Nonlinear Optimization – Sheet 06

Exercise 1

We use the same code for visualization_functions.py, armijo_procedures.py, example_functions.py and globalized_newton_UP.py as in sheet 5.

We implement algorithm 5.41:

```
import numpy as np

def truncated_CG(A, b, Minv, eps_rel):
    "return approximate solution of Ad = b"

    #l = 0
    d = 0
    zeta = -b
    p = -Minv @ zeta
    delta = - np.dot(zeta, p)
    gamma = delta
    while delta >= eps_rel**2 * delta:
        q = A @ p
        theta = np.dot(q, p)
        if theta > 0:
            alpha = delta/theta
            d = d + alpha * p
            zeta = zeta + alpha * q
            new_p = - Minv @ zeta
            new_delta = - np.dot(zeta, new_p)
            beta = new_delta/delta
            delta = new_delta
            gamma = delta + beta**2 * gamma
            p = new_p + beta*p
        else:
            break
    return d, gamma
```

. By slightly twisting globalized_newton_UP.py we obtain algorithm 5.44.

```
import numpy as np
from armijo_procedures import armijo_backtracking
from truncated_CG import truncated_CG

def globalized_inexact_newton_UP(
    x_0, f, f_prime, f_two_prime, M_inv, eta_k, sigma, eta, rho, p,
        beta, eps=1e-5, max_iter=100
):
    """
    Implements Algorithm 5.30
    """
    debug = False

    k = 0
    f_k = f(x_0)
    r = f_prime(x_0)
    d_G = -M_inv @ r
    delta = -r.transpose() @ d_G
    history = {
```

```
        "iterates": [x_0],
        "objective_values": [f_k],
        "gradient_norms": [np.sqrt(delta)],
        "step_lengths": [],
    }
    x = x_0
    while delta > eps**2 and k < max_iter:

        if debug == True:
            print(f'solve {f_two_prime(x)} * x = {-r}')
        #d_N = np.linalg.solve(f_two_prime(x), -r) #exact
        d_N, d_N_norm_squared = truncated_CG(f_two_prime(x), -r, M_inv,
            eps_rel=eta_k(k)) #inexact

        if np.dot(f_prime(x), d_N) <= - min(eta, rho * np.linalg.norm(
            d_G)**p) * np.sqrt(delta) * np.sqrt(d_N_norm_squared):
            d = d_N
        else:
            d = d_G

        phi = lambda alpha: f(x + alpha * d)
        phi_0 = f_k  # f(x + 0*d) = f(x)
        phi_prime_0 = -delta

        if debug == True:
            case = ''
            if np.array_equiv(d, d_G):
                case = 'gradient'
            elif np.array_equiv(d, d_N):
                case = 'newton'
            else:
                case = 'bug!'
            print(f'{case}, phi_0={phi_0}, phi_prime_0={phi_prime_0} at
                x={x} and d={d}')
        alpha = armijo_backtracking(1, phi, phi_0, phi_prime_0, sigma,
            beta) #initial trial step size is 1

        x = x + alpha * d
        f_k = f(x)
        r = f_prime(x)
        d_G = -M_inv @ r
        delta = -r.transpose() @ d_G
        k = k + 1

        history["step_lengths"].append(alpha)
        history["iterates"].append(x)
        history["objective_values"].append(f_k)
        history["gradient_norms"].append(np.sqrt(delta))

    return history
```

. In 1.py, we implement the code to test algorithm 5.44 on the rosenbrock function and to generate plots (see floating figure).

```
import numpy as np
import matplotlib.pyplot as plt
```

```
from visualization_functions import (
    plot_2d_iterates_contours,
    plot_f_val_diffs,
    plot_step_sizes,
    plot_grad_norms,
)
from globalized_inexact_newton_UP import globalized_inexact_newton_UP
from globalized_newton_UP import globalized_newton_UP
from example_functions import rosenbrock

a = 1
b = 100
rosenbrock_f = lambda x: rosenbrock(a, b, x)[0]
rosenbrock_prime = lambda x: rosenbrock(a, b, x)[1]
rosenbrock_two_prime = lambda x: rosenbrock(a, b, x)[2]

configurations = [
    ([1, 2], 1e-4),
    ([2, 1], 1e-4),
    ([2.5, 2], 1e-4),
    ([0, 4], 1e-4),
]
rosenbrock_histories_n = []
rosenbrock_labels_n = []

for configuration in configurations:
    rosenbrock_histories_n.append(
        globalized_newton_UP(
            configuration[0],
            rosenbrock_f,
            rosenbrock_prime,
            rosenbrock_two_prime,
            np.identity(2),
            sigma=1e-4,
            eta=.5,
            rho=1e-6,
            p=.1,
            beta=.5,
            eps=1e-10,
            max_iter=100,
        )
    )
    rosenbrock_labels_n.append(f"x0: {configuration[0]}, sigma:{
       configuration[1]}")

rosenbrock_histories_ni = []
rosenbrock_labels_ni = []
for configuration in configurations:
    rosenbrock_histories_ni.append(
        globalized_inexact_newton_UP(
            configuration[0],
            rosenbrock_f,
            rosenbrock_prime,
            rosenbrock_two_prime,
            np.identity(2),
            eta_k = lambda _ : 1e-3,
```

```
            sigma=1e-4,
            eta=.5,
            rho=1e-6,
            p=.1,
            beta=.5,
            eps=1e-10,
            max_iter=1000,
        )
    )
    rosenbrock_labels_ni.append(f"x0: {configuration[0]}, sigma: {
        configuration[1]}")

plot_grad_norms(
    histories=rosenbrock_histories_n,
    labels=rosenbrock_labels_n,
)
plot_grad_norms(
    histories=rosenbrock_histories_ni,
    labels=rosenbrock_labels_ni
)

plot_2d_iterates_contours(
    rosenbrock_f,
    histories=rosenbrock_histories_n,
    labels=rosenbrock_labels_n,
    xlims=[-3, 3],
    ylims=[0,7],
    title="Iterates and iso-lines of Rosenbrock function for globalized
        newton"
)
plot_2d_iterates_contours(
    rosenbrock_f,
    histories=rosenbrock_histories_ni,
    labels=rosenbrock_labels_ni,
    xlims=[-3, 3],
    ylims=[-2,5],
    title="Iterates and iso-lines of Rosenbrock function for globalized
        inexact newton"
)
plt.show()
```

Exercise 2

Let $f : \mathbb{R}^n \to \mathbb{R}$ be differentiable, $A \in \mathrm{GL}_n(\mathbb{R}), b \in \mathbb{R}^n$ and $g(y) = f(Ax + b)$.

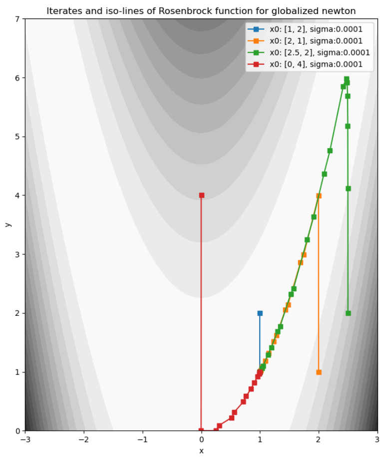
(i) Let $(x^{(k)}), (y^{(k)})$ be generated by a applying full quasi Newton steps as in

$$x^{(k+1)} = x^{(k)} + (H_f^{(k)})^{-1} f'(x^{(k)})^T, \quad \text{from } x^{(0)} = Ay^{(0)} + b,\ H_f^{(0)} \text{ spd}$$
$$y^{(k+1)} = y^{(k)} + (H_g^{(k)})^{-1} f'(y^{(k)})^T, \quad \text{from } y^{(0)},\ H_g^{(0)} = A^T H_f^{(0)}.$$

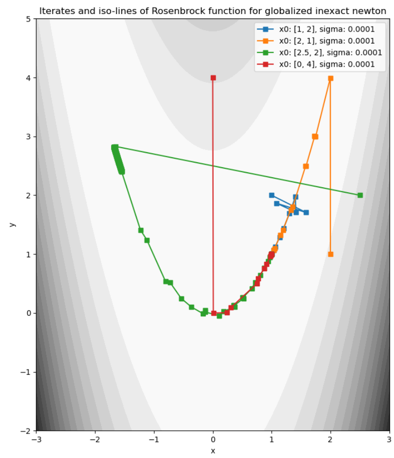
Show $x^{(k)} = Ay^{(k)} + b$ for all $k \in \mathbb{N}$ when BFGS or DFP are applied to update the model Hessian.

Proof. We induct on k. The case $k = 0$ is true by assumption. From now on we omit brackets in supersctipt indices. By induction hypothesis and chain rule

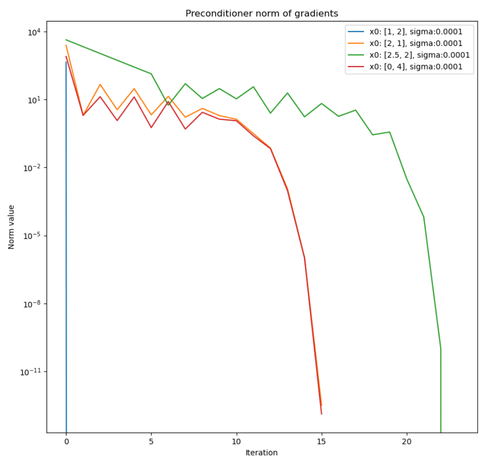
$$Ay^{k+1} + b = x^k - A(H_g^k)^{-1} g'(y^k)^T = x^k - A(H_g^k)^{-1} A^T f'(x^k)^T.$$



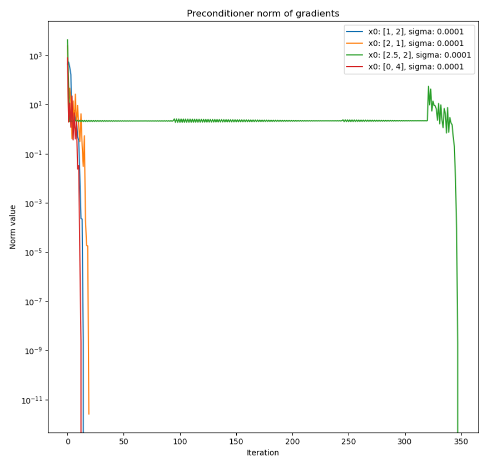
(a) Iterates for the globalized newton method



(b) Iterates for the globalized inexact newton method



(c) gradient norms for the globalized newton method



(d) gradient norms for the globalized inexact newton method

Thus, it suffices to show $A(H_g^k)^{-1}A^T = (H_f^k)^{-1}$ for all $k \in \mathbb{N}$. We show this by induction on k. For $k = 0$ this is true by assumption.

- DFP: We use the update formula for the inverse model hessians. Let $\eta^k = \nabla f(x^{k+1}) - \nabla f(x^k)$, $s^k = x^{k+1} - x^k$ and $\lambda^k = \nabla g(y^{k+1}) - \nabla g(y^k)$, $t^k = y^{k+1} - y^k$ and we set $B = (H_g^k)^{-1}$. By the formula for DFP and induction

$$\begin{aligned}(H_f^{k+1})^{-1} &= ABA^T - \frac{ABA^T\eta^k(\eta^k)^TABA^T}{(\eta^k)^TABA^T\eta^k} + \rho s^k(s^k)^T \\ &= A\left(B - \frac{BA^T\eta^k(A^T\eta^k)^TB}{(A^T\eta^k)^TBA^T\eta^k} - \rho A^{-1}s^k(s^k)^TA^{-T}\right)A^T,\end{aligned}$$

where $\rho = 1/(\eta^k)^Ts^k$ and a direct computation shows $s^k = At^k$, thus $\rho = 1/(A^T\eta^k)^Tt^k$ and

$$(H_f^{k+1})^{-1} = A\left(B - \frac{BA^T\eta^k(A^T\eta^k)^TB}{(A^T\eta^k)^TBA^T\eta^k} - \frac{1}{(A^T\eta^k)^Tt^k}t^k(t^k)^T\right)A^T$$

inspecting this equation, we see that it suffices to show $A^T\eta^k = \lambda^k$. By induction hypothesis we know $A(H_g^k)^{-1}A^T = (H_f^k)^{-1}$ which shows $Ay^{k+1} + b = x^{k+1}$ using this information we see

$$A^T\eta^k = A^T\nabla f(Ay^{k+1} + b) - A^T\nabla f(Ay^k + b) = \nabla g(y^{k+1}) - \nabla g(y^k) = \lambda^k$$

implying the result.

- BFGS: We use the update formula for the inverse model hessians. Let $\eta^k = \nabla f(x^{k+1}) - \nabla f(x^k)$, $s^k = x^{k+1} - x^k$ and $\lambda^k = \nabla g(y^{k+1}) - \nabla g(y^k)$, $t^k = y^{k+1} - y^k$ and we set $B = (H_g^k)^{-1}$. As above we have $A^T\eta^k = \lambda^k$, $s^k = At^k$. By the formula for BFGS and induction:

$$\begin{aligned}(H_f^{k+1})^{-1} =& ABA^T + \rho(s^k - ABA^T\eta^k)(s^k)^T \\ &+ \rho s^k(s^k - ABA^T\eta^k)^T - \rho^2(s^k - ABA^T\eta^k)^T\eta^k s^k(s^k)^T \\ &= A\Big(B + \rho(t^k - B\lambda^k)(t^k)^T \\ &+ \rho t^k(t^k - B\lambda^k)^T - \rho^2(t^k - B\lambda^k)^T\lambda^k t^k(t^k)^T\Big)A^T\end{aligned}$$

where we used that $\rho^2(t^k - B\lambda^k)^T\lambda^kA = A\rho^2(t^k - B\lambda^k)^T\lambda^k$ because $\rho^2(t^k - B\lambda^k)^T\lambda^k$ is just a number. And the term in the big blue brackets is just $(H_g^{k+1})^{-1}$.

□

(ii) Let $(x^{(k)}), (y^{(k)})$ be generated by a applying full inverse quasi Newton steps as in

$$\begin{aligned}x^{(k+1)} &= x^{(k)} + (B_f^{(k)})f'(x^{(k)})^T\,, &&\text{from } x^{(0)} = Ay^{(0)} + b, && B_f^{(0)} \text{ spd} \\ y^{(k+1)} &= y^{(k)} + (B_g^{(k)})f'(y^{(k)})^T\,, &&\text{from } y^{(0)}, && B_g^{(0)} = A^{-1}B_f^{(0)}A^{-T}.\end{aligned}$$

Show $x^{(k)} = Ay^{(k)} + b$ for all $k \in \mathbb{N}$ when BFGS or DFP are applied to update the model Hessian.

Proof. We induct on k. The case $k = 0$ is true by assumption. From now on we omit brackets in supersctipt indices. By induction hypothesis and chain rule

$$Ay^{k+1} + b = x^k - AB_g^kg'(y^k)^T = x^k - AB_g^kA^Tf'(x^k)^T.$$

Thus, it suffices to show $AB_g^kA^T = B_f^k$ for all $k \in \mathbb{N}$. We show this by induction on k. For $k = 0$ this is true by assumption.

- DFP: We use the update formula for the inverse model hessians. Let $\eta^k = \nabla f(x^{k+1}) - \nabla f(x^k)$, $s^k = x^{k+1} - x^k$ and $\lambda^k = \nabla g(y^{k+1}) - \nabla g(y^k)$, $t^k = y^{k+1} - y^k$ and we set $B = B_g^k$. By the formula for DFP and induction

$$\begin{aligned} B_f^{k+1} &= ABA^T - \frac{ABA^T\eta^k(\eta^k)^T ABA^T}{(\eta^k)^T ABA^T\eta^k} + \rho s^k(s^k)^T \\ &= A\left(B - \frac{BA^T\eta^k(A^T\eta^k)^T B}{(A^T\eta^k)^T BA^T\eta^k} - \rho A^{-1}s^k(s^k)^T A^{-T}\right)A^T, \end{aligned}$$

where $\rho = 1/(\eta^k)^T s^k$ and a direct computation shows $s^k = At^k$, thus $\rho = 1/(A^T\eta^k)^T t^k$ and

$$B_f^{k+1} = A\left(B - \frac{BA^T\eta^k(A^T\eta^k)^T B}{(A^T\eta^k)^T BA^T\eta^k} - \frac{1}{(A^T\eta^k)^T t^k}t^k(t^k)^T\right)A^T$$

inspecting this equation, we see that it suffices to show $A^T\eta^k = \lambda^k$. By induction hypothesis we know $AB_g^k A^T = B_f^k$ which shows $Ay^{k+1} + b = x^{k+1}$. Using this information we see

$$A^T\eta^k = A^T\nabla f(Ay^{k+1} + b) - A^T\nabla f(Ay^k + b) = \nabla g(y^{k+1}) - \nabla g(y^k) = \lambda^k$$

implying the result.

- BFGS: We use the update formula for the inverse model hessians. Let $\eta^k = \nabla f(x^{k+1}) - \nabla f(x^k)$, $s^k = x^{k+1} - x^k$ and $\lambda^k = \nabla g(y^{k+1}) - \nabla g(y^k)$, $t^k = y^{k+1} - y^k$ and we set $B = B_g^k$. As above we have $A^T\eta^k = \lambda^k$, $s^k = At^k$. By the formula for BFGS and induction:

$$\begin{aligned} B_f^{k+1} &= ABA^T + \rho(s^k - ABA^T\eta^k)(s^k)^T \\ &\quad + \rho s^k(s^k - ABA^T\eta^k)^T - \rho^2(s^k - ABA^T\eta^k)^T\eta^k s^k(s^k)^T \\ &= A\Big(B + \rho(t^k - B\lambda^k)(t^k)^T \\ &\quad + \rho t^k(t^k - B\lambda^k)^T - \rho^2(t^k - B\lambda^k)^T\lambda^k t^k(t^k)^T\Big)A^T \end{aligned}$$

where we used that $\rho^2(t^k - B\lambda^k)^T\lambda^k A = A\rho^2(t^k - B\lambda^k)^T\lambda^k$ because $\rho^2(t^k - B\lambda^k)^T\lambda^k$ is just a number. And the term in the big blue brackets is just B_g^{k+1}.

□

Exercise 3

We begin with deriving the inverse DFP-Update formula:

$$\begin{aligned} \Psi_{DFP}(H, s, y) &= (\mathrm{Id} - \rho ys^T)H(\mathrm{Id} - \rho ys^T) + \rho yy^T = H - \rho ys^T H + \rho^2 ys^T Hsy^T - \rho Hsy^T + \rho yy^T \\ &= H + \rho[y, Hs]\begin{pmatrix} 1 + \rho s^T Hs & -1 \\ -1 & 0 \end{pmatrix}\begin{bmatrix} y^T \\ s^T H \end{bmatrix} \end{aligned}$$

So we can set: $A = H, U = \rho[y, Hs], C = \begin{pmatrix} 1 + \rho s^T Hs & -1 \\ -1 & 0 \end{pmatrix}$ and $V = \begin{bmatrix} y^T \\ s^T H \end{bmatrix}$ in the Sherman-

Morrison-Woodbury formula and (setting $B = H^{-1}$,) compute:

$$
\begin{aligned}
VA^{-1}U &= \begin{bmatrix} y^T \\ s^T H \end{bmatrix} B\rho[y, Hs] = \rho \begin{pmatrix} y^T By & y^T s \\ s^T y & s^T Hs \end{pmatrix}. \\
C^{-1} + VA^{-1}U &= \begin{pmatrix} 0 & -1 \\ -1 & -1-\rho s^T Hs \end{pmatrix} + \rho \begin{pmatrix} y^T By & y^T s \\ s^T y & s^T Hs \end{pmatrix} = \begin{pmatrix} \rho y^T By & \rho y^T s - 1 \\ \rho s^T y - 1 & -1 \end{pmatrix} \\
&= \begin{pmatrix} \rho y^T By & 0 \\ 0 & -1 \end{pmatrix} \\
(C^{-1} + VA^{-1}U)^{-1} &= \begin{pmatrix} \frac{1}{\rho y^T By} & 0 \\ 0 & -1 \end{pmatrix} \\
A^{-1}U(C^{-1} + VA^{-1}U)^{-1}VA^{-1} &= B\rho[y, Hs] \begin{pmatrix} \frac{1}{\rho y^T By} & 0 \\ 0 & -1 \end{pmatrix} \begin{bmatrix} y^T \\ s^T H \end{bmatrix} B = [\frac{By}{y^T By}, -\rho s] \begin{bmatrix} y^T B \\ s^T \end{bmatrix} = \frac{Byy^T B}{y^T By} - \rho ss^T
\end{aligned}
$$

Subtracting this expression from B gives the desired result.
Now for the inverse BFGS-Update formula:

$$
\Psi_{BFGS}(H, s, y) = H - \frac{Hss^T H}{s^T Hs} + \rho yy^T = H - \rho[Hs, y] \begin{pmatrix} \frac{1}{\rho s^T Hs} & 0 \\ 0 & -1 \end{pmatrix} \begin{bmatrix} s^T H \\ y^T \end{bmatrix}.
$$

So we can set: $A_{new} = H, U_{new} = -\rho[Hs, y], C_{new} = \begin{pmatrix} \frac{1}{\rho s^T Hs} & 0 \\ 0 & -1 \end{pmatrix}$ and $V_{new} = \begin{bmatrix} s^T H \\ y^T \end{bmatrix}$. One easily sees, that

$$
A_{new} + U_{new}C_{new}V_{new} = A^{-1} - A^{-1}U(C^{-1} + VA^{-1}U)^{-1}VA^{-1},
$$

up to the change of variables $s \leftrightarrow y$, $H \leftrightarrow B$, so one obtains the Inverse of $A_{new} + U_{new}C_{new}V_{new}$ by computing $A + UCV$ and substituting the respective variables, which is precisely the claim.

Exercise 4

From the B_{BFGS} update formula we can inductively derive the following:

$$
B^k_{\text{BFGS}} r = \prod_{\ell=k}^{0} (\operatorname{Id} - \rho^\ell s^\ell (y^\ell)^T) \cdot B^{(0)}_{\text{BFGS}} \cdot \prod_{\ell=0}^{k} (\operatorname{Id} - \rho^\ell y^\ell (s^\ell)^T) \cdot r + \sum_{\ell=0}^{k} \rho^\ell s^\ell (s^\ell)^T r \,.
$$

Lines 1-4 of Algorithm 5.53 compute the product $r_1 := \prod_{\ell=0}^{k} (\operatorname{Id} - \rho^\ell y^\ell (s^\ell)^T) \cdot r$: this is clear after calculating $\rho^\ell y^\ell (s^\ell)^T r = (\rho^\ell (s^\ell)^T r) y^\ell$ via Indexschlacht, where the expression in brackets is the scalar factor α^ℓ in the algorithm.

Line 5 then gives the product $r_2 := B^{(0)}_{\text{BFGS}} r_1$.

Then in lines 6-9 we calculate both the product $\prod_{\ell=k}^{0} (\operatorname{Id} - \rho^\ell s^\ell (y^\ell)^T) \cdot r_2$ and the sum $\sum_{\ell=0}^{k} \rho^\ell s^\ell (s^\ell)^T r$ at once: we have $\alpha^\ell s^\ell = (\rho^\ell (s^\ell)^T r) s^\ell = \rho^\ell s^\ell (s^\ell)^T \cdot r$ and the beta expression takes care of the product.