02610 Optimization and Data Fitting

Exercises for Week 2

1 Search directions

1.1 (by hand)

Investigate whether the direction of $\boldsymbol{p} = [2, -1]^T$ is a descent direction with respect to the function

$$f(\boldsymbol{x}) = f([x_1, x_2]) = x_1^2 + x_1 x_2 - 4x_2^2 + 10$$

at $\boldsymbol{x} = [1, 1]^T$.

1.2 (by hand)

Consider the function

$$f(\boldsymbol{x}) = f(x_1, x_2) = (x_1 + x_2^2)^2.$$

At the point $\boldsymbol{x} = [1, 0]^T$ we consider the search direction $\boldsymbol{p} = [-1, 1]^T$. Show that $\boldsymbol{p}$ is a descent direction and find all minimizers of the problem

$$\min_{\alpha > 0} f(\boldsymbol{x} + \alpha \boldsymbol{p}).$$

2 Steepest descent method and Newton's method

2.1 (by hand)

Consider the problem

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) = f([x_1, x_2]) = x_1^2 + x_1 x_2 - 4x_2^2 + 10.$$

Using a calculator (or a computer, if you prefer) to perform one iteration of the steepest descent method with exact line search, starting at $\boldsymbol{x}_0 = [1, 0]$.

2.2 (by hand)

Consider the problem

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) = (x_1 + 2x_2 - 3)^2 + (x_1 - 2)^2.$$

Using a calculator (or a computer, if you prefer) to perform one iteration of Newton's method with exact line search, starting at $\boldsymbol{x}_0 = [0, 0]^T$.

2.3 (in Python)

In this exercise, we implement the steepest descent method and Newton's method. We use a simple univariate function

$$f(x) = x - \mu \log(x), \quad \mu = 1, \quad x > 0 \tag{1}$$

as the objective function to test our implementations. x^* denotes its true minimizer, and what is it?
First, we write a Python function, `PenFun1`, to return the function value of (1) and its first and second order derivatives. The function file can be found in the same folder as this file in DTU Learn.
In the following, we will implement the steepest descent algorithm and Newton's method with a given step length and test it on the problem (1) using a command such as (using the steepest descent algorithm)

```
x0 = 0.1
mu = 1.0
xopt, stat = steepestdescent_method(alpha, PenFun1, x0, mu)
```

where `alpha` is a fixed step length given as input.

1. Implement the steepest descent algorithm to find a minimizer. You can do that by completing the following Python code:

```
import numpy as np

def steepestdescent_method(alpha, fundfun, x0, *args):
    # Convert x0 to a numpy array for consistency
    x = np.array(x0, ndmin=1)
    is_scalar = x.size == 1

    # Solver settings and info
    maxit = 100 * (1 if is_scalar else len(x))
    tol = 1.0e-10

    # Initial iteration
    stat = {"converged": False, "nfun": 0,
                      "iter": 0, "X": [x.copy()], "F": [], "dF": []}
    it = 0
    f, df, _ = fundfun(x, *args)
    converged = (np.linalg.norm(df, np.inf) <= tol)

    # Store data for plotting
    stat["F"].append(f)
    stat["dF"].append(df.copy())

    # Main loop of steepest descent
```

```
while not converged and (it < maxit):
    it += 1

    # Steepest descent step
    # TODO -- Insert code between the lines
    # ================================================




    # ================================================

    f, df, _ = fundfun(x, *args)
    converged = (np.linalg.norm(df, np.inf) <= tol)

    # Store data for plotting
    stat['X'].append(x.copy())
    stat['F'].append(f)
    stat['dF'].append(df.copy())
    stat['nfun'] += 1

stat['iter'] = it
# Prepare return data
if not converged:
    stat['converged'] = converged
    return None, stat
stat['converged'] = converged
# Convert the solution back to a scalar if the input was a scalar
x_result = x[0] if is_scalar else x
return x_result, stat
```

(a) Using the starting point $x_0 = 0.1$ and the step length $\alpha = 0.1$, make a table of k, x_k, $e_k = |x_k - x^*|$, $|f'(x_k)|$ and $f(x_k)$. Also plot $|f'(x_k)|$, $f(x_k)$ and e_k as function of the iteration number.

(b) Try your code for different positive starting points x_0 with the step length $\alpha = 0.1$. Plot $|f'(x_k)|$, $f(x_k)$ and e_k as function of the iteration number.

(c) Try different step length α, and plot $|f'(x_k)|$, $f(x_k)$ and e_k as function of the iteration number.

2. Implement the Newton's method with the step length 1 to find the minimizer. You should also make a table of k, x_k, $e_k = |x_k - x^*|$, $|f'(x_k)|$ and $f(x_k)$. (Hint: you just need to make a few modifications to the steepest descent code. Call the new Python function as `newton`. To solve a linear system $A\boldsymbol{x} = \boldsymbol{b}$, you can simply use `x = numpy.linalg.solve(A, b)`.)

3. Compare the results from the steepest descent method with the starting point $x_0 = 0.01$ and the step length $\alpha = 0.1$ with Newton's methods from the same

starting point. Comment on the convergence of the both methods. Comment on advantages and dis-advantages of each method.

2.4 (in Python)

In this exercise, we use a multivariable function:

$$f(\boldsymbol{x}) = f(x_1, x_2) = \frac{1}{2}x_1^2 + 5x_2^2 \tag{2}$$

as the objective function to test our implementations. $\boldsymbol{x}^*$ denotes its true minimizer, and what is it?

1. First, we write a Python function called `MvFun` to return the function value of (2) and its gradient and Hessian.
2. Use your implementation of the steepest descent method with the starting point $\boldsymbol{x}_0 = [10, 1]^T$ and the step length $\alpha = 0.05$ to find the minimizer of (2). Plot $f(\boldsymbol{x}_k)$, e_k and $\|\nabla f(x_k)\|_2$ as functions of the iteration number.
3. Use your implementation of Newton's method with the starting point $\boldsymbol{x}_0 = [10, 1]^T$ and the step length $\alpha = 1$ to find the minimizer of (2). Plot $f(\boldsymbol{x}_k)$, e_k and $\|\nabla f(x_k)\|_2$ as functions of the iteration number.
4. Comment on the convergence of the both methods. Comment on advantages and dis-advantages of each method.

2.5 (in Python)

In this exercise, we will improve your Python implementations for the steepest descent method and Newton's method by incorporating the backtracking line search.

1. Modify your Python function for the steepest descent method to include the backtracking line search, which can be found in the textbook page 37 or in the slides. You can name your new Python function as `steepestdescent_line`. In the backtracking line search, we can set $\rho = 0.5$ and $c = 0.1$. In addition, the initial guess of the step length in the backtracking, i.e., $\bar{\alpha}$, is always set as 1.
2. Add backtracking line search to Newton implementation, and name it as `newton_line`.
3. Test the steepest descent method and the Newton's method with backtracking line search on the function (2). Plot $f(\boldsymbol{x}_k)$, e_k, $\|\nabla f(x_k)\|_\infty$ as well as α_k as functions of the iteration number. Compare the results with the ones in Exercise 2.4.

3 Convergence rate

3.1 (by hand)

Show that the sequence $x_k = 1 + (0.5)^{2^k}$ is Q-quadratically convergent to 1.

3.2 (by hand)

Does the sequence $x_k = 1/k!$ converge Q-superlinearly or Q-quadratically?