Department of Mathematics and Computer Science
University of Southern Denmark, Odense

April 5, 2025
Marco Chiarandini

# AI505 – Optimization

## Sheet 06, Spring 2025

---

**Solution:**
Included.
Exercises with the symbol $^+$ are to be done at home before the class. Exercises with the symbol $^*$ will be tackled in class. The remaining exercises are left for self training after the exercise class. Some exercises are from the text book and the number is reported. They have the solution at the end of the book.

### Exercise $1^+$ (10.9)

Solve the constrained optimization problem

$$\begin{aligned} &\underset{x}{\text{minimize}} && \sin\left(\frac{4}{x}\right) \\ &\text{subject to} && x \in [1, 10] \end{aligned}$$

using both the transform $x = t_{a,b}(\hat{x})$ and a sigmoid transform for constraint bounds $x \in [a, b]$:

$$x = s(\hat{x}) = a + \frac{(b-a)}{1 + \exp(-\hat{x})}$$

Why is the $t_{a,b}$ transform better in this case than the sigmoid transform?

**Solution:**
The problem is minimized at $x^* = 1$, which is at the constraint boundary. Solving with the $t$-transform yields the unconstrained objective function:

$$f_t(\hat{x}) = \sin\left(\frac{4}{5.5 + 4.5\frac{2\hat{x}}{1+\hat{x}^2}}\right)$$

which has a single global minimum at $\hat{x} = -1$, correctly corresponding to $x^*$.
The sigmoid transform has an unconstrained objective function:

$$f_s(\hat{x}) = \sin\left(\frac{4}{1 + \frac{9}{1+\exp(-\hat{x})}}\right)$$

Unfortunately, the lower-bound $a$, in this case $x = 1$, is reached only as $\hat{x}$ approaches minus infinity. The unconstrained optimization problem obtained using the sigmoid transform does not have a solution.

### Exercise $2^+$ (10.1)

Solve

$$\begin{aligned} &\underset{x}{\text{minimize}} && x \\ &\text{subject to} && x \geq 0 \end{aligned}$$

using the quadratic penalty method with $\rho > 0$. Solve the problem in closed form.

**Solution:**

First reformulate the problem as $f(x) = x + \rho \max(-x, 0)^2$ for which the derivative is

$$f'(x) = \begin{cases} 1 + 2\rho x & \text{if } x < 0 \\ 1 & \text{otherwise} \end{cases}$$

This unconstrained objective function can be solved by setting $f'(x) = 0$, which yields the solution $x^* = -1/2\rho$. Thus, as $\rho \to \infty$ we have that $x^* \to 0$.

## Exercise 3* (10.11)

Suppose we want to minimize $x_1^3 + x_2^2 + x_3$ subject to the constraint that $x_1 + 2x_2 + 3x_3 = 6$. How might we transform this into an unconstrained problem with the same minimizer?
Solve the transformed problem numerically.

**Solution:**

We can rearrange the constraint in terms of $x_1$:

$$x_1 = 6 - 2x_2 - 3x_3$$

and substitute the relation into the objective:

$$\min_{x_2, x_3} x_2^2 + x_3 - (2x_2 + 3x_3 - 6)^3$$

## Exercise 4* (10.13)

Consider using a penalty method to optimize

$$\begin{aligned} &\underset{x}{\text{minimize}} \ 1 - x^2 \\ &\text{subject to } |x| \leq 2 \end{aligned}$$

Optimization with the penalty method typically involves running several optimizations with increasing penalty weights. Impatient engineers may wish to optimize once using a very large penalty weight. Explain what issues are encountered for both the count penalty method and the quadratic penalty method.
Implement the method and solve the problem numerically.

**Solution:**

The transformed objective function is $f(x) = 1 - x^2 + \rho p(x)$, where $p$ is either a count penalty or a quadratic penalty:

$$p_{count}(x) = (|x| > 2) \quad p_{quadratic}(x) = \max(|x| - 2, 0)^2$$

The count penalty method does not provide any gradient information to the optimization process. An optimization algorithm initialized outside of the feasible set will be drawn away from the feasible region because $1 - x^2$ is minimized by moving infinitely far to the left or right from the origin. The large magnitude of the count penalty is not the primary issue; small penalties can lead to similar problems.
The quadratic penalty method does provide gradient information to the optimization process, guiding searches toward the feasible region. For very large penalties, the quadratic penalty method will produce large gradient values in the infeasible region. In this problem, the partial derivative is:

$$f'(x) = -2x + \rho \begin{cases} 2(x - 2) & \text{if } x > 2 \\ 2(x + 2) & \text{if } x < -2 \\ 0 & \text{otherwise} \end{cases}$$

For very large values of $\rho$, the partial derivative in the infeasible region is also large, which can cause problems for optimization methods. If $\rho$ is not large, then infeasible points may not be sufficiently penalized, resulting in infeasible solutions.

## Exercise 5*

Verify the numerical results of Example 10.3 and 10.4 (pages 172-173) of the text book.

**Solution:**

Classifying problems and deciding which method among those studied in this course can be used to address them is a skill that will be tested at the oral exam. Hence, questions like this are to be expected.

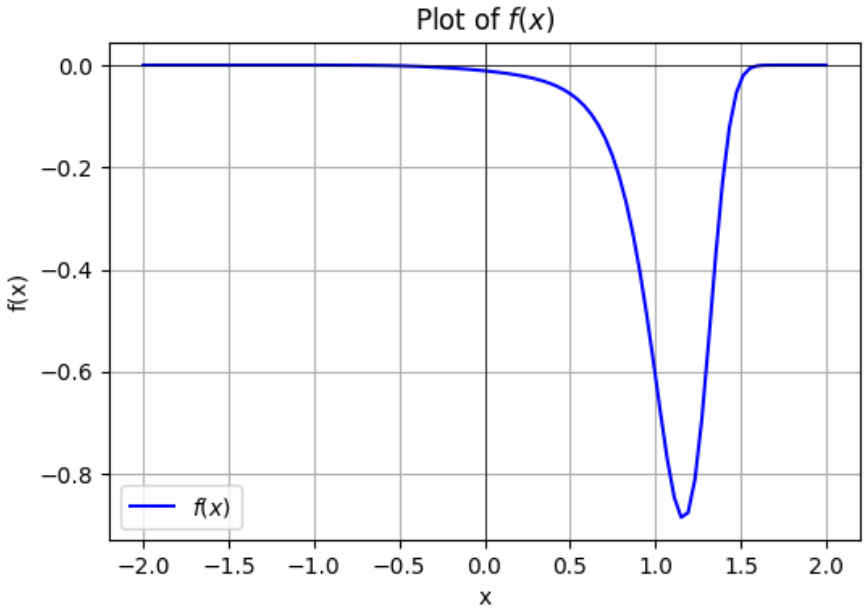
**Example 10.3** can be approached in two ways:

1. as a univariate optimization problem with the given smooth derivative
2. as a root finding problem on the derivative function

In Python:

```
import numpy
import scipy as sp
import matplotlib.pyplot as plt

def func(x: numpy.array):
    return -numpy.exp(-(x[0]*x[1] -3/2)**2-(x[1]-3/2)**2)

def func1(x: float):
    _x=numpy.array([x**2, x])
    return func(_x)

def f1prime(x: float):
    return -6 * func1(x) * (x**5 - 3/2*x**2+1/3*x-1/2)

opt = sp.optimize.minimize_scalar(func1, method='golden')
print(opt)

root = sp.optimize.root_scalar(f1prime, x0=0, method='secant')
print(opt)

# Generate x values
x = numpy.linspace(-2, 2, 100)
y = func1(x)


# Create the plot
plt.figure(figsize=(6, 4))
plt.plot(x, y, label=r"$f(x)$", color="blue")
plt.xlabel("x")
plt.ylabel("f(x)")
plt.title("Plot of $f(x)$")
plt.axhline(0, color='black', linewidth=0.5)
plt.axvline(0, color='black', linewidth=0.5)
plt.legend()
plt.grid(True)

# Show the plot
# plt.show()
plt.savefig("ex10_3.png")
```

```
message:
          Optimization terminated successfully;
          The returned value satisfies the termination criteria
          (using xtol = 1.4901161193847656e-08 )
 success: True
```




```
    fun: -0.8879747422664449
      x: 1.1652486100083423
    nit: 38
   nfev: 43
message:
         Optimization terminated successfully;
         The returned value satisfies the termination criteria
         (using xtol = 1.4901161193847656e-08 )
success: True
    fun: -0.8879747422664449
      x: 1.1652486100083423
    nit: 38
   nfev: 43
```

**Example 10.4** This problem corresponds to solving a system of *nonlinear equations* with the same number of equalities conditions as number of variables (if more variables than equations, we need to fix some variables). There is no need to optimize. In general, the system may have no solutions, a unique solution, or many solutions.

**Netwon's method** A Method to solve this problem is by reducing it to the solution of a system of linear equations.
Let $r_1(\boldsymbol{x}) = 0, \ldots, r_m(\boldsymbol{x}) = 0$ be the non linear equations and $\boldsymbol{r}(\boldsymbol{x}) = \{r_1(\boldsymbol{x}), \ldots, r_m(\boldsymbol{x})\}$ be the set of left hand sides of these equations. We aim at finding a direction $\boldsymbol{p}$ to move $\boldsymbol{x}_k$ towards
Starting from the Taylor expansion:

$$\boldsymbol{r}(\boldsymbol{x} + \boldsymbol{p}) = \boldsymbol{r}(\boldsymbol{x}) + \int_0^1 J(\boldsymbol{x} + t\boldsymbol{p})\boldsymbol{p}dt$$

We can define a linear model $M_k(p)$ of $\boldsymbol{r}(\boldsymbol{x}_k + \boldsymbol{p})$ by approximating the right hand side:

$$M_k(\boldsymbol{p}) \stackrel{def}{=} r(\boldsymbol{x}_k) + J(\boldsymbol{x}_k)p$$

```
# Newton's Method for Nonlinear Equations:
Choose x_0;
for k =0,1,2,... do
    Calculate a solution p_k to the Newton equations J(x_k)p_k = −r(x_k);
    x_{k+1} ← x_k + p_k ;
end
```

When the iterate $x_k$ is close to a nondegenerate root $x^*$, Newton's method converges superlinearly. Potential shortcomings of the method include the following:

- When the starting point is remote from a solution, Algorithm can behave erratically. When $J(x_k)$ is singular, the Newton step may not even be defined.
- First-derivative information (the Jacobian matrix $J$) may be difficult to obtain.
- It may be too expensive to find and calculate the Newton step pk exactly when n is large.
- The root $x^*$ in question may be degenerate, that is, $J(x^*)$ may be singular.

**Broyden's method** Secant methods, also known as quasi-Newton methods, do not require calculation of the Jacobian $J(x)$. Instead, they construct their own approximation to this matrix, updating it at each iteration so that it mimics the behavior of the true Jacobian $J$ over the step just taken. The approximate Jacobian, which we denote at iteration $k$ by $B_k$, is then used to construct a linear model analogous to the one above, namely

$$M_k(\boldsymbol{p}) \stackrel{def}{=} \boldsymbol{r}(\boldsymbol{x}_k) + B_k \boldsymbol{p}$$

The most successful practical algorithm is Broyden's method, for which the update formula is

$$\boldsymbol{s}_k = \boldsymbol{x}_{k+1} - \boldsymbol{x}_k, \qquad \boldsymbol{y}_k = \boldsymbol{r}(\boldsymbol{x}_{k+1}) - \boldsymbol{r}(\boldsymbol{x}_k)$$

$$B_{k+1} = B_k + \frac{(\boldsymbol{y}_k - B_k \boldsymbol{s}_k)\boldsymbol{s}_k^T}{\boldsymbol{s}_k^T \boldsymbol{s}_k}$$

```
# Broyden:
Choose x_0 and a nonsingular initial Jacobian approximation B_0;
for k = 0, 1, 2, ... do
    Calculate a solution p_k to the linear equations;
    B_k p_k − r(x_k);
    Choose α_k by performing a line search along p_k;
    x_{k+1} ← x_k + α_k p_k;
    s_k ← x_{k+1} − x_k;
    y_k ← r(x_{k+1}) − r(x_k);
    Obtain B_{k+1} from the formula above;
end
```

Under certain assumptions, Broyden's method converges superlinearly, that is,

$$\left\|\boldsymbol{x}_{k+1} - x^*\right\| = o(\left\|\boldsymbol{x}_k - \boldsymbol{x}^*\right\|)$$

An initial guess for the Jacobian is $(-1/\alpha)$ or the computed Jacobian.
Implementations in scipy: `root(method='broyden1')`:

```
def Lagrangian(var: numpy.array):
    lmb = var[2]
    x1 = var[0]
    x2 = var[1]
    return func(numpy.array([x1,x2])) + lmb * (x1-x2**2)

def Lagrangian_nabla(var: numpy.array):
    lmb = var[2]
    x1 = var[0]
    x2 = var[1]
    dx1 = 2*x2 * func(numpy.array([x1,x2])) * (3/2-x1*x2)-lmb
```

```
    dx2 = 2*lmb * x2 + func(numpy.array([x1,x2])) * (-2*x1*(x1*x2-3/2)-2*(x2-3/2))
    dlmb = x2**2-x1
    return numpy.array([dx1,dx2,dlmb])

numpy.set_printoptions(suppress=True)
res = sp.optimize.root(Lagrangian_nabla, x0=[0,0,0], method='broyden1', tol=1e-14)
print(res)
print(res.x)
print(Lagrangian_nabla(res.x))

res = sp.optimize.root(Lagrangian_nabla, x0=[0,0,0], method='lm', tol=1e-14)
print(res)
print(res.x)
print(Lagrangian_nabla(res.x))
```

```
message: The maximum number of iterations allowed has been reached.
 success: False
  status: 2
     fun: [-1.956e-03 -3.863e-03  3.962e-03]
       x: [ 8.307e-01 -9.136e-01  2.030e-03]
     nit: 8000
  method: broyden1
    nfev: 16010
[ 0.83067483 -0.91358452  0.00202981]
[-0.00195573 -0.0038628   0.00396184]
 message: xtol=0.000000 is too small, no further improvement in the approximate
            solution is possible.
 success: False
  status: 3
     fun: [-3.053e-16  5.551e-17  0.000e+00]
       x: [ 1.358e+00  1.165e+00  1.701e-01]
  method: hybr
    nfev: 43
    fjac: [[-6.240e-01 -6.789e-01  3.870e-01]
           [-7.383e-02 -4.419e-01 -8.940e-01]
           [-7.779e-01  5.864e-01 -2.256e-01]]
       r: [-2.584e+00 -3.006e+00 -2.467e+00 -3.908e+00 -1.652e+00
            2.316e+00]
     qtf: [ 1.528e-16 -1.988e-18  2.701e-16]
[1.35780432 1.16524861 0.1700643 ]
[-0.  0.  0.]
```

The Broyden method was not successful. The method 'hybr' uses a modification of the Powell hybrid method as implemented in MINPACK and perfromed better. However, results were highly dependent on the initial point.
We should now proceed to test the KKT conditions.

## Exercise 6*

Derive the Karush-Kuhn-Tucker conditions (FONCs) for constrained *maximization* problems.

**Solution:**

In an optimal solution $\boldsymbol{x}^*$, the gradient has to stay in the tangent cone of the active constraints. In the minimization case, imposing $\boldsymbol{\mu} \geq \mathbf{0}$ we derived:

$$\nabla f(\boldsymbol{x}^*) = -\boldsymbol{\mu} \cdot \nabla \boldsymbol{g}(\boldsymbol{x}^*) - \boldsymbol{\lambda} \cdot \nabla \boldsymbol{h}(\boldsymbol{x}^*)$$

In a maximization case, maintaining $\boldsymbol{\mu} \geq \mathbf{0}$ we derive:

$$\nabla f(\boldsymbol{x}^*) = \boldsymbol{\mu} \cdot \nabla \boldsymbol{g}(\boldsymbol{x}^*) - \boldsymbol{\lambda} \cdot \nabla \boldsymbol{h}(\boldsymbol{x}^*)$$

The $\boldsymbol{\lambda}$ multipliers are free ($\boldsymbol{\lambda} \in \mathbb{R}$) hence we do not need to change sign, because they can adapt. Hence, for maximization there is a change of sign only in front of the multipliers $\boldsymbol{\mu}$ in the gradient condition:

$$\begin{cases} \nabla f(\boldsymbol{x}^*) - \boldsymbol{\mu} \cdot \nabla \boldsymbol{g}(\boldsymbol{x}^*) + \boldsymbol{\lambda} \cdot \nabla \boldsymbol{h}(\boldsymbol{x}^*) = 0 \boldsymbol{h}(\boldsymbol{x}^*) = \boldsymbol{0} \\ \boldsymbol{\mu} \cdot \boldsymbol{g}(\boldsymbol{x}^*) = 0 \\ \boldsymbol{g}(\boldsymbol{x}^*) \leq \boldsymbol{0}, \ \boldsymbol{h}(\boldsymbol{x}^*) = \boldsymbol{0} \\ \boldsymbol{\mu} \geq \boldsymbol{0} \end{cases}$$

## Exercise 7*

Consider the following constrained optimization problem:

$$\min \ \left(x_1 - \frac{3}{2}\right)^2 + \left(x_2 - \frac{1}{2}\right)^4 \quad \text{s.t.} \quad \begin{bmatrix} 1 - x_1 - x_2 \\ 1 - x_1 + x_2 \\ 1 + x_1 - x_2 \\ 1 + x_1 + x_2 \end{bmatrix} \geq 0$$

Plot the problem and determine the optimal solution reasoning on the plot. Then, show that in the optimal solution the KKT conditions hold.

**Solution:**

The solution is $\boldsymbol{x}^* = (1, 0)^T$. The first and second constraints are active at this point. Denoting them by $c_1$ and $c_2$ (and the inactive constraints by $c_3$ and $c_4$), we have

$$\nabla f(\boldsymbol{x}^*) = \begin{bmatrix} -1 \\ -\frac{1}{2} \end{bmatrix}, \nabla c_1 = \begin{bmatrix} -1 \\ -1 \end{bmatrix}, \nabla c_2 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}.$$

Therefore, the KKT conditions are satisfied when we set

$$\lambda^* = \left[\tfrac{3}{4}, \tfrac{1}{4}, 0, 0\right]^T.$$

## Exercise 8

Consider the problem

$$\min_{(x1,x2)} \ 0.5(x_1^2 + x_2^2)$$
$$\text{subject to } \ x_1 - 1 \geq 0.$$

Write the Lagrangian function, the dual function and solve the dual problem.

**Solution:**

The *Lagrangian function* is:

$$\mathcal{L}(x_1, x_2, \lambda_1) = 0.5(x_1^2 + x_2^2) - \lambda_1(x_1 - 1).$$

If we hold $\lambda_1$ fixed, this is a convex function of $[x - 1, x_2]$. Therefore, the infimum with respect to $[x_1, x_2]$ is achieved when the partial derivatives with respect to $x_1$ and $x_2$ are zero, that is,

$$x_1 - \lambda_1 = 0, \quad x_2 = 0.$$

By substituting these infimal values into $\mathcal{L}(x_1, x_2, \lambda_1)$ we obtain the *dual objective*:

$$q(\lambda_1) = 0.5(\lambda_1^2 + 0) - \lambda_1(\lambda_1 - 1) = -0.5\lambda_1^2 + \lambda_1.$$

Hence, the *dual problem* is

$$\max_{\lambda_1 \geq 0} \ -0.5\lambda_1^2 + \lambda_1$$

which clearly has the solution $\lambda_1 = 1$.