla \mathcal{L}(x^*(t), \widehat{\mu}) &= \nabla f_0(x^*(t)) + \nabla \phi(x^*(t)) + A^\top \widehat{\mu} = \\
&= \nabla f_0(x^*(t)) + t \cdot \sum_{i=1}^{m} h'(f_i(x^*(t))) \nabla f_i(x^*(t)) + A^\top \widehat{\mu} = 0 \\
\lambda_i^* &= t \cdot \frac{\partial\, h(f_i(x^*(t)))}{\partial\, f_i} = \frac{t}{f_i^2(x^*(t))},\ \ \mu^* = \widehat{\mu} \\
f_0(x^*(t)) &= \sum_{i=1}^{m} f_i(x^*(t)) \lambda_i^* + (\mu^*)^\top (Ax - b)
\end{aligned}$$

Соотв. значение двойственной функции (тоже с семинара):

$$g(\lambda^*(t), \mu^*) = f_0(x^*(t)) + \sum_{i=1}^{m} \lambda_i^*(t)\ f_i(x^*(t)) + (\mu^*)^\top (Ax^*(t) - b)$$

$x^*(t)$ допустимо $\implies$ $Ax^*(t) = b,\ f_i(x^*(t)) < 0$, тогда зазор двойственности принимает вид:

$$f_0(x^*(t)) - g(\lambda^*(t), \mu^*) = -\sum_{i=1}^{m} \frac{t}{f_i(x^*(t))}$$

$$-\sum_{i=1}^{m} \lambda_i^* \cdot f_i(x^*(t)) = \sum_{i=1}^{m} \frac{-t}{f_i(x^*(t))} \geqslant 0$$

1. Убрать зависимость от всего, кроме t и m, можно только если $-uh'(u) = const$, т.е.

$$h'(u) = -\frac{C_1}{u}$$

$$h(u) = C_1 \ln(-u) + C_2$$

При $C_1 < 0$ барьер будет выпуклым.

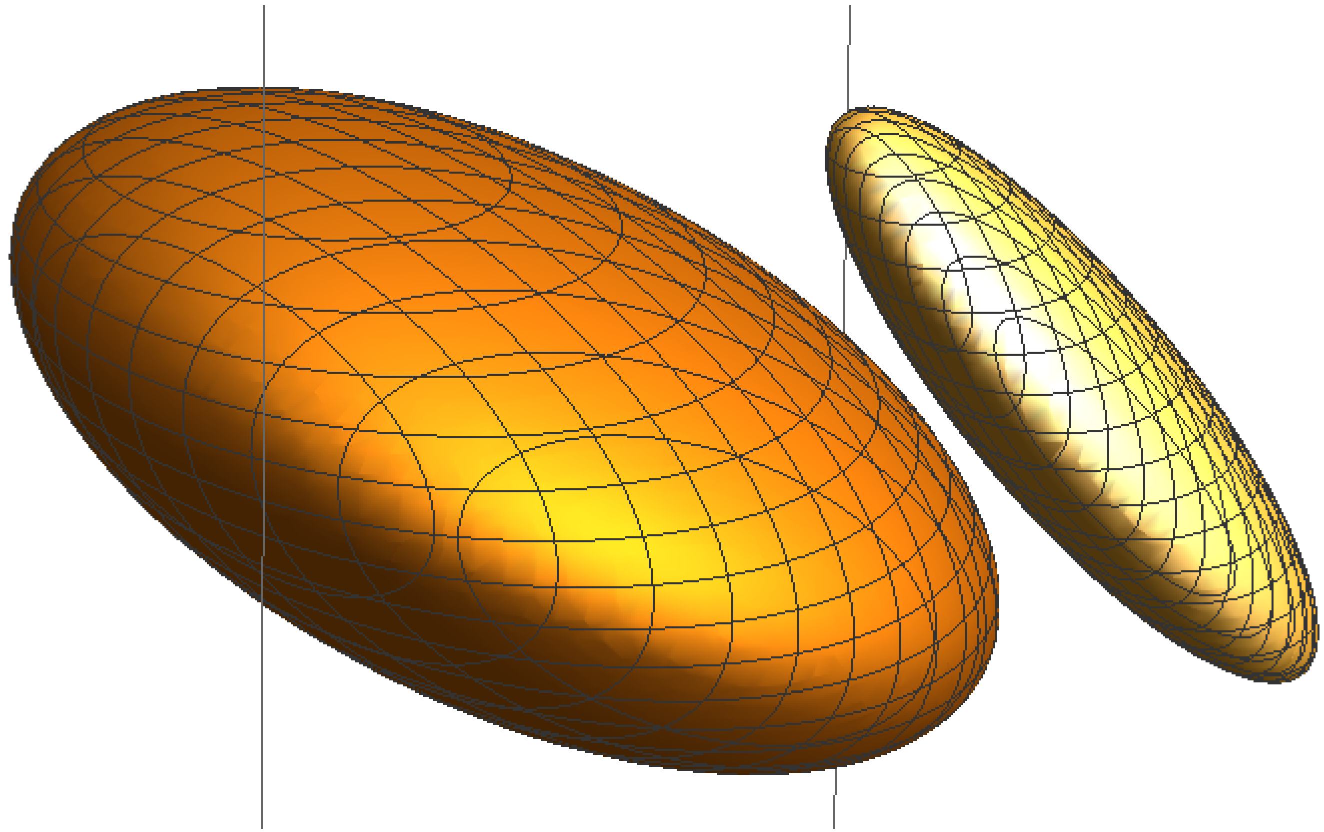
Задача 4:

Проверить, пересекаются ли два заданных эллипсоида, сформулировав задачу оптимизации в терминах SDP.

$$E_0 = \left\{ \mathbf{x} \in \mathbb{R}^3 \;\middle|\; \mathbf{x}^\top \begin{pmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 13/36 & 13/36 \\ 1/9 & 13/36 & 49/36 \end{pmatrix} \mathbf{x} \leq 1 \right\}$$

$$E_1 = \left\{ \mathbf{x} \in \mathbb{R}^3 \;\middle|\; \mathbf{x}^\top \begin{pmatrix} \sqrt{2} & \sqrt{2} & \sqrt{2} \\ \sqrt{2} & 2\sqrt{2} & 2\sqrt{2} \\ \sqrt{2} & 2\sqrt{2} & 1+2\sqrt{2} \end{pmatrix} \mathbf{x} + \begin{pmatrix} -6\sqrt{2} \\ -10\sqrt{2} \\ -10\sqrt{2} \end{pmatrix}^\top \mathbf{x} \leq 1 - 13\sqrt{2} \right\}$$

Решение: К счастью, задача допускает визуализацию, потому ответ можно узнать в самом начале, построив график в любом матпакете:



Видим, что эллипсоиды не пересекаются. Естественно, уже в размерности 4 строить графиик затруднительно, зато можно решать задачу оптимизации. Теперь нужно её грамотно сформулировать:

Сформулируем вначале задачу в самом простом виде: с квадратичной целевой функцией и квадратичными же ограничениями. Её можно решать, к примеру, барьерным методом. Вид, в котором заданы фигуры, намекает на такую постановку задачи:

$$\min_{\mathbf{x} \in \mathbb{R}^3} \mathbf{x}^\top \mathbf{A}_0 \mathbf{x} + 2\mathbf{b}_0^\top \mathbf{x} + \mathbf{c}_0$$
$$\text{s.t. } \mathbf{x}^\top \mathbf{A}_1 \mathbf{x} + 2\mathbf{b}_1^\top \mathbf{x} + \mathbf{c}_1 \leq 0$$

Где в целевой функции и ограничениях записаны квадратичные формы, задающие поверхность эллипса.
Очевидно, что минимум будет ≤ 0 тогда и только тогда, когда эллипсоиду пересекаются: если точка принадлежит эллипсоиду, то ассоциированная кадратичная форма ≤ 0 в этой точке. Для одного это условие закодировано в ограничениях, потому должно выполняться для второго.
Скормим задачу `cvxpy` и посмотрим, что получится.

In [19]:
```
E0, E1 = {}, {}
E0["A"] = np.matrix(
    [[1/9, 1/9, 1/9],
     [1/9, 13/36, 13/36],
     [1/9, 13/36, 49/36]]
)
E0["b"] = np.matrix("0; 0; 0")
E0["c"] = -1
E1["A"] = np.matrix(
    [[np.sqrt(2), np.sqrt(2), np.sqrt(2)],
     [np.sqrt(2), 2 * np.sqrt(2), 2 * np.sqrt(2)],
     [np.sqrt(2), 2 * np.sqrt(2), 2 * np.sqrt(2) + 1]]
)
E1["b"] = np.matrix(
    [[-3 * np.sqrt(2)],
     [-5 * np.sqrt(2)],
     [-5 * np.sqrt(2)]]
)
E1["c"] = 13 * np.sqrt(2) - 1
```

In [20]:
```
x = cvx.Variable(3)
A0 = cvxopt.matrix(E0["A"])
b0 = cvxopt.matrix(E0["b"])
c0 = cvx.Constant(E0["c"])
A1 = cvxopt.matrix(E1["A"])
b1 = cvxopt.matrix(E1["b"])
c1 = cvx.Constant(E1["c"])
quad_obj = cvx.quad_form(x, A0) + 2 * b0.T * x + c0
quad_constr = [cvx.quad_form(x, A1) + 2 * b1.T * x + c1 <= 0]
quad_prob = cvx.Problem(cvx.Minimize(quad_obj), quad_constr)
```

In [21]:
```
quad_prob.solve(verbose=True)
```

```
ECOS 2.0.4 - (C) embotech GmbH, Zurich Switzerland, 2012-15. Web: www.embotech.com/ECOS

It     pcost       dcost      gap   pres   dres    k/t    mu     step   sigma     IR    |   BT
 0  +2.559e-17  +1.701e+00  +2e+01  3e-01  3e-01  1e+00  3e+00    ---    ---    1  1  - |  -  -
 1  +8.553e-02  +4.703e-01  +3e+00  5e-02  4e-02  3e-01  6e-01  0.8076  3e-02   1  1  1 |  0  0
 2  +6.054e-01  +8.893e-01  +1e+00  2e-02  2e-02  2e-01  2e-01  0.9106  3e-01   2  2  2 |  0  0
 3  +1.018e+00  +1.063e+00  +8e-02  3e-03  2e-03  4e-02  2e-02  0.9418  3e-02   2  2  2 |  0  0
 4  +1.103e+00  +1.118e+00  +2e-02  8e-04  5e-04  1e-02  4e-03  0.9372  2e-01   2  2  2 |  0  0
 5  +1.129e+00  +1.130e+00  +2e-03  7e-05  4e-05  1e-03  4e-04  0.9213  1e-03   2  1  1 |  0  0
 6  +1.131e+00  +1.131e+00  +2e-04  9e-06  5e-06  1e-04  4e-05  0.9064  4e-02   2  1  1 |  0  0
 7  +1.132e+00  +1.132e+00  +9e-06  4e-07  2e-07  5e-06  2e-06  0.9611  3e-03   3  1  1 |  0  0
 8  +1.132e+00  +1.132e+00  +7e-07  3e-08  2e-08  3e-07  1e-07  0.9890  7e-02   2  1  1 |  0  0
 9  +1.132e+00  +1.132e+00  +7e-08  3e-09  2e-09  3e-08  1e-08  0.9106  6e-03   3  1  1 |  0  0
10  +1.132e+00  +1.132e+00  +1e-08  5e-10  3e-10  6e-09  3e-09  0.8051  5e-03   1  1  1 |  0  0
11  +1.132e+00  +1.132e+00  +4e-09  2e-10  9e-11  2e-09  9e-10  0.7628  9e-02   2  1  1 |  0  0

OPTIMAL (within feastol=2.0e-10, reltol=3.8e-09, abstol=4.3e-09).
Runtime: 0.000434 seconds.
```

Out[21]: 0.13158339105842987

`cvxpy` подтверждает: эллипсоиды не пересекаются. Теперь переформулируем задачу на языке SDP, как на семинаре:

- Рассмотрим такую задачу

$$\begin{aligned} \min f_0(x) \\ \text{s.t. } f_1(x) \le 0, \end{aligned}$$

где $f_i(x) = x^\top A_i x + 2b_i^\top x + c_i$

- Для получения SDP задачи, которая даёт оценку снизу на решение исходной задачи запишем $f_i(x)$ в таком виде

$$f_i(x) = \begin{bmatrix} x \\ 1 \end{bmatrix}^\top \begin{bmatrix} A_i & b_i \\ b_i^\top & c_i \end{bmatrix} \begin{bmatrix} x \\ 1 \end{bmatrix} \le 0$$

- Теперь проанализируем решение следующей задачи

$$\begin{aligned} &\max t \\ \text{s.t. } &\tau \ge 0 \\ &\begin{bmatrix} A_0 & b_0 \\ b_0^\top & c_0 - t \end{bmatrix} + \tau \begin{bmatrix} A_1 & b_1 \\ b_1^\top & c_1 \end{bmatrix} \succeq 0 \end{aligned}$$

- Пусть x допустимая точка в исходной задаче, а t, τ_1 допустимые точки для введённой задаче, тогда т.к. $f_1(x) \le 0, \ \tau \ge 0$:

$$0 \le [\, x \quad 1 \,] \left(\begin{bmatrix} A_0 & b_0 \\ b_0 & c_0 - t \end{bmatrix} + \tau \begin{bmatrix} A_1 & b_1 \\ b_1^\top & c_1 \end{bmatrix} \right) \begin{bmatrix} x \\ 1 \end{bmatrix} = f_0(x) - t + \tau f_1(x) \le f_0(x) - t$$

Отсюда $f_0(x) \ge t$, т.е. $\max t = \min f_0(x)$, задачи эквивалентны.
Теперь осталось понять, как она запишется в терминах `cvxpy`:

In [22]:

```
t = cvx.Variable()
tau = cvx.Variable()

f_i = [np.block([[E_i["A"], E_i["b"]],
                 [E_i["b"].T, E_i["c"]]])
      for E_i in [E0, E1]]

E_44 = np.zeros((4, 4))
E_44[-1, -1] = 1

sdp_constr = [tau >= 0,
              (-t * E_44 + cvxopt.matrix(f_i[0]))
              + tau * cvxopt.matrix(f_i[1]) >> 0]
sdp_obj = t
sdp_problem = cvx.Problem(cvx.Maximize(sdp_obj), sdp_constr)
```

In [23]:
```
sdp_problem.solve(verbose=True)
```

```
----------------------------------------------------------------------------
	SCS v2.0.2 - Splitting Conic Solver
	(c) Brendan O'Donoghue, Stanford University, 2012-2017
----------------------------------------------------------------------------
Lin-sys: sparse-indirect, nnz in A = 12, CG tol ~ 1/iter^(2.00)
eps = 1.00e-05, alpha = 1.50, max_iters = 5000, normalize = 1, scale = 1.00
acceleration_lookback = 20, rho_x = 1.00e-03
Variables n = 2, constraints m = 11
Cones:	linear vars: 1
	sd vars: 10, sd blks: 1
Setup time: 2.04e-03s
----------------------------------------------------------------------------
 Iter | pri res | dua res | rel gap | pri obj | dua obj | kap/tau | time (s)
----------------------------------------------------------------------------
     0| 4.39e+00  7.89e+01  4.75e-01  2.90e+00  9.06e+00  0.00e+00  1.93e-02
    80| 6.20e-07  2.94e-06  2.02e-06 -1.32e-01 -1.32e-01  2.07e-15  2.12e-02
----------------------------------------------------------------------------
Status: Solved
Timing: Solve time: 2.12e-02s
	Lin-sys: avg # CG iterations: 1.47, avg solve time: 2.38e-04s
	Cones: avg projection time: 6.52e-06s
	Acceleration: avg step time: 1.32e-06s
----------------------------------------------------------------------------
Error metrics:
dist(s, K) = 0.0000e+00, dist(y, K*) = 2.0961e-09, s'y/|s||y| = 9.1846e-11
primal res: |Ax + s - b|_2 / (1 + |b|_2) = 6.1966e-07
dual res:   |A'y + c|_2 / (1 + |c|_2) = 2.9419e-06
rel gap:    |c'x + b'y| / (1 + |c'x| + |b'y|) = 2.0180e-06
----------------------------------------------------------------------------
c'x = -0.1316, -b'y = -0.1316
============================================================================
```

Out[23]: 0.1315786595859687

Ура! Ответы совпали, эллипсы не пересекаются.

Задача 5

Пусть известны координаты N точек $x_1, \dots, x_N$ на плоскости. Поставьте задачу полуопределённого программирования для поиска точки на плоскости, максимальное евклидово расстояние от которой до точек $x_1, \dots, x_N$ было бы минимально. Найдите такую точку, если точки x_i – это города Афины, Москва, Берлин, Копенгаген, Прага и Варшава, а Европа расположена на плоскости.

Решение: Задач оптимизации запишется так:

$$\min_{x\in\mathbb{R}^2}\max_{i=1...N}\|x-x_i\|_2^2$$

Квадраты можно навешивать, поскольку это ни на что не влияет. Такая задача допускает эквивалентную переформулировку:

$$\begin{aligned}&\min_{x,a} a\\ \text{s.t. } &\|x-x_i\|_2^2\le a^2,\ i=1\dots N\\ &a\ge 0\end{aligned}$$

В самом деле: если максимум $\le a$, то остальные и подавно, потому минимальное a и даст ответ.
А это уже практически SDP-задача, нужно только немного ограничения привести в матричный вид.
Причём заметим, что это, по сути, задача об окружности минимального радиуса, содержащей все данные точки.
Точно такую же задачу, только в общем случае — для эллипсоидов в n-мерном пространстве — мы разбирали на семинаре.
Вспоминаем, что сфера это тоже эллипс, сводим задачу к предыдущей. Цитирую семинар:

Эллипс можно задать аффинным преобразованием:

$$\{x\mid \|x\|_2^2\le 1\}\to\{u\mid \|u\|_2^2\le 1,\ u=Ax+b\}$$

Тогда площадь увеличивается в $\det(A^{-1})$ раз.

- Детерминант не является выпуклой/вогнутой функцией
- $\log\det(A^{-1})=-\log\det(A)$ - выпуклая функция при $A\in\mathbb{S}^n_{++}$

В данном случае эллипс имеет вид

$$\frac{(x^{(1)}-x_*^{(1)})^2}{a^2}+\frac{(x^{(2)}-x_*^{(2)})^2}{a^2}\le 1$$

где x_* — оптимальное положение центра, a — радиус шара — неизвестные параметры. Отсюда

$$A=\operatorname{diag}(a^{-1}),\ b=-x_*/a,\ \ -\log\det A=-\log a^{-2}=2\log a$$

И если в общем виде задача оптимизации в SDP-записи имеет такой вид:

$$\begin{aligned} &\min_{A,b} -\log\det(A) \\ \text{s.t. } &\|Ax_i + b\|_2 \le 1 \\ &A \succ 0 \end{aligned}$$

То в этом частном случае она имеет вид:

$$\begin{aligned} &\min_{x,a} a^2 \\ \text{s.t. } &\|x_i - x\|_2^2 \le a^2 \\ &a > 0 \end{aligned}$$

Убрать логарифм можно, т.к. это монотонно возрастающая функция, параметр масштаба из-под нормы выносится.
Значит, естественная формулировка задачи и есть её формулировка в SDP в смысле семинара, чтд.

Ограничение на норму можно переписать в матричном виде. Для этого нужны два утверждения:

1. При условии обратимости подматрицы D определитель блочно-диагональной матрицы записывается в виде:
$$\det\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \det(A - BD^{-1}C)\det(D)$$
Этот факт известен из курса линейной алгебры.
2. Ограничение вида $(Ax+b)^\top(Ax+b) - c^\top x - d \le 0$ допускает матричную запись вида:
$$\begin{bmatrix} I & Ax+b \\ (Ax+b)^\top & c^\top x + d \end{bmatrix} \succeq 0$$
Эквивалентность доказывается с помощью критерия Сильвестра: все миноры, кроме последнего, > 0 по очевидным причинам, а последний имеет нужный вид по формуле выше.

В данном частном случае:

$$\begin{aligned} A &= \operatorname{diag}(a^{-1}) \\ b &= -x \;/\; a \\ c &= 0 \\ d &= 1 \end{aligned}$$

$$
\begin{aligned}
&\min_{x,a} a \\
&\text{s.t. } \begin{bmatrix} I_2 & a^{-1}(x_i - x) \\ a^{-1}(x_i - x)^\top & 1 \end{bmatrix} \succeq 0 \\
&a > 0
\end{aligned}
$$

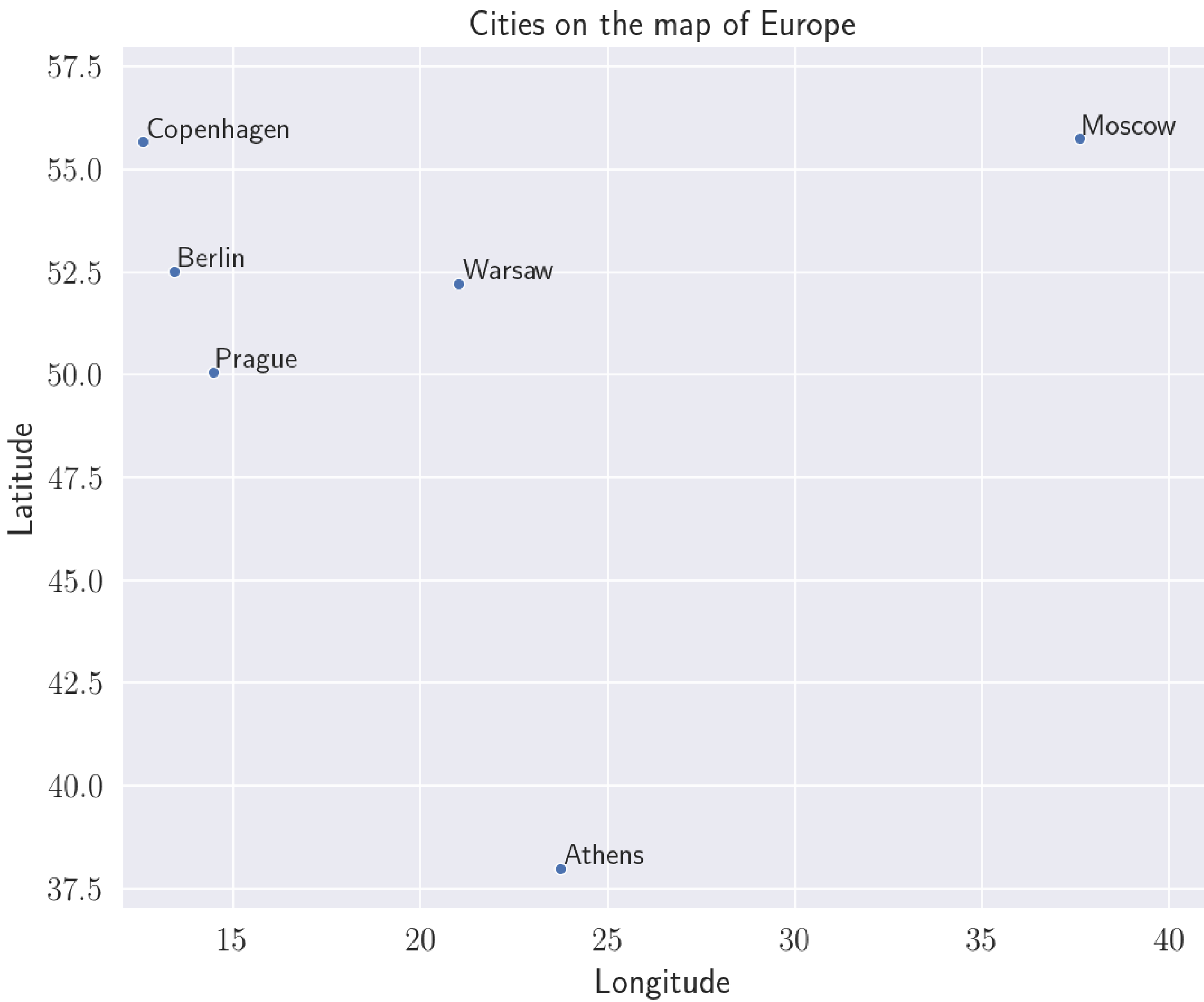
Итого получаем N матричных ограничений. Их можно склеить в одну большую блочно-диагональную матрицу. Всё по той же формуле для определителя блочной матрицы можно показать по индукции, что положительная определённость матрицы из **таких** (их структура существенна!) блоков эквивалентна положительной определённости каждого из блоков. Если так сделать, то получится честная SDP-запись, но **зачем**.

In [24]:
```
labels = ["Athens", "Moscow", "Berlin", "Copenhagen", "Prague", "Warsaw"]
X = np.matrix(
    [[37.9838, 23.7275], # Афины
     [55.7558, 37.6173], # Москва
     [52.5200, 13.4050], # Берлин
     [55.6761, 12.5683], # Копенгаген
     [50.0755, 14.4378], # Прага
     [52.2297, 21.0122]] # Варшава
)
N = X.shape[0]
```

Научимся строить эти точки на графике: потом это пригодится.

In [25]:
```
def draw_cities_on_the_map(points, labels, special_points=None, sp_labels=None):
    plt.figure(figsize=(10,8))
    plt.title("Cities on the map of Europe")
    sns.set(font_scale=1.5)
    ax = sns.scatterplot(x=np.ravel(points[:, 1]), y=np.ravel(points[:, 0]))
    for i in range(points.shape[0]):
        point = np.ravel(points[i, :])
        plt.text(point[1] + .1, point[0] + .1, labels[i], fontsize=15)
    if special_points is not None:
        for i in range(len(special_points)):
            sp = special_points[i]
            sns.scatterplot([sp[1]], [sp[0]], color="red")
            plt.text(sp[1] + .1, sp[0] + .1, sp_labels[i], fontsize=15)
            lines = [[(sp[1], sp[0]),
                      (points[i, 1], points[i, 0])]
                     for i in range(points.shape[0])]
            lc = mc.LineCollection(lines,
                                   colors="black",
                                   linewidths=0.5,
                                   linestyles='--')
            ax.add_collection(lc)
    plt.xlim((12, 41))
    plt.ylim((37, 58));
    plt.xlabel("Longitude");
    plt.ylabel("Latitude");
```

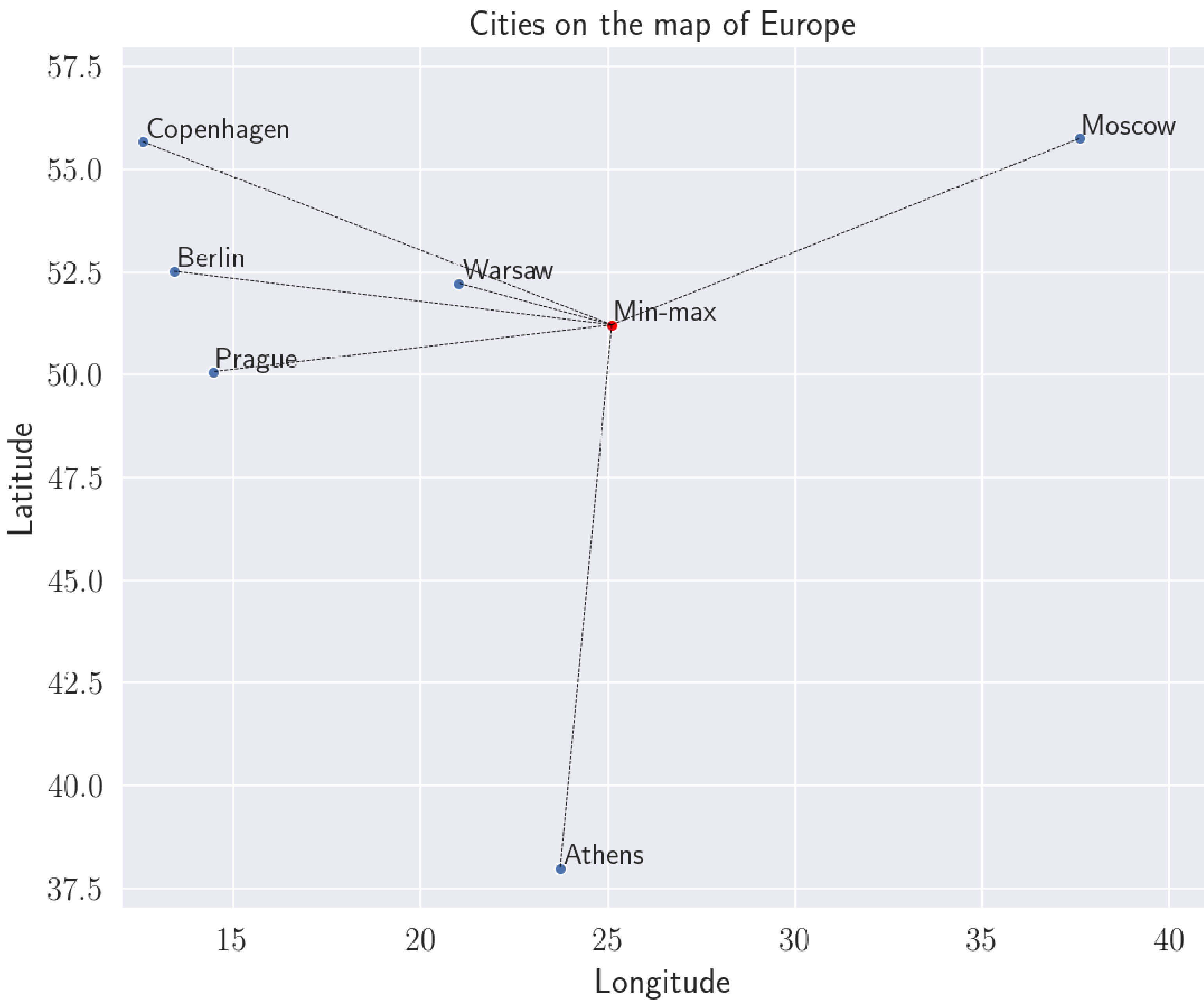
In [26]:
```
draw_cities_on_the_map(X, labels)
```


Cities on the map of Europe
Copenhagen
Moscow
Berlin
Warsaw
Prague
Athens
57.5
55.0
52.5
50.0
47.5
45.0
42.5
40.0
37.5
Latitude
15
20
25
30
35
40
Longitude

Опять-таки, попробуем вначале решить задачу по-простому: просто всунуть задачу в солвер в том виде, в каком она есть, без сведения к SDP:

In [27]:
```
x = cvx.Variable(2)
a = cvx.Variable()

naive_obj = a
naive_constr = [a >= 0] + [cvx.norm(x - np.ravel(X[i, :]))**2 <= a for i in range(N)]
naive_prob = cvx.Problem(cvx.Minimize(naive_obj), naive_constr)

naive_prob.solve(verbose=True)
print(x.value)
print(naive_prob.value)

draw_cities_on_the_map(X, labels, [x.value], ["Min-max"])
assert np.isclose(naive_prob.value, np.max([np.linalg.norm(X[i, :] - x.value)**2 for i in range(N)]))
```

```
ECOS 2.0.4 - (C) embotech GmbH, Zurich Switzerland, 2012-15. Web: www.embotech.com/ECOS

It     pcost       dcost      gap   pres   dres    k/t    mu     step   sigma     IR    |   BT
 0  +4.007e-17  -1.934e+00  +2e+02  3e-01  1e+00  1e+00  8e+00    ---    ---    1  1  - |  -  -
 1  +5.011e+01  +5.778e+01  +4e+01  4e-01  2e+00  1e+01  2e+00  0.9533  2e-01   1  1  1 |  0  0
 2  +4.854e+01  +6.114e+01  +3e+01  3e-01  9e-01  1e+01  1e+00  0.5922  4e-01   1  1  1 |  0  0
 3  +7.795e+01  +8.193e+01  +5e+00  5e-02  2e-01  4e+00  2e-01  0.8721  6e-02   1  1  1 |  0  0
 4  +9.506e+01  +1.005e+02  +3e+00  5e-02  1e-01  6e+00  2e-01  0.5969  4e-01   2  1  1 |  0  0
 5  +1.270e+02  +1.301e+02  +8e-01  2e-02  4e-02  3e+00  4e-02  0.8165  9e-02   2  1  1 |  0  0
 6  +1.334e+02  +1.371e+02  +6e-01  2e-02  4e-02  4e+00  3e-02  0.4130  5e-01   2  1  2 |  0  0
 7  +1.565e+02  +1.579e+02  +1e-01  5e-03  1e-02  1e+00  8e-03  0.8549  1e-01   2  1  1 |  0  0
 8  +1.616e+02  +1.630e+02  +1e-01  5e-03  8e-03  1e+00  5e-03  0.4845  4e-01   3  2  2 |  0  0
 9  +1.739e+02  +1.741e+02  +1e-02  8e-04  1e-03  3e-01  8e-04  0.8914  4e-02   2  2  2 |  0  0
10  +1.761e+02  +1.762e+02  +5e-03  2e-04  4e-04  9e-02  2e-04  0.8305  2e-01   2  1  1 |  0  0
11  +1.770e+02  +1.770e+02  +4e-04  2e-05  3e-05  7e-03  2e-05  0.9239  8e-04   1  1  1 |  0  0
12  +1.771e+02  +1.771e+02  +6e-05  3e-06  4e-06  1e-03  3e-06  0.9439  1e-01   2  1  1 |  0  0
13  +1.771e+02  +1.771e+02  +7e-06  3e-07  5e-07  1e-04  3e-07  0.8877  3e-03   2  1  1 |  0  0
14  +1.771e+02  +1.771e+02  +1e-06  7e-08  1e-07  3e-05  7e-08  0.9223  1e-01   2  1  1 |  0  0
15  +1.771e+02  +1.771e+02  +2e-07  8e-09  1e-08  3e-06  8e-09  0.8829  3e-03   3  1  1 |  0  0
16  +1.771e+02  +1.771e+02  +3e-08  2e-09  2e-09  6e-07  1e-09  0.9274  1e-01   2  1  1 |  0  0

OPTIMAL (within feastol=2.2e-09, reltol=1.5e-10, abstol=2.7e-08).
Runtime: 0.000767 seconds.

[51.21938397 25.10710708]
177.08399378168693
```



Cities on the map of Europe
Copenhagen
Moscow
Berlin
Warsaw
Min-max
Prague
Athens
57.5
55.0
52.5
50.0
47.5
45.0
42.5
40.0
37.5
Latitude
15
20
25
30
35
40
Longitude

Построим интереса ради точку, которая минимизирует сумму расстояний — т.н. геометрическую медиану.

Задача именная — "о геометрической медиане набора точек".

$$\min_{x \in \mathbb{R}^2} \sum_{i=1}^{N} \|x - x_i\|_2$$

Она допускает такую переформулировку:

$$\begin{aligned} &\min_{x,a} \sum_{i=1}^{N} a^{(i)} \\ \text{s.t. } &\|x - x_i\|_2 \le a^{(i)},\ i = 1 \dots N \\ &a \succeq 0 \end{aligned}$$

Это уже хуже, чем было, потому что в терминах SDP она формулируется сложно (нужно поступать как в задаче 4, а тут N вспомогательных переменных). Здесь абсолютно аналогичное сведение к блочно-диагональной матрице, только теперь a_i могут быть различными.

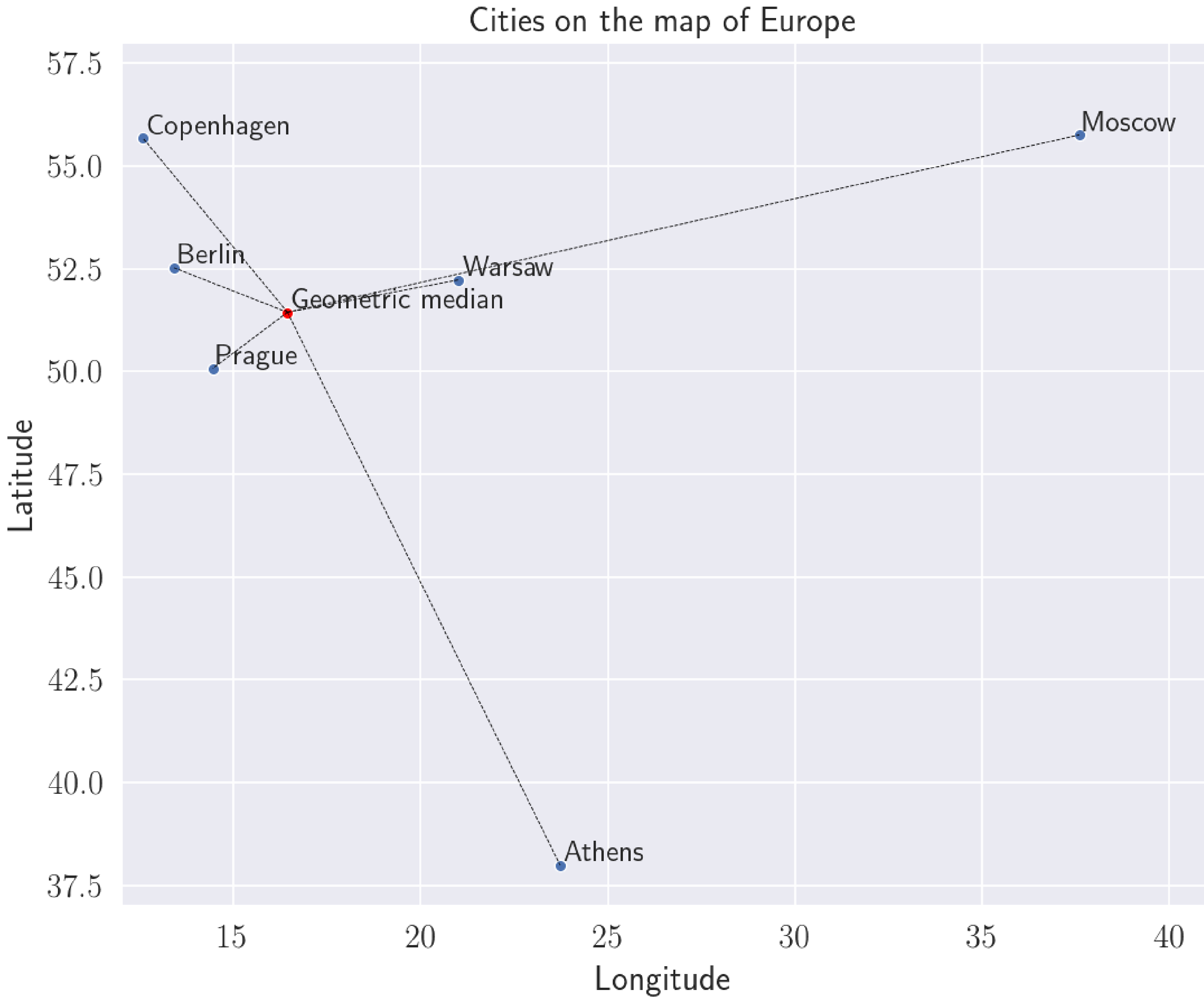
In [28]:
```
x = cvx.Variable(2)
a = cvx.Variable(N)

naive_obj = cvx.sum(a)
naive_constr = [a >= 0] + [cvx.norm(x - np.ravel(X[i, :])) <= a[i] for i in range(N)]
naive_prob = cvx.Problem(cvx.Minimize(naive_obj), naive_constr)
```

In [29]:
```
naive_prob.solve(verbose=True)
print(x.value)
print(naive_prob.value)
```

```
ECOS 2.0.4 - (C) embotech GmbH, Zurich Switzerland, 2012-15. Web: www.embotech.com/ECOS

It     pcost       dcost      gap   pres   dres    k/t    mu     step   sigma     IR    |   BT
 0  +0.000e+00  -0.000e+00  +6e+01  3e-01  1e-02  1e+00  3e+00    ---    ---    1  1  - |  -  -
 1  +1.057e+01  +1.147e+01  +3e+01  2e-01  8e-03  2e+00  2e+00  0.6416  4e-01   1  1  1 |  0  0
 2  +3.173e+01  +3.267e+01  +1e+01  1e-01  3e-03  1e+00  6e-01  0.7458  7e-02   1  1  1 |  0  0
 3  +4.327e+01  +4.371e+01  +3e+00  4e-02  1e-03  6e-01  2e-01  0.7779  7e-02   1  1  1 |  0  0
 4  +4.855e+01  +4.898e+01  +2e+00  3e-02  7e-04  5e-01  9e-02  0.6563  3e-01   1  1  1 |  0  0
 5  +5.205e+01  +5.212e+01  +2e-01  5e-03  1e-04  8e-02  1e-02  0.8818  2e-02   1  1  1 |  0  0
 6  +5.289e+01  +5.290e+01  +1e-02  2e-04  5e-06  4e-03  6e-04  0.9657  2e-02   1  1  1 |  0  0
 7  +5.294e+01  +5.294e+01  +3e-04  8e-06  2e-07  1e-04  2e-05  0.9681  6e-04   1  1  1 |  0  0
 8  +5.294e+01  +5.294e+01  +2e-05  4e-07  8e-09  7e-06  9e-07  0.9549  3e-03   1  1  1 |  0  0
 9  +5.294e+01  +5.294e+01  +8e-07  2e-08  4e-10  3e-07  5e-08  0.9507  5e-04   2  1  1 |  0  0
10  +5.294e+01  +5.294e+01  +2e-07  4e-09  8e-11  7e-08  9e-09  0.8124  1e-02   1  1  1 |  0  0

OPTIMAL (within feastol=3.5e-09, reltol=3.1e-09, abstol=1.6e-07).
Runtime: 0.000370 seconds.

[51.44066025 16.43477405]
52.9385715412679
```

In [30]:

```
draw_cities_on_the_map(X, labels, [x.value], ["Geometric median"])
print(np.sum([np.linalg.norm(X[i, :] - x.value) for i in range(N)]))
```



Cities on the map of Europe
Copenhagen
Moscow
Berlin
Warsaw
Geometric median
Prague
Athens
57.5
55.0
52.5
50.0
47.5
45.0
42.5
40.0
37.5
Latitude
15
20
25
30
35
40
Longitude

52.93857231740351