lektion14

January 23, 2020

Table of Contents

[1]:
```
from IPython.display import display
import numpy as np
import matplotlib.pyplot as plt
from sympy import *
init_printing()
#%matplotlib qt
%matplotlib inline
%config InlineBackend.figure_format = 'svg'
x = symbols('x:2')
mu = symbols('mu')
```

1 Lektion 14

1.1 Extrema unter Nebenbedingungen

Problemstellung

Wir wollen für hinreichend oft differenzierbare reellwertige Funktionen $f : \mathbb{R}^\kappa \to \mathbb{R}$ und $g_j : \mathbb{R}^\kappa \to \mathbb{R}$, $j = 1, \ldots, m$, das Minimierungsproblem

$$\min_x f(x) \qquad s.d. \quad g_j(x) = 0 \text{ für } j = 1, \ldots, m \qquad (*) \tag{1}$$

lösen.

Definition

x^* ist eine lokale Lösung von $(*)$, falls $g_j(x^*) = 0$ für alle $j = 1, \ldots, m$ und falls es $\epsilon > 0$ gibt, mit

$$f(x^*) \leq f(x) \quad \text{für alle} \quad x \text{ mit } \|x - x^*\| < \epsilon \text{ und } g_j(x) = 0 \quad j = 1, \ldots, m.$$

1.1.1 Notwendige Bedingungen erster Ordnung

Definition (Lagrangefunktion) Für $\mu = (\mu_1, \ldots, \mu_m) \in \mathbb{R}^m$ ist die Lagrangefunktion zu $(*)$:

$$\mathcal{L}(x, \mu) = f(x) - \sum_{j=1}^{m} \mu_j g_j(x)$$

In der Analysis 2 wird folgender Satz bewiesen:

Satz (Notwendige Bedingungen erster Ordnung)

Ist x^* eine lokale Lösung von $(*)$ und ist die Menge $\{\nabla_x g_j(x^*), j = 1, \ldots, m\}$ linear unabhängig, dann gibt es $\mu_j^* \in \mathbb{R}$, $j = 1 \ldots, m$ so, dass

$$\nabla_x \mathcal{L}(x^*, \mu^*) = 0 \tag{2}$$

$$g_j(x^*) = 0 \; \forall \; j = 1, \ldots, m \tag{3}$$

kurz:

$$\nabla \mathcal{L}(x^*, \mu^*) = 0$$

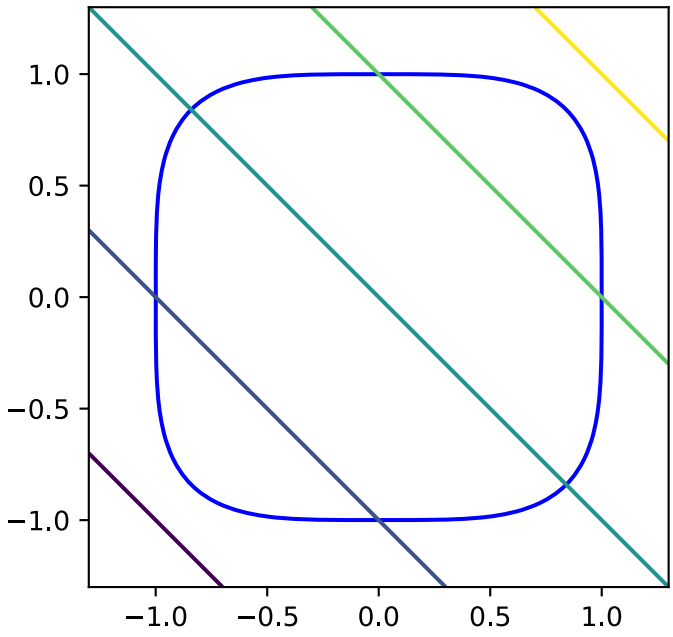
Beispiel

[2]:
```
g = x[0]**4+x[1]**4-1 # Fall m = 1
f = x[0]+x[1]

gn = lambdify(x, g)
fn = lambdify(x, f)
gn(1, 2)
```

[2]:
16

[3]:
```
x0, x1 = np.linspace(-1.3, 1.3, 100), np.linspace(-1.3, 1.3, 100)
X0, X1 = np.meshgrid(x0, x1)
fig = plt.figure()
ax = fig.gca()
pg = ax.contour(X0, X1, gn(X0, X1), 0, colors='blue')
ax.contour(X0, X1, fn(X0, X1), [-2, -1, 0, 1, 2])
ax.set_aspect('equal')
```

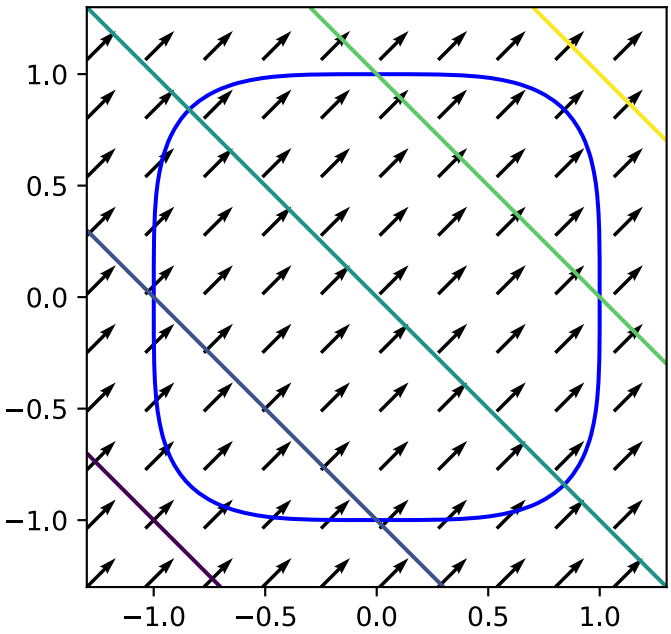


[4]:
```
grad_g = Matrix(2,1,[g.diff(var) for var in x])
grad_f = Matrix(2,1,[f.diff(var) for var in x])
grad_g
```

[4]:
$$\begin{bmatrix} 4x_0^3 \\ 4x_1^3 \end{bmatrix}$$

[5]:
```
grad_gn = lambdify(x,grad_g)
grad_fn = lambdify(x,grad_f)
```

[6]:
```
fig = plt.figure()
ax = fig.gca()
pg = ax.contour(X0, X1, gn(X0, X1), 0, colors='blue')
ax.contour(X0, X1, fn(X0, X1), [-2, -1, 0, 1, 2])
GF = grad_fn(X0[::10, ::10], X1[::10, ::10])
ax.quiver(X0[::10, ::10], X1[::10, ::10], GF[0], GF[1], angles='xy', scale=20)
ax.set_aspect('equal')
```



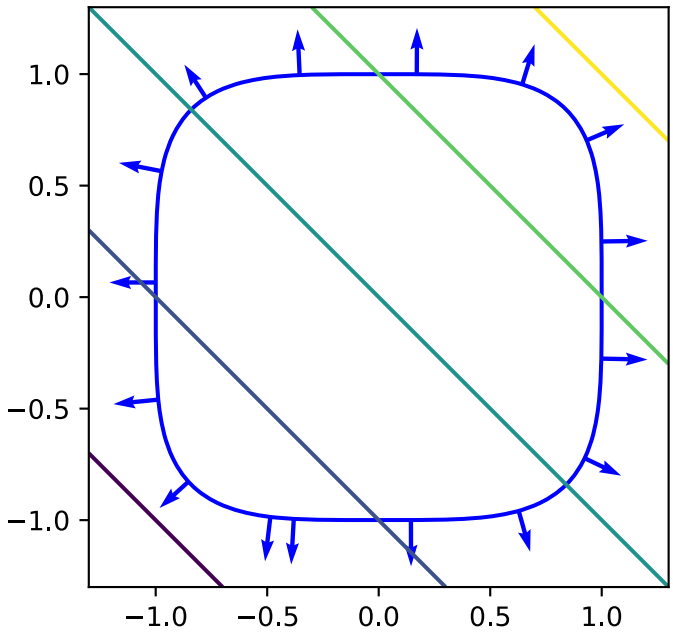
[7]:
```
# 305 Punkte auf der g(x)=0 Linie
(pg.allsegs[1][0]).shape
```

[7]:
(305, 2)

[8]:
```
# Punkte auf der g(x)=0 Linie
SG = np.array((pg.allsegs)[1][0])
X0G, X1G = SG[::20, 0], SG[::20, 1]
X0G**4+X1G**4
```

[8]:
```
array([0.99986689, 0.99995981, 0.99950674, 0.99898692, 0.99903615,
       0.99918032, 0.99900014, 0.99898292, 0.99945686, 0.99959936,
       0.99897277, 0.99896993, 0.99921457, 0.99960604, 0.99898485,
       0.99904591])
```

[9]:
```
fig = plt.figure()
ax = fig.gca()
pg = ax.contour(X0, X1, gn(X0, X1), 0, colors='blue')
ax.contour(X0, X1, fn(X0, X1), [-2, -1, 0, 1, 2])
GG = grad_gn(X0G, X1G)
ax.quiver(X0G, X1G, GG[0], GG[1], angles='xy', scale=50, color='blue')
ax.set_aspect('equal')
```



[10]:
```
# Gradient bzgl. x der Lagrangefunktion
GLF = grad_f-mu*grad_g
```

[11]:
```
# Diese Punkte erfüllen die notwendingen Bedingungen
M = solve([GLF, g])
M
```

[11]:
$$\left[\left\{\mu : -\frac{2^{\frac{3}{4}}}{4},\ x_0 : -\frac{2^{\frac{3}{4}}}{2},\ x_1 : -\frac{2^{\frac{3}{4}}}{2}\right\},\ \left\{\mu : \frac{2^{\frac{3}{4}}}{4},\ x_0 : \frac{2^{\frac{3}{4}}}{2},\ x_1 : \frac{2^{\frac{3}{4}}}{2}\right\},\ \left\{\mu : \frac{\sqrt{2}\,(-1+i)}{8},\ x_0 : -\frac{\sqrt{2}}{4} + \frac{\sqrt{6}}{4} + \frac{\sqrt{2}i}{4}\right.$$

[12]:
```
# Wir suchen uns die reellen Lösungen raus
rsol = []
for sol in M:
    if (im(sol[mu])==0) & (im(sol[x[0]])==0) & (im(sol[x[1]])==0):
        rsol.append(sol)
rsol
```

[12]:
$$\left[\left\{\mu : -\frac{2^{\frac{3}{4}}}{4},\ x_0 : -\frac{2^{\frac{3}{4}}}{2},\ x_1 : -\frac{2^{\frac{3}{4}}}{2}\right\},\ \left\{\mu : \frac{2^{\frac{3}{4}}}{4},\ x_0 : \frac{2^{\frac{3}{4}}}{2},\ x_1 : \frac{2^{\frac{3}{4}}}{2}\right\}\right]$$

[13]:
```
g.subs(rsol[0]) # Probe
```

[13]:
$$0$$

[14]:
```
g.subs(rsol[1]) # Probe
```

[14]:

0

In der Optimierung werden folgende Sätze bewiesen:

1.1.2 Notwendige Bedingungen zweiter Ordnung

Satz (Zweite Ordnung notwendige Bedingungen)

Ist x^* eine lokale Lösung von (*) und ist die Menge $\{\nabla_x g_j(x^*), j = 1, \ldots, m\}$ linear unabhängig, dann gilt

$$s^T \nabla_{xx} \mathcal{L}(x^*, \mu^*) s \geq 0 \quad \forall s \in \{s \in \mathbb{R}^n : s^T \nabla_x g_j(x^*) = 0 \ \forall j = 1, \ldots, m\}.$$

1.1.3 Hinreichende Bedingungen zweiter Ordnung

Satz (Zweite Ordnung hinreichende Bedingungen)

Gibt es für x^* mit $g_j(x^*) =0 $ für $j = 1, \ldots, m$ Lagrangemultiplikatoren $\mu_j^* \in \mathbb{R}$, $j = 1, ;m$ so, dass $\nabla_x \mathcal{L}(x^*, \mu^*) = 0$ und ist

$$s^T \nabla_{xx} \mathcal{L}(x^*, \mu^*) s > 0 \quad \forall s \in \{s : s^T \nabla_x g_j(x^*) = 0 \ \forall j = 1, \ldots, m, s \neq 0\},$$

so ist x^* eine lokale Lösung von $(*)$.

[15]:
```
H = Matrix(2, 2,[ (f-mu*g).diff(x0,x1) for x0 in x for x1 in x])
H
```

[15]:
$$\begin{bmatrix} -12\mu x_0^2 & 0 \\ 0 & -12\mu x_1^2 \end{bmatrix}$$

[16]:
```
H.subs(rsol[0]) # diese Matrix ist positiv definit
```

[16]:
$$\begin{bmatrix} 3\sqrt[4]{2} & 0 \\ 0 & 3\sqrt[4]{2} \end{bmatrix}$$

Also haben wir hier ein Minimum (Achtung: Für das Minimum reicht es, dass nur ein Teil der Hessematrix positiv definit ist, siehe obiger Satz)

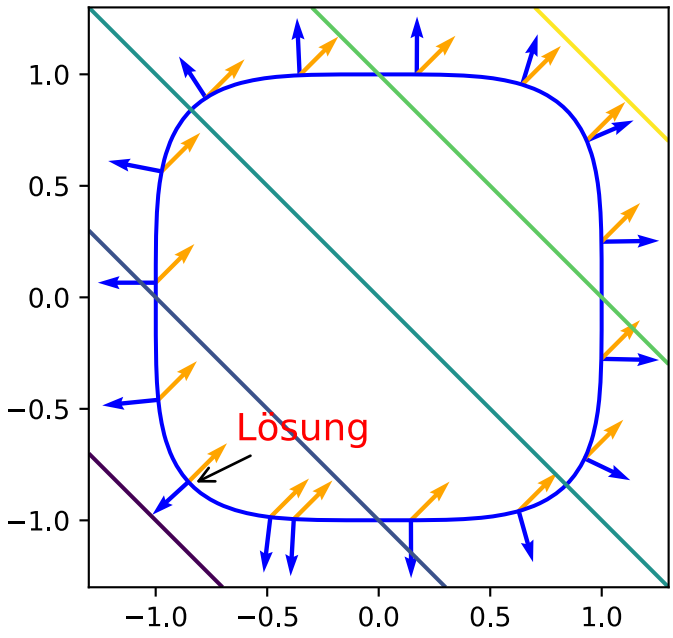
[17]:
```
H.subs(rsol[1])
```

[17]:
$$\begin{bmatrix} -3\sqrt[4]{2} & 0 \\ 0 & -3\sqrt[4]{2} \end{bmatrix}$$

[18]:
```
xx0=rsol[0][x[0]]
xx1=rsol[0][x[1]]
```

[19]:
```
fig = plt.figure()
ax = fig.gca()
pg = ax.contour(X0, X1, gn(X0, X1), 0, colors='blue')
```

```
ax.contour(X0, X1, fn(X0, X1), [-2, -1, 0, 1, 2])
GG = grad_gn(X0G, X1G)
GF = grad_fn(X0G, X1G)
ax.quiver(X0G, X1G, GG[0], GG[1], angles='xy', scale=40, color='blue')
ax.quiver(X0G, X1G, GF[0], GF[1], angles='xy', scale=15, color='orange')
ax.set_aspect('equal')
ax.annotate('Lösung', (xx0, xx1), (xx0+.2, xx1+.2),\
            arrowprops={'arrowstyle':'->'}, color='red', fontsize=14);
```



[ ]: