Unconstrained Optimization

DÉPARTEMENT ARCHITECTURE DES SYSTÈMES D'INFORMATION
4^{th} year
G. Gasso

Objective
Implement gradient descent and Newton method for unconstrained optimization. Compare the convergence and computation time of both algorithms.

Problem formulation

Let consider the minimization problem of the Rosenbrock's function

$$\min_{\boldsymbol{\theta} \in \mathbb{R}^2} J(\boldsymbol{\theta}) \quad \text{with} \quad J(\boldsymbol{\theta}) = (1-\theta_1)^2 + 100\left(\theta_2 - \theta_1^2\right)^2 \tag{1}$$

We will derive theoretically the solution and implement gradient descent and Newton methods to compute numerically the solution.

1 Our goal ...

1. Determine the stationary point $\boldsymbol{\theta}^*$ of $J(\boldsymbol{\theta})$.
   Hint: compute the gradient vector $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ and solve the equation $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$
2. Show that this stationary point $\boldsymbol{\theta}^*$ is a minimum of J.
   Hint: compute the Hessian matrix $\mathbf{H}(\boldsymbol{\theta})$ and check that $\mathbf{H}(\boldsymbol{\theta}^*)$ is positive definite.

2 ... and how we reach it

We want to compute numerically a solution of $\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ with the following iterative approach

- Initialize $\boldsymbol{\theta}_0$, $k = 0$
- Repeat until convergence
  - Compute the descent direction $\mathbf{h}_k$
  - Select the step size α_k
  - Update the solution $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha_k \mathbf{h}_k$; and set $k \leftarrow k+1$

2.1 Gradient descent method

1. How the direction of descent $\mathbf{h}_k$ is chosen in this case?
2. Write a function `J = mycriterion(`$\boldsymbol{\theta}$`)` that computes the value of J (see Eq. 1) given a vector $\boldsymbol{\theta}$.

```
import numpy as np

def mycriterion(theta):
  J = ...
  return J
```

3. Write a function `d = mygradient(`$\boldsymbol{\theta}$`)` that calculate the gradient of the function J (1)

```python
def mygradient(theta):
  gradJ = ...
  return gradJ
```

4. The contours the J can be shown as hereafter. The initial vector $\boldsymbol{\theta}_0$ is provided below (you may change it)

```python
import matplotlib.pyplot as plt
# contour plot of rosenbrock function
n = 100
points_x1, points_x2 = np.meshgrid(np.linspace(-1.25, 1.5, n), np.
    linspace(-1.75, 1.5, n))
f = (1-points_x1)**2 + 100*((points_x2 - points_x1**2)**2)
f = f.reshape(points_x1.shape)
levels = np.concatenate((np.array([0, 1]), np.arange(5, 45, 5)))
fig = plt.figure(1, figsize=(8,4))
cp = plt.contourf(points_x1, points_x2, f, levels, alpha=0.95, cmap="RdBu
    ")
plt.colorbar()

# initial vector
theta0 = np.array([-1.0, 0.0])
plt.figure(fig.number)
plt.scatter(theta0[0], theta0[1], marker="o", color="k", facecolor="k", s
    =150)
plt.text(-1.1, -0.5, r"${\theta}_0$", {"color": "k", "fontsize": 20})
plt.xticks(fontsize=16), plt.yticks(fontsize=16)
```

5. Complete your script in order to implement the gradient descent method. The convergence criterion will be $\|\nabla J(\boldsymbol{\theta})\| \leq 10^{-3}$ or a maximum number of iterations is reached. Test your algorithm either with a fixed step size $\alpha_k = \alpha$ and α_k computed using the backtracking method (apply the Armijo's rule).

```python
from scipy.linalg import norm

# maximal number of iteration
iter_max = 2500
# threshold on the norm of the gradient
thresh = 1e-2
# store ongoing results
history_J = np.empty(iter_max)
history_theta = np.empty((theta0.shape[0], iter_max))
# initialization
iter = 0
theta = theta0.copy()
# store the initial theta and related gradient and criterion
history_theta[:,iter] = theta0
history_J[iter] = mycriterion(theta0)
grad = mygradient(theta0)
while (iter <= iter_max-2) and (norm(grad) > thresh):
  # compute descent direction
  direction = ...
```

```
# select the step size alpha
alpha = ...

# update the solution
theta += alpha*direction

# increase iteration number
iter += 1

# store the current solution and criterion
history_theta[:,iter] = theta
history_J[iter] = ...

# compute the new gradient
grad = ...
```

6. Plot the evolution of J over iterations. Compare the obtained solution $\hat{\boldsymbol{\theta}}$ at convergence with the optimal one $\boldsymbol{\theta}^*$.

7. Comment on the convergence speed of the algorithm and the quality of the solution.

2.2 Newton method

We want to compute the solution of problem (1) using Newton method.

1. Write a function `H = myhessian(`θ`)` in order to compute the Hessian matrix

```
def myhessian(theta):
 HessianJ = ...
return HessianJ
```

2. Inspiring from the gradient descent method, complete your script by the implementation of Newton method.
   Hint : you will soon notice that the $\mathbf{H}(\boldsymbol{\theta})$ matrix is not always positive definite. To circumvent it, regularize the optimization problem by considering instead $\mathbf{H} \leftarrow \mathbf{H} + \lambda \mathbf{I}$ with $\lambda > 0$ a fixed parameter to be chosen.

3. Compare the convergence speed with the previous case.