02610 Optimization and Data Fitting

Exercises for Week 7

1 The Levenberg-Marquardt solution (by hand)

1. Express the solution $\boldsymbol{p}$ of

$$(J^T J + \lambda I)\boldsymbol{p} = -J^T \boldsymbol{r}$$

in terms of the singular-value decomposition of J and the scalar λ.

2. Express its squared-norm $\|\boldsymbol{p}\|_2^2$ in these same terms, and show that

$$\lim_{\lambda \to 0} \boldsymbol{p} = -\sum_{\sigma_i \neq 0} \frac{\boldsymbol{u}_i^T \boldsymbol{r}}{\sigma_i} \boldsymbol{v}_i.$$

2 The Gauss-Newton method (in Python)

Consider a simple problem with $n = 1$ and $m = 2$

$$\min_{x \in \mathbb{R}} f(x) = \frac{1}{2}(x+1)^2 + \frac{1}{2}(\lambda x^2 + x - 1)^2,$$

that is, the residual is

$$\boldsymbol{r}(x) = \begin{bmatrix} x+1 \\ \lambda x^2 + x - 1 \end{bmatrix}.$$

1. (by hand) Calculate f' and f''. Show that $x = 0$ is a stationary point for f. Further, show that if $\lambda < 1$, then $x = 0$ is a local minimizer.

2. (by hand) Calculate the Jacobian J of f, and check if your calculation is correct according to

$$f'(x) = J(x)^T \boldsymbol{r}(x).$$

3. (in Python) Write a Python function to compute the residual $\boldsymbol{r}$ and the Jacobian J. You can start this function with

```
def fun_rJ_Q2(x, args):
```

where `args` stores the value of λ.

4. (in Python) Download the Python function `GaussNewton_line` and save in the same folder as `fun_rJ_Q2.py` written in the previous question. Set $\lambda = 0$ and $x_0 = 0.1$, and turn on the backtracking line search, then call

```
xopt, stat = GaussNewton_line(fun_rJ_Q2, flag_line, x0, lambda)
```

Plot $e_k = |x_k - 0|$, $|f'(x_k)|$ and $f(x_k)$ as functions of the iteration number. You should see that your method find the minimizer in one iteration.

5. (in Python) Set $\lambda = 0.1$ and $x_0 = 0.1$ with the backtracking line search. If you plot e_{k+1}/e_k, can you see that it converges linearly?

6. (in Python) Set $\lambda = -2$, $x_0 = 0.1$ and **without** the line search. Then, what happens now? If we use the backtracking line search, would it become better?

Explanations: By substituting J into the Gauss-Newton iteration step and ignore high order terms on x, we will get

$$x_{k+1} = x_k + (\lambda - 1)x_k + O(x_k^2) = \lambda x_k + O(x_k^2).$$

Thus, if $|\lambda| < 1$, we will have linear convergence. If $\lambda < -1$, the Gauss-Newton method cannot find the minimizer.

3 Exponential Fit (in Python)

In this exercise, we try to fit the data in `data_exe3.mat` with the function

$$\phi(\boldsymbol{x}, t) = x_1 e^{-x_3 t} + x_2 e^{-x_4 t}.$$

1. (by hand) Calculate the Jacobian J of the objective function f for your LSQ data fitting problem

$$\min_{\boldsymbol{x} \in \mathbb{R}^4} f(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{r}(\boldsymbol{x})\|_2^2 = \frac{1}{2}\sum_{i=1}^{m} \left(y_i - \phi(\boldsymbol{x}, t_i)\right)^2.$$

2. (in Python) Write a Python function to compute the residual $\boldsymbol{r}$ and the Jacobian J. You can start this function with

```
def fun_rJ_Q3(x, t, y):
```

3. Set $\boldsymbol{x}_0 = [1, -1, 1, 2]^T$, and call `GaussNewton_line` with the backtracking line search to solve this nonlinear data fitting problem. Plot $\|\nabla f(\boldsymbol{x}_k)\|_2$ and $f(\boldsymbol{x}_k)$ as functions of the iteration number. Which solution do you get?

4. Change the starting point into $\boldsymbol{x}_0 = [3, -3, 3, 3]^T$, and apply Gauss-Newton method again. What happens now? Why?

5. Implement Levenberg-Marquardt method. You can do that by completing the following Python code, which can be downloaded from DTU Learn:

```
import numpy as np
from linearLSQ import linearLSQ

def Levenberg_Marquardt_method(fun_rJ, x0, *args):
    # Determine if x0 is a scalar or an array
    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
# --------- TODO: Calculate the Jacobian Jx, the residual rx,
#                   the function value f and the gradient df.
rx, Jx =
f =
df =

converged = (np.linalg.norm(df, np.inf) <= tol)
stat["nfun"] += 1

# Initial lambda
lambda_val = np.linalg.norm(np.dot(Jx.T, Jx))

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

# Main loop of L-M method
while not converged and it < maxit:
    it += 1

    # --- TODO: Calculate the search direction by solving a linear LSQ problem
    A =
    b =
    p = linearLSQ(A, b).flatten()

    # --- TODO: Update the iterate, Jacobian, residual, f
    x_new =
    rx_new, Jx_new =
    f_new =

    # --- TODO: Update the Lagrange parameter lambda_val



    # Accept or reject x_new
    if rho > 0:
        x = x_new
        rx = rx_new
```

```
            f = f_new
            Jx = Jx_new
            df = np.dot(Jx.T, rx).flatten()

        # check if it is converged
        converged = (np.linalg.norm(df, np.inf) <= tol)
        stat["nfun"] += 1

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

    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
```

6. Set the starting point as $\boldsymbol{x}_0 = [3, -3, 3, 3]^T$, and apply Levenberg-Marquardt method. Does the method converge? If yes, which solution do you get?

   Set the last iterate as $\boldsymbol{x}^*$, and calculate $\boldsymbol{e}_k = \|\boldsymbol{x}_k - \boldsymbol{x}^*\|_2$. Plot $\boldsymbol{e}_{k+1}/\boldsymbol{e}_k$ with respect to k. In the last a few iterations, can you see that the method converge superlinearly?

   Plot all data as points and the fit function as a curve. Are you satisfied with your fit function?

4 `scipy.optimize.least_squares`

1. Read about `scipy.optimize.least_squares` from the link: `https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.least_squares.html#scipy.optimize.least_squares`.

2. Use the default setting to solve the data fitting problem in Question 3 with the starting point $[3, -3, 3, 3]^T$. Plot the fit function.

3. Change the algorithm to `'lm'`, i.e., Levenberg Marquardt method, and solve the same problem again, and plot the fit function in the same figure.

4. Figure out how to supply the Jacobian, $J(\boldsymbol{x})$, to `least_squares`. Then, apply the Levenberg-Marquardt method with given $J(\boldsymbol{x})$ to solve the same problem

again, and compare the solution that you obtained with the one in Question 3.6.

(If you really cannot figure out how to supply the Jacobian, you can download `Test_lsq.py` in DTU Learn and see how I would do it.)