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

February 24, 2025
Marco Chiarandini

AI505 – Optimization

Sheet 02, 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$^+$ (3.1)

Give an example of a problem when Fibonacci search can be applied while the bisection method not.

Exercise 2$^+$ (3.4)

Suppose we have $f(x) = x^2/2 - x$. Apply the bisection method to find an interval containing the minimizer of f starting with the interval $[0, 1000]$. Execute three steps of the algorithm (you can do this by hand or in Python).

Exercise 3$^+$ (3.5)

Suppose we have a function $f(x) = (x+2)^2$ on the interval $[0, 1]$. Is 2 a valid Lipschitz constant for f on that interval?

Exercise 4$^+$ (4.2)

Find examples where each of the four termination conditions would not work individually, showing the importance of having more than one.

Exercise 5$^+$ (4.2)

The first Wolfe condition requires

$$f(\mathbf{x}_k + \alpha \mathbf{d}_k) \leq f(\mathbf{x}_k) + \beta\alpha\nabla \mathbf{d}_k f(\mathbf{x}_k)$$

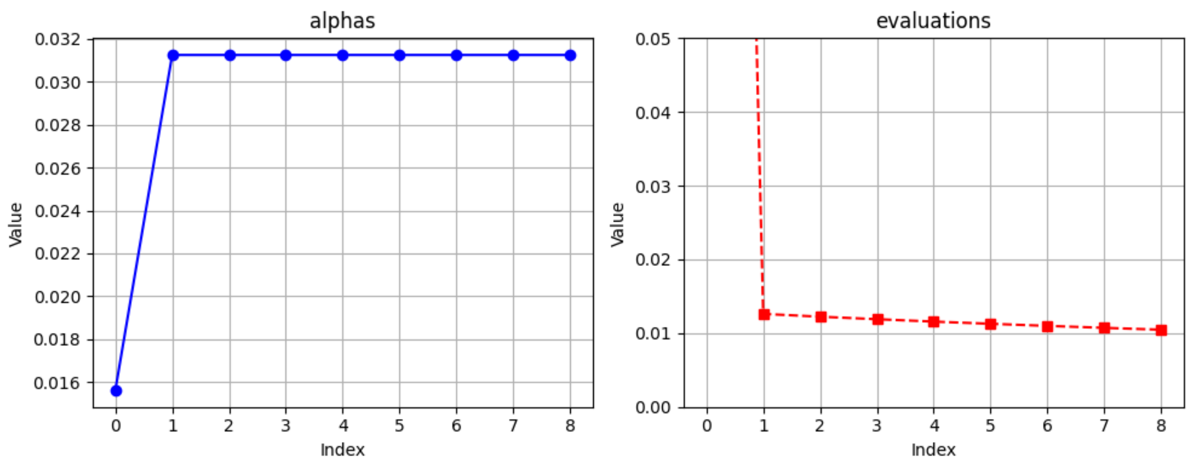
What is the maximum step length α that satisfies this condition, given that $f(\mathbf{x}) = 5 + x_1^2 + x_2^2$, $\mathbf{x}_k = [-1, -1]$, $\mathbf{d} = [1, 0]$, and $\beta = 10^{-4}$?

Exercise 6*

The steepest descent algorithm is a Descent Direction Iteration method that moves along $d_k = -\nabla f(\mathbf{x}_k)$ at every step. Program steepest descent algorithms using the backtracking line search. Use them to minimize the Rosenbrock function. Set the initial step length $\alpha_0 = 1$ and print the step length used by each method at each iteration. First, try the initial point $x_0 = [1.2, 1.2]$ and then the more difficult starting point $x_0 = [-1.2, 1]$.

Solution:

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

def rosenbrock(X: np.array, a: int=1, b: int=5):
    """
    Vectorized Rosenbrock function.

    Parameters:
        X: np.ndarray of shape (N, 2) where each row is [x0, x1]
        a: int, default 1
        b: int, default 5

    Returns:
        np.ndarray of shape (N,) with function evaluations.
    """
    X = np.atleast_2d(X) # Ensure input is always 2D
    x0, x1 = X[:, 0], X[:, 1] # Extract columns
    return (a-x0)**2 + b*(x1 - x0**2)**2

def gradient(x: np.array, a: int=1, b: int=5):
    # see page 12 of textbook
    return np.array([-2*(a-x[0])-4*b*x[0]*(x[1]-x[0]**2),2*b*(x[1]-x[0]**2)])

def backtracking_line_search(f, grad, x, d, alpha_0=1, p=0.5, beta=1e-4):
    y, g, alpha = f(x), grad(x), alpha_0
    while ( f(x + alpha * d) > y + beta * alpha * np.dot(g, d) ) :
        alpha *= p
    return alpha

def steepest_descent(f, gradient, x_0: np.array, alpha_0: float):
    S=10
    alpha = np.empty((S),dtype=float)
    alpha[0] = alpha_0
    x = np.empty((S,2),dtype=float)
    x[0]=x_0
    last = S-1
    for k in range(S-1):
        alpha[k] = backtracking_line_search(f, gradient, x[k], -gradient(x[k]),alpha_0)
        x[k+1] = x[k] - alpha[k] * gradient(x[k])
        if np.linalg.norm(x[k+1]-x[k], 2) <= 0.001:
            last = k+1
            break
    return x[0:last,:], alpha[0:last], f(x[0:last,:])

points, alphas, evaluations = steepest_descent(rosenbrock, gradient, np.array([1.2,1.2]),
    1)

print(points, alphas, evaluations)


# Create the figure with two subplots
fig, axes = plt.subplots(1, 2, figsize=(10, 4)) # 1 row, 2 columns

# First subplot
axes[0].plot(alphas, marker='o', linestyle='-', color='b', label='alphas')
axes[0].set_title('alphas')
axes[0].set_xlabel('Index')
axes[0].set_ylabel('Value')
#axes[0].set_ylim(0.028, 0.035) # Set y-axis limits
axes[0].grid(True)

# Second subplot
axes[1].plot(evaluations, marker='s', linestyle='--', color='r', label='evaluations')
```



```
axes[1].set_title('evaluations')
axes[1].set_xlabel('Index')
axes[1].set_ylabel('Value')
axes[1].set_ylim(0, 0.05) # Set y-axis limits
axes[1].grid(True)

# Adjust layout and show the plots
plt.tight_layout()


plt.savefig("steepest_descent.png")
```

Note, the exercise is not finished, one should try with different algorithms for solving the line search problem.

Exercise 7*

Descent direction methods may use search directions other than the steepest descent mentioned in the previous exercise. In general, which descent direction guarantees to produce a decrease in f?

Solution:
One that makes an angle of strictly less than $\pi/2$ radians with $-\nabla f(\mathbf{x}_k)$.
We can verify this claim by using Taylor's theorem updated in directional mode:

$$f(\mathbf{x}_k + h\mathbf{d}_k) = f(\mathbf{x}_k) + h\nabla_{\mathbf{d}_k} f(\mathbf{x}_k) + O(h^2) = f(\mathbf{x}_k) + h\mathbf{d}_k^T \nabla f(\mathbf{x}_k) + O(h^2).$$

where we used the rule (2.9) in the text book about directional derivative $\nabla_{\mathbf{d}} f(\mathbf{x}) = \nabla f(\mathbf{x})^T \mathbf{d} = \mathbf{d}^T \nabla f(\mathbf{x})$.
When $\mathbf{d}_k$ is a downhill direction, the angle θ_k between $\mathbf{d}_k$ and $\nabla f(\mathbf{x}_k)$ has $\cos\theta_k < 0$, so that

$$\mathbf{d}_k^T \nabla f(\mathbf{x}_k) = \|\mathbf{d}_k\| \, \|\nabla f(\mathbf{x}_k)\| \cos\theta_k < 0.$$

It follows that $f(\mathbf{x}_k + \mathbf{d}_k) < f(\mathbf{x}_k)$ for all positive but sufficiently small values of h.