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

February 12, 2025
Marco Chiarandini

AI505 – Optimization

Sheet 01, 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.

Exercises on

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).

Solution:

```python
import numpy as np
import matplotlib.pyplot as plt


# Define the function
def f(x1, x2):
    return 8 * x1 + 12 * x2 + x1**2 - 2 * x2**2


# Define the grid for plotting
x1 = np.linspace(-10, 10, 400)
x2 = np.linspace(-10, 10, 400)
X1, X2 = np.meshgrid(x1, x2)
Z = f(X1, X2)

# Create the contour plot
plt.figure(figsize=(8, 6))
contour = plt.contour(X1, X2, Z, levels=20, cmap="viridis")
plt.colorbar(contour)
plt.xlabel("x1")
plt.ylabel("x2")
plt.title("Contour plot of f(x1, x2) = 8*x1 + 12*x2 + x1^2 - 2*x2^2")
plt.grid(True)
plt.savefig("contour.png")
```

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

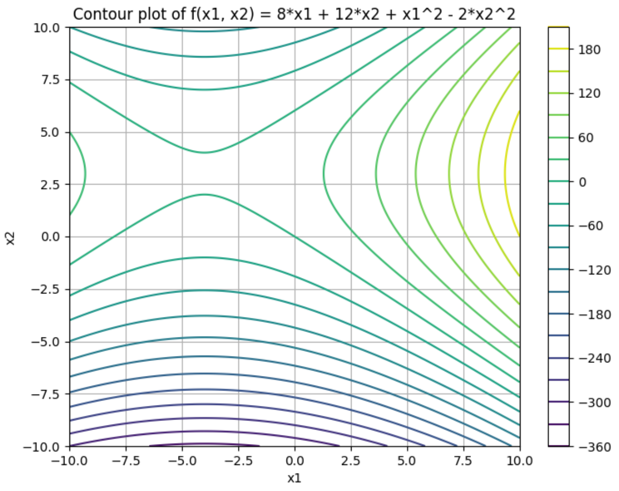

Figure 1: The contour plot of the figure in Exercise 1.

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

Solution:

Since Q is positive semidefinite, the function $f(x)$ is a quadratic form (see the slides $\mathbf{x}A\mathbf{x}$ to confirm this) which is convex. However, we are asked to prove it using the definition, not just citing properties of quadratic forms.

Exercise 4

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

Solution:

We assume to have optimal points x_i from the set of global optimizers $i = 1..n$. Let $x^* = \sum_i \lambda_i x_i$, λ in $[0, 1]$, $\sum_i \lambda = 1$ be any convex combination of them. Because f is convex we have that:

$$f\left(\sum_i \lambda_i x_i\right) \leq \sum_i \lambda_i f(x_i)$$

Since x_i are optimal solutions, $f(x_i) = z$ and there does not exist any other point x' with $f(x') < z$. Using the fact that $\sum_i \lambda = 1$ then:

$$f\left(\sum_i \lambda_i x_i\right) \leq \sum_i \lambda_i f(x_i) = z$$

and since no other point can be better we necessarily have that $f(\sum_i \lambda_i x_i) = z$ and, hence, that any convex combination like x^* is optimal itself. The set of optimal solutions is therefore convex.

Exercise 5*

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

Solution:

Substituting $\mathbf{x}_0$ and $\mathbf{p}$ we want to solve:

$$\min_\alpha f\left(\begin{bmatrix}1\\0\end{bmatrix} + \alpha\begin{bmatrix}-1\\1\end{bmatrix}\right) = \min_\alpha f\left(\begin{bmatrix}1-\alpha\\\alpha\end{bmatrix}\right) = \min_\alpha (1 - \alpha + \alpha^2)^2$$

We look for the points where the necessary conditions of local optimality are satisfied:

$$f'(\mathbf{x}_0, \mathbf{p}, \alpha) = 2(1-\alpha+\alpha^2)(-1+2\alpha) = (4\alpha-2)(\alpha^2-\alpha+1)$$

$$f''(\mathbf{x}_0, \mathbf{p}, \alpha) = 4\alpha^2 - 4\alpha + (2\alpha-1)(4\alpha-2) + 4$$

The first derivative is zero at the only real number: $\alpha_1 = 1/2$. The second term gives a complex number:

$$\alpha_{2,3} = \frac{1 \pm \sqrt{1-4}}{2}$$

In $\alpha_1 = 1/2$, the second derivative is 3 hence positive. The point $\alpha$ is therefore a local minimum. If the function is convex the point is also a global minimum.
The function is convex if its second derivative is always larger or equal to zero, that is:

$$4\alpha^2 - 4\alpha + (2\alpha-1)(4\alpha-2) + 4 \geq 0$$

or equivalently

$$2(6\alpha^2 - 2\alpha + 3) \geq 0$$

The second factor does not have any real root and it is positive in α_1, hence it is always positive.
We could have carried out the analysis in Python as follows:

```python
import numpy as np
import matplotlib.pyplot as plt
import sympy as sp


def f(alpha):
    return (1 - alpha + alpha**2) ** 2


# Define the range of alpha values
alpha_values = np.linspace(-2, 2, 400)
f_values = f(alpha_values)

# Plot the function
plt.plot(
    alpha_values, f_values, label=r"$f(\alpha) = (1 - \alpha + \alpha^2)^2$", color="b"
)
plt.xlabel(r"$\alpha$")
plt.ylabel(r"$f(\alpha)$")
plt.title("Plot of the Function")
plt.legend()
plt.grid()
plt.savefig("function_plot.png")


# Compute the derivatives using sympy
alpha = sp.symbols('alpha')
f_sym = (1 - alpha + alpha**2)**2
deriv_f = sp.diff(f_sym, alpha)
second_deriv_f = sp.diff(deriv_f, alpha)
print("Derivative of f(alpha):", deriv_f)
print("Second derivative of f(alpha):", second_deriv_f)
```

Exercise 6$^+$

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

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

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

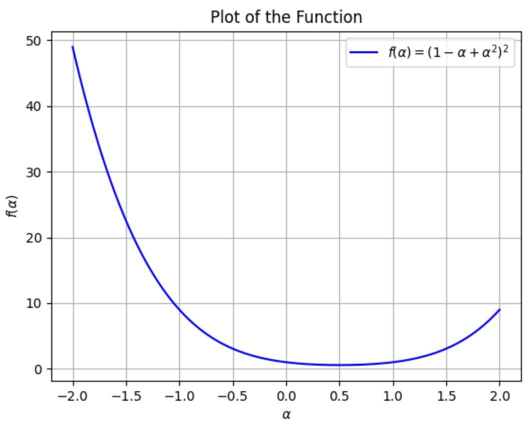

Figure 2: The function $f(\alpha)$ in Exercise 5.

Solution:

For each component j of the vector function $f(\mathbf{x})$ we have a gradient vector in the corresponding column of $J(\mathbf{x})$. Each of the partial derivatives can be approximated by forward difference as follows:

$$\frac{\partial f(\mathbf{x})_j}{\partial x_i} \approx \frac{f(\mathbf{x} + h\mathbf{e}_i)_j - f(\mathbf{x})_j}{h}$$

$$\begin{bmatrix} \frac{f(\mathbf{x}+h\mathbf{e}_1)_1 - f(\mathbf{x})_1}{h} & \frac{f(\mathbf{x}+h\mathbf{e}_1)_2 - f(\mathbf{x})_2}{h} & \cdots & \frac{f(\mathbf{x}+h\mathbf{e}_1)_m - f(\mathbf{x})_m}{h} \\ \frac{f(\mathbf{x}+h\mathbf{e}_2)_1 - f(\mathbf{x})_1}{h} & \frac{f(\mathbf{x}+h\mathbf{e}_2)_2 - f(\mathbf{x})_2}{h} & \cdots & \frac{f(\mathbf{x}+h\mathbf{e}_2)_m - f(\mathbf{x})_m}{h} \\ \vdots & & \ddots & \vdots \\ \frac{f(\mathbf{x}+h\mathbf{e}_n)_1 - f(\mathbf{x})_1}{h} & \frac{f(\mathbf{x}+h\mathbf{e}_n)_2 - f(\mathbf{x})_2}{h} & \cdots & \frac{f(\mathbf{x}+h\mathbf{e}_n)_n - f(\mathbf{x})_n}{h} \end{bmatrix}$$

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:

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

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

Solution:

Dual numbers $D(a, b)$ can be writen as $a + b\epsilon$, where ϵ satisfies $\epsilon^2 = 0$, so we can drop all $O(\epsilon^2)$ terms. The four rules are:

- $(a + b\epsilon) \pm (c + d\epsilon) = (a \pm c) + (b \pm d)\epsilon$
- $(a + b\epsilon) \times (c + d\epsilon) = (ac) + (ad + bc)\epsilon$
- $(a + b\epsilon)/(c + d\epsilon) = (a/c) + (ad - bc)/c^2\epsilon$

As the other rules, the last one is *designed* such that the mulitpliers of ϵ implement the quotient rules of derivatives: Let $h(x) = \dfrac{f(x)}{g(x)}$, where both f and g are differentiable and $g(x) \neq 0$, the quotient rule states that the derivative of $h(x)$ is

$$h'(x) = \frac{f'(x)g(x) - f(x)g'(x)}{(g(x))^2}.$$

```python
import math


class Dual:
    """
    A Dual number that holds two numbers: a value and a gradient.
    """

    def __init__(self, val: float | int, grad: float | int):
        assert type(val) in {float, int}
        assert type(grad) in {float, int}
        self.v = val
        self.g = grad

    def __add__(self: "Dual", other: "Dual") -> "Dual":
        return Dual(self.v + other.v, self.g + other.g)

    def __mul__(self: "Dual", other: "Dual") -> "Dual":
        return Dual(self.v * other.v, self.v * other.g + self.g * other.v)

    def __sub__(self: "Dual", other: "Dual") -> "Dual":
        return Dual(self.v + other.v, self.g + other.g)

    def __truediv__(self: "Dual", other: "Dual") -> "Dual":
        return Dual(
            self.v / other.v, (self.g * other.v - self.v * other.g) / (other.v**2)
        )

    def __repr__(self):
        return "Dual(v=%.4f, g=%.4f)" % (self.v, self.g)
```

We also need to implement how elementary function would treat dual numbers, in particular the derivative that they apply:

$$\frac{\partial \ln(x)}{\partial x} = \frac{x'}{x}$$

$$\frac{\partial \max(x, p)}{\partial x} = \begin{cases} 0 & \text{if } p > x \\ x' & \text{if } p < x \end{cases}$$

```
def log(a: "Dual") -> "Dual":
    return Dual(math.log(a.v), a.g / a.v)


def max(a: "Dual", b: int) -> "Dual":
    return Dual(a.v if a.v > b else b, 0 if b >= a.v else a.g)


a = Dual(3, 1)
b = Dual(2, 0)
print(log(a * b + max(a, 2)))
```

We get the same result as the text book:

```
Dual(v=2.1972, g=0.3333)
```

Exercise 12* Python

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

$$f(x_1, x_2, x_3) = \max\left\{0, \frac{x_1 + (-x_2x_3)^2}{x_2x_3}\right\}.$$

Exercise 13* (3.6)

Suppose we have a unimodal function defined on the interval $[1, 32]$. After three function evaluations of our choice, will we be able to narrow the optimum to an interval of at most length 10? Why or why not? How much more can we reduce with one further evaluation?

Solution:

The best you can do is use Fibonacci Search. With 3 evaluations the uncertainty is shrunk by a factor of $F_{n+1} = F_4 = 3$; that is, to $(32 - 1)/3 = 10 + 1/3 > 10$. With 4 evaluations it is shrunk by a factor of $F_{n+1} = F_5 = 5$; that is to $(32 - 1)/5 = 6.2$. So 4 would be necessary to reduce the interval to at most length 10. The uncertainty shrinks by a factor of $10.3/6.2 = 1.66$ by adding one more evaluation after three or: $\frac{F_{n+1}}{F_n} = \frac{F_5}{F_4} = 1.66$ (quite close to the golden ratio 1.61803 of the Golden ratio search).