AI505, Optimization – Exercise Sheet 01

2026-02-17

Exercises with the symbol $^+$ are to be done at home before the exercise 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 in parenthesis. They have the solution at the end of the book.

Exercise 1 $^+$ Python

Show that the function $f(x) = 8x_1 + 12x_2 + x_1^2 - 2x_2^2$ has only one stationary point, and that it is neither a maximum or minimum, but a saddle point (or inflection point). Plot the contour lines of f in Python (see slides 17, 18 of the tutorial material Part 3).

Exercise 2 $^+$

Write the second-order Taylor expansion for the function $\cos(1/x)$ around a nonzero point x, and the third-order Taylor expansion of $\cos(x)$ around any point x. Evaluate the second expansion for the specific case of $x = 1$.

Exercise 3

Suppose that $f(\boldsymbol{x}) = \boldsymbol{x}^T Q \boldsymbol{x}$, where Q is an $n \times n$ *symmetric positive semidefinite* matrix. Show using the definition of convex functions, that $f(\boldsymbol{x})$ is convex on the domain $\mathbb{R}^n$. Hint: It may be convenient to prove the following equivalent inequality: $f(\boldsymbol{y} + \alpha(\boldsymbol{x} - \boldsymbol{y})) - \alpha f(\boldsymbol{x}) - (1-\alpha) f(\boldsymbol{y}) \leq 0$ for all $\alpha \in [0,1]$ and all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$.

Exercise 4

Suppose that f is a convex function. Show that the set of global minimizers of f is a convex set.

Exercise 5 *

Consider the function $f(x_1, x_2) = (x_1 + x_2^2)^2$. At the point $\boldsymbol{x}_0 = [1, 0]$ we consider the search direction $\boldsymbol{p} = [-1, 1]$. Show that $\boldsymbol{p}$ is a descent direction and find all minimizers of the problem $\min_\alpha f(\boldsymbol{x}_0 + \alpha \boldsymbol{p})$.

Exercise 6 $^+$

Consider the case of a vector-valued function $f : \mathbb{R}^n \to \mathbb{R}^m$. The matrix $J(x)$ of dimension $n \times m$ of first derivatives for this function is defined as follows:

$$J(\boldsymbol{x}) = \left[\frac{\partial}{\partial x_i} f_j \right]_{\substack{i=1..n \\ j=1..m}} .$$

Write the forward-difference calculations needed to compute $J(\boldsymbol{x})$ at a given point $\boldsymbol{x}$.

Exercise 7 $^+$ (2.1)

Adopt the forward difference method to approximate the Hessian of $f(x)$ using its gradient, $\nabla f(x)$.

Exercise 8 (2.6)

Combine the forward and backward difference methods to obtain a difference method for estimating the second-order derivative of a function f at x using three function evaluations.

Exercise 9 Python (2.3)

Implement in Python a finite difference method and the complex step method and compute the gradient of $f(x) = \ln x + e^x + 1/x$ for a point x close to zero. What term dominates in the expression?

Exercise 10 * (2.5)

Draw the computational graph for $f(x, y) = \sin(x+y^2)$. Use the computational graph with forward accumulation to compute $\frac{\partial f}{\partial y}$ at $(x, y) = (1, 1)$. Label the intermediate values and partial derivatives as they are propagated through the graph.

Exercise 11 * Python

Implement dual numbers in Python overriding the operators `+`,`-`,`*`,`/`. Test the implementation on the following operations:

- ϵ `*` ϵ
- $1/(1 + \epsilon)$
- $(1 + 2\epsilon)$`*`$(3 - 4\epsilon)$

Calculate the forward accumulation of the dual numbers $a = 3 + 1\epsilon$ and $b = 2$ on the computational graph of $\log(a * b + \max(a, 2))$.

Exercise 12 * Python

Read about nanograd and use it to compute by reverse accumulation the gradient of

$$f(x_1, x_2, x_3) = \max\left\{0, \frac{x_1 + (-x_2 x_3)^2}{x_2 x_3}\right\}.$$

```python
import sys
sys.path.append('sheet01/code') # Ensure Python can find the folder
from nanograd import Var

# Point where you want the gradient
x1_0, x2_0, x3_0 = 3.0, 5.0, 9.0

# Create Variables
x1 = Var(x1_0)
x2 = Var(x2_0)
x3 = Var(x3_0)

# Define f(x1,x2,x3)
d = x2 * x3
e = x1 + (-d) ** 2
f = (e / d).relu()     # scalar output

# Reverse accumulation
f.backward()

print("f(x1,x2,x3) =", f.v)
print("df/dx1 =", x1.grad)
print("df/dx2 =", x2.grad)
print("df/dx3 =", x3.grad)
```

```
f(x1,x2,x3) = 45.06666666666667
df/dx1 = 0.022222222222222223
df/dx2 = 8.986666666666668
df/dx3 = 4.992592592592593
```