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

February 27, 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.

Solution:

Fibonacci search is preferred when derivatives are not available.

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).

Solution:

We can use the bisection method to find the roots of $f'(x) = x - 1$. After the first update, we have $[0, 500]$. Then, $[0, 250]$. Finally, $[0, 125]$.

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?

Solution:

No, the Lipschitz constant must bound the derivative everywhere on the interval, and $f'(1) = 2(1+2) = 6$.

Exercise 4^+ (4.1)

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

Solution:

- Step-size termination condition: Consider running a descent method on $f(x) = 1/x$ for $x > 0$. The minimum does not exist and the descent method will forever proceed in the positive x direction with ever-increasing step sizes.
- Terminating based on gradient magnitude: a descent method applied to $f(x) = -x$ will also forever proceed in the positive x direction. The function is unbounded below, so neither a step-size termination condition nor a gradient magnitude termination condition would trigger.

- Termination condition to limit the number of iterations: always in effect.

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 $\beta = 10^{-4}$?

Solution:

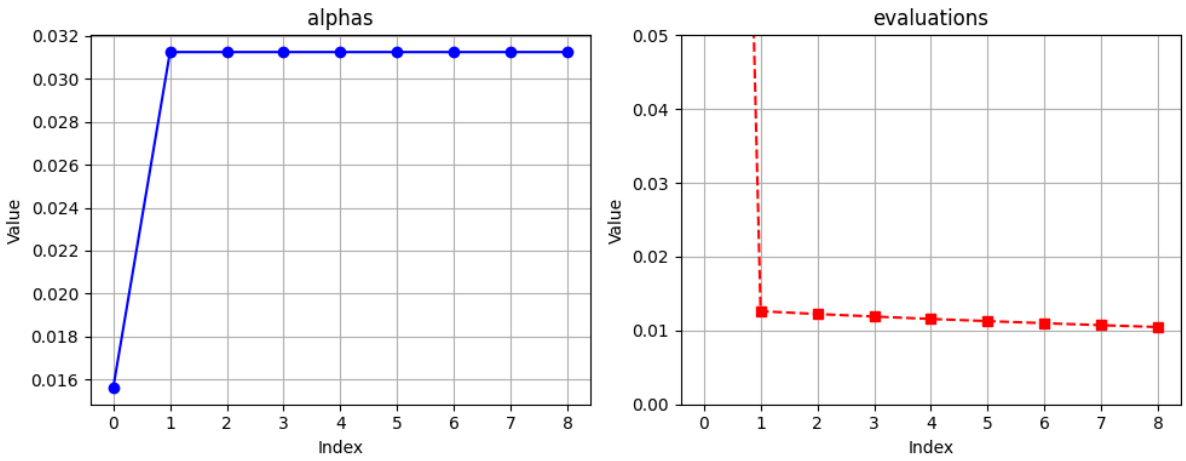
Applying the first Wolfe condition to our objective function yields $6 + (-1 + \alpha)^2 \leq 7 - 2\alpha \cdot 10^{-4}$, which can be simplified to $\alpha^2 - 2\alpha + 2 \cdot 10^{-4}\alpha \leq 0$. This equation can be solved to obtain $\alpha \leq 2(1 - 10^{-4})$. Thus, the maximum step length is $\alpha = 1.9998$.

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 $x_0 = [-1.2, 1]$.
Consider implementing and comparing also other ways for solving the line search problem and the conjugate gradient.

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.

Exercise 8*

Show that the positive definiteness of a matrix implies symmetry.

Solution:

Let the quadratic form f be defined by

$$f(\mathbf{x}) \overset{def}{=} \mathbf{x}^T A \mathbf{x}$$

where $A \in \mathbb{R}^{n \times n}$
Since $\mathbf{x}^T A \mathbf{x}$ is a scalar, then $(\mathbf{x}^T A \mathbf{x})^T = \mathbf{x}^T A \mathbf{x}$, i.e., $x^T A^T \mathbf{x} = \mathbf{x}^T A \mathbf{x}$. Hence,

$$\mathbf{x}^T \left(\frac{A - A^T}{2} \right) \mathbf{x} = 0$$

Thus, the skew-symmetric part of matrix A does not contribute anything to the quadratic form. What is left is, then, the symmetric part

$$\frac{A + A^T}{2}$$

which is diagonalizable and has real eigenvalues and orthogonal eigenvectors, all nice properties.

Exercise 9+ (5.1)

Compute the gradient of $\mathbf{x}^T A \mathbf{x} - \mathbf{b}^T \mathbf{x}$ when A is symmetric.

Solution:

The real-valued function $f : \mathbb{R}^n \to \mathbb{R}$, $f \overset{def}{=} \mathbf{x}^T A \mathbf{x} - \mathbf{b}^T \mathbf{x}$ has derivative $\frac{\mathrm{d}f(\mathbf{x})}{\mathrm{d}x}$ which is a $1 \times n$ matrix, i.e., it is a row vector. The gradient of the function $\nabla f(\mathbf{x})$ is its transpose, which is a column vector in $\mathbb{R}^n$ with components the partial derivatives of f:

$$\nabla f(\mathbf{x})_i = \frac{\partial f(\mathbf{x})}{\partial x_i}, \qquad i = 1, \ldots, n$$

A vector-valued function like the linear transformation $g : \mathbb{R}^n \to \mathbb{R}^m$, $g(x) \overset{def}{=} A\mathbf{x}$, has derivative $\frac{\mathrm{d}g(\mathbf{x})}{\mathrm{d}\mathbf{x}}$ that denotes the $m \times n$ matrix of first-order partial derivatives of the transformation from x to $g(\mathbf{x})$:

$$\frac{\mathrm{d}g(\mathbf{x})}{\mathrm{d}\mathbf{x}} = \begin{bmatrix} \frac{\partial g(\mathbf{x})_1}{\partial x_1} & \frac{\partial g(\mathbf{x})_1}{\partial x_2} & \cdots & \frac{\partial g(\mathbf{x})_1}{\partial x_n} \\ \frac{\partial g(\mathbf{x})_2}{\partial x_1} & \frac{\partial g(\mathbf{x})_2}{\partial x_2} & \cdots & \frac{\partial g(\mathbf{x})_2}{\partial x_n} \\ \vdots & & \ddots & \vdots \\ \frac{\partial g(\mathbf{x})_m}{\partial x_1} & \frac{\partial g(\mathbf{x})_m}{\partial x_2} & \cdots & \frac{\partial g(\mathbf{x})_m}{\partial x_n} \end{bmatrix}$$

Such a matrix is called the Jacobian matrix of the transformation $g(\cdot)$.

Since the ith element of g is given by:

$$g(\mathbf{x})_i \sum_{\ell=1}^{n} a_{i,\ell} x_\ell$$

it follows that

$$\frac{\partial g(\mathbf{x})_i}{\partial x_j} = a_{ij}$$

and hence that

$$\frac{\mathrm{d}g(\mathbf{x})}{\mathrm{d}\mathbf{x}} = A$$

The element-by-element calculations involve cumbersome manipulations and, thus, it is useful to derive the necessary results for matrix differentiation and have them readily available.
Some matrix derivatives useful for us are reported in the table below. They are presented alongside similar-looking scalar derivatives to help memory. This doesn't mean matrix derivatives always look just like scalar ones. In these examples, b is a constant scalar, and B is a constant matrix.

Scalar derivative	Vector derivative
$f(x) \to \frac{\mathrm{d}f}{\mathrm{d}x}$	$g(\mathbf{x}) \to \frac{\mathrm{d}g}{\mathrm{d}\mathbf{x}}$
$bx \to b$	$\mathbf{x}^T B \to B$
$bx \to b$	$\mathbf{x}^T b \to b$
$x^2 \to 2x$	$\mathbf{x}^T \mathbf{x} \to 2\mathbf{x}$
$bx^2 \to 2bx$	$\mathbf{x}^T B\mathbf{x} \to 2B$

For our specific case we have $\nabla f(\mathbf{x}) = 2Ax - b$.

Exercise 10* (5.2)

Apply one step of gradient descent to $f(x) = x^4$ from $x_0 = 1$ with both a unit step factor and with exact line search.

Solution:

The derivative is $f'(x) = 4x^3$. Starting from $x_0 = 1$ and with unit step factor:

$$f'(1) = 4 \to x_1 = 1 - 4 = -3$$
$$f'(-3) = 4\mathring{\mathrm{u}}(-27) = -108 \to x_2 = -3 + 108 = 105$$

With line search we solve $\min_\alpha\{x_k - \alpha\nabla f(\mathbf{x}_k)\}$. Starting at $x_0 = 1$:

$$f'(1) = 4 \min\{1 - \alpha 4\} = 1 \qquad \to x_1 = 1 - 1 \times 4 = -3$$
$$f'(-3) = 4\mathring{\mathrm{u}}(-27) = -108 \min\{-3 + \alpha 108\} = 0 \qquad \to x_2 = -3 + 0 \times 108 = -3$$

With line search we are not guaranteed to converge to optimum.

Exercise 11* (5.7)

In conjugate gradient descent, what is the descent direction at the first iteration for the function $f(x, y) = x^2 + xy + y^2 + 5$ when initialized at $[x, y] = [1, 1]$? What is the resulting point after two steps of the conjugate gradient method?

Solution:

The conjugate gradient method initially follows the steepest descent direction. The gradient is

$$\nabla f(x, y) = [2x + y, 2y + x]$$

which for $(x, y) = (1, 1)$ is $[3, 3]$. The direction of steepest descent is opposite the gradient, $d_0 = [-3, -3]$. It is possible to prove the following proposition:

Proposition 1 *A twice differentiable function $f : \mathbb{R}^n \to \mathbb{R}$ is* convex, *if and only if the Hessian $\nabla^2 f(x)$ is* positive semi-definite *for all $x \in \mathbb{R}^n$.*

See https://wiki.math.ntnu.no/_media/tma4180/2016v/note2.pdf for a proof.
The Hessian is

$$H = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

Since the function is quadratic and the Hessian is positive definite (you can check this by showing that both eigenvalues are positive), the conjugate gradient method converges in at most two steps. Thus, the resulting point after two steps is the optimum, $(x, y) = (0, 0)$, where the gradient is zero.