Numerical Optimization with Python

Assignment 01 - programming part: Gradient Descent with fixed step size

In this exercise we will:

- Implement Gradient Descent iterative algorithm for unconstrained minimization.
- Test it on several examples and get some impression of its behavior and some drawbacks
- Organize our project so it is ready for further extensions throughout the course
- Learn how to use one of Python's testing frameworks

1. Instructions for project organization:
    a. Your numerical optimization project should have two directories: `src` and `tests`.
    b. Your `src` directory should have two modules: `unconstrained_min.py` (your algorithms) and `utils.py` (common functions such as plotting, printouts to console, etc.)
    c. Your `tests` directory should have two modules: `test_gradient_descent.py` and `examples.py`

2. Requirements for implementing Gradient Descent:
    a. The function should be implemented in `unconstrained_min.py`. The function signature: `gradient_descent(f, x0, step_size, obj_tol, param_tol, max_iter)`
    b. `f` is the function minimized, `x0` is the starting point, `step_size` is the coefficient multiplying the gradient vector in the algorithm update rule, `max_iter` is the maximum allowed number of iterations.
    c. `obj_tol` is the numeric tolerance for successful termination in terms of small enough change in objective function values, between two consecutive iterations ($f(x_{i+1})$ and $f(x_i)$).
    d. `param_tol` is the numeric tolerance for successful termination in terms of small enough distance between two consecutive iterations iteration locations (x_{i+1} and x_i).
    e. At each iteration, the algorithm reports (prints to console) the iteration number i, the current location x_i, the current objective value $f(x_i)$, the current step length taken $\|x_i - x_{i-1}\|$ and the current change in objective function value $|f(x_i) - f(x_{i-1})|$. The iteration reporting function should be implemented in `utils.py` and be invoked by the algorithm.

f. The function returns the last location and a success/failure Boolean flag, according to the termination conditions (either one of the tolerances successfully achieved or maximal number of iterations achieved first). The success/failure status should be printed to console in human readable form describing the result (which convergence/failure, etc.).

3. Requirements for implementing `examples.py`:
   a. All examples in this exercise will be functions taking a vector `x` and returning two output values: the scalar function value evaluated at `x` and the vector valued gradient at `x`.
   b. Implement three quadratic examples: $f(x) = x^T Q x$ for the following Q's:
      i. $Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
      ii. $Q = \begin{bmatrix} 5 & 0 \\ 0 & 1 \end{bmatrix}$
      iii. $Q = \begin{bmatrix} \frac{\sqrt{3}}{2} & -0.5 \\ 0.5 & \frac{\sqrt{3}}{2} \end{bmatrix}^T \begin{bmatrix} 5 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{\sqrt{3}}{2} & -0.5 \\ 0.5 & \frac{\sqrt{3}}{2} \end{bmatrix}$
   c. Implement the Rosenbrock function: $f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$.
   d. Implement a linear function $f(x) = a^T x$ for some nonzero vector a you choose.

4. Requirements for implementing `test_gradient_descent.py`:
   a. See the very first, basic example in https://docs.python.org/3/library/unittest.html for test module structure using Python's `unittest` framework.
   b. For each of the tests described next, the path taken by the algorithm should be plotted, overlaid on a plot of function contours. See for example: https://matplotlib.org/stable/gallery/images_contours_and_fields/contour_demo.html for contour plot reference. The plotting function (contours and path) should be implemented in `utils.py` and invoked by the tests.
   c. Implement `test_quad_min()` which should minimize your quadratic examples from `examples.py`. Set the starting point to $(1, 1)$, the step size to 0.1, maximum iterations to 100, step tolerance to 10^{-8} and objective tolerance to 10^{-12}. Play a little with different step lengths to get an impression of the resulting behavior.
   d. Implement `test_rosenbrock_min()` which should minimize your Rosenbrock example from `examples.py`. This example is challenging to optimize. Play with some choices of

parameters and observe the behavior and the difficulties. For submission, you should arrive near $(1,1)$ as the minimum, when you set the starting point to $(2,2)$, the step size to 0.001, maximum iterations to 10000, step tolerance to 10^{-8} and objective tolerance to 10^{-7}.

e. Implement `test_lin_min()` which should attempt (and fail) to minimize your linear example.

5. Submission instructions:
    a. Submit a single file, your report in PDF format.
    b. Your report should include the plots created by each of your tests (contours and iteration paths)
    c. For each test – your report should include the last iteration report printed to console (the details of your final iterate and success/failure algorithm output flag).
    d. (Your report should not include your code)

Good luck!