TP 4

TP4 : The Rosenbrock function

The objective of this TP is to study the function of Rosenbrock

$$\begin{aligned} f\colon \mathbb{R}^2 &\to \mathbb{R} \\ (x, y) &\mapsto (1 - x)^2 + a^2(x - y^2)^2. \end{aligned}$$

And this function will also provide an example of gradient descent algorithm application.

4.1 Preparation

1. Determine the $\nabla f(x, y)$ gradient of the f function into any $(x, y) \in \mathbb{R}^2$ point.
2. Solve system $\nabla f(x, y) = 0$ to find critical points of f. We need to find three critical points called a_1, a_2, a_3.
3. Determine the hessian matrix $\mathrm{Hess}_f(x, y)$ of the f function in any $(x, y) \in \mathbb{R}^2$ and then at a_1, a_2, a_3 points.
4. Infer the nature of the critical points a_1, a_2, a_3 by determining the eigenvalues of $\mathrm{Hess}_f(a_i)$ for $i \in \{1, 2, 3\}$.

4.2 Lab session

The results of the preparation will be checked in three ways: graphically, via symbolic calculation and numerically via gradient descent algorithm. First define the function **rosenbrock**(X, a) that returns $f(X)$ for $X \in \mathbb{R}^2$ and $a \in \mathbb{R}$.

4.2.1 Part 1 : Visual analysis

1. Define the function **graph_Rosenbrock**(a, X, Y) which takes as parameters a value a, two arrays X and Y and returns the graphic representation $(X, Y, f(X, Y))$ in three dimensions (use **meshgrid**). Test the function

with $a = 1$, and two well chosen X and Y arrays. Use **np.linspace** for lists X and Y.

2. Represent the function f with $a = 0.2$ with the same intervals X and Y as the previous question.

3. Implement a function **graph_Rosenbrock_contour**(a, X, Y, nb_lines) that gives the representation of f via **nb_lines** level lines. Use **plt.contour**.

4. By making several graphs, try to spot a bounding box of the critical points of f.

4.2.2 Part 2 : Symbolic calculations

For symbolic analysis of the f function as it was done in preparation, use module **sympy**.

1. Set three symbolic variables x, y, a, and the symbolic variable **sym_Rosenbrock** equals $f(x, y)$.

2. To determine the gradient of f, the **diff** method and the matrix layout of the result **sym.Matrix** can be used. Determine the gradient of f at (x, y), and store it in a variable **sym_gradient**.

3. Then solve the system $\nabla f(x, y) = 0$ using the **solve** module of **sympy**. You must find the critical points a_1, a_2, a_3 of the preparation.

4. Determine the $\text{Hess}_f(x, y)$ matrix using the **jacobian** method.

5. Now, determine $h_i = \text{Hess}_f(a_i)$ for $i \in \{1, 2, 3\}$. Use **subs** method.

6. Via the **eigenvalues** module, determine the eigenvalues of h_i for $i = 1, 3$.

7. Can we corroborate the results of the preparation on the nature of the critical points?

4.2.3 Partie 3 : Numerical analysis

In this part, parameter a will be set at $a = 1$.

1. Write a **rosenbrock_grad(X)** function that gives the numeric value of the gradient of f at point X for $a = 1$.

2. Compile the code presented in the notebook. What does the red arrow represent? Does the length of this arrow mean anything?

3. Recall and write the pseudocode of the update formula of the gradient descent algorithm at constant steps.

4. Create a function **gradient_descent(J_grad, x_init,alpha, max_iterations, epsilon)** that takes as input a function **J_grad** giving the gradient of a function f, a array **x_init** corresponding to the initialization of the gradient descent algorithm, a strictly positive number **alpha** corresponding to the constant step of descent, a maximum number of iteration, and a strictly positive real number **epsilon** serving as a stop test and returns

the final position x, the number of iterations, and an array with the successive positions during the execution of the algorithm to have a trace of the gradient descent path.

5. Test this function by varying the parameters **alpha, x_init** and **max_iterations**. Does the numerical solution match one of the analytical solutions of the 2nd part?

6. Execute the code proposed in the notebook and comment on the graph obtained.

7. Recall and write the pseudocode of the update formula of the optimal step gradient descent algorithm.

8. To find the optimal step for each iteration, the function **gss(f,a,b,tol)** implementing the algorithm "Golden Section Search" is given. Redo the previous questions with the optimal step gradient descent algorithm until you get the same type of chart as the 6th question. Qualitatively compare these two methods.

9. In this question, the objective, given a starting point **x_init**= (0.1.0.9), is to compare the quadratic errors (in $||\cdot||_2$, use **np.linalg.norm**) at each iteration, of both the constant step and optimal step gradient descent algorithms. Graphically represent the evolution of the quadratic error in both cases on a linear and then logarithmic scale.

10. Comment on the results obtained.