Assignment 1
Large-scale Optimization FTN0452

Li Ju

September 19, 2023

1 Problem 2.8

a).

Convexity

Proof. Given that f is m-strongly convex, by definition we have

$$\begin{aligned} f(y) - f(x) &\geq \frac{m}{2}\|y-x\|^2 + (y-x)^\top \nabla f(x) \\ &= \frac{m}{2}(y-x)^\top(y-x) + (y-x)^\top(\nabla f(x) - mx) + m(y-x)^\top x \\ &= \frac{m}{2}(y-x)^\top(y-x+2x) + (y-x)^\top(\nabla f(x) - mx) \\ &= \frac{m}{2}(\|y\|^2 - \|x\|^2) + (y-x)^\top(\nabla f(x) - mx) \\ f(y) - \frac{m}{2}\|y\|^2 &\geq f(x) - \frac{m}{2}\|x\|^2 + (y-x)^\top(\nabla f(x) - mx) \end{aligned}$$

With $q(x) \coloneqq f(x) - \frac{m}{2}\|x\|^2$ and $\nabla q(x) = \nabla f(x) - mx$, we have $q(y) \geq q(x) + (y-x)^\top \nabla q(x)$.

Let $z = (1-\alpha)x + \alpha y, \alpha \in [0,1]$, we have

$$\begin{aligned} q(x) &\geq q(z) + (x-z)^\top \nabla q(z) \\ &= q(z) + \alpha(x-y)^\top \nabla q(z) \qquad (1) \\ q(y) &\geq q(z) + (y-z)^\top \nabla q(z) \\ &= q(z) + (1-\alpha)(y-x)^\top \nabla q(z) \qquad (2) \end{aligned}$$

Summing up $(1-\alpha)\cdot(1) + \alpha \cdot (2)$, we have

$$\begin{aligned} (1-\alpha)q(x) + \alpha q(y) &\geq q(z) + (1-\alpha)\alpha(y-x)\nabla q(z) + (1-\alpha)\alpha(x-y)\nabla q(z) \\ &= q((1-\alpha)x + y) \end{aligned}$$

□

Smoothness

Proof. Given that f is m-strongly convex with L-continuous gradients, we have

$$
\begin{aligned}
f(y) &\le f(x) + (y-x)^\top \nabla f(x) + \frac{L}{2}\|y-x\|^2 \\
f(y) &\le f(x) + (y-x)^\top \nabla f(x) + \frac{L}{2}\|y-x\|^2 - \frac{m}{2}\|y-x\|^2 + \frac{m}{2}\|y-x\|^2 \\
f(y) - \frac{m}{2}\|y\|^2 &\le f(x) - \frac{m}{2}\|x\|^2 + (y-x)^\top(\nabla f(x) - mx) + \frac{L-m}{2}\|y-x\|^2
\end{aligned}
$$

With $q(x) \coloneqq f(x) - \frac{m}{2}\|x\|^2$ and $\nabla q(x) = \nabla f(x) + mx$, we have

$$
q(y) \le q(x) + (y-x)^\top \nabla q(x) + \frac{L-m}{2}\|y-x\|^2 \tag{3}
$$

Since we have proved the convexity of q, by Lemma 1, we have co-coercivity for q.

$$
\begin{aligned}
\frac{1}{L-m}\|\nabla q(y) - \nabla q(x)\|^2 &\le (y-x)^\top(\nabla q(y) - \nabla q(x)) \\
\|\nabla q(y) - \nabla q(x)\|^2 &\le (L-m)(y-x)^\top(\nabla q(y) - \nabla q(x)) \\
&\le (L-m)\|y-x\|^\top\|\nabla q(y) - \nabla q(x)\| \\
\|\nabla q(y) - \nabla q(x)\| &\le (L-m)\|y-x\|
\end{aligned}
$$

□

Lemma 1 (Co-coercivity)**.** *If convex function f is upper bounded by a quadratic function $f(y) \le f(x) - (y-x)^\top \nabla f(x) + \frac{L}{2}\|y-x\|^2$, we have*

$$
\frac{1}{L}\|\nabla f(y) - \nabla f(x)\|^2 \le (y-x)^\top(\nabla f(y) - \nabla f(x))
$$

Proof.

$$
f(y) - f(x) = f(y) - f(z) + f(z) - f(x) \le (y-z)^\top \nabla f(y) + (z-x)^\top \nabla f(x) + \frac{L}{2}\|z-x\|^2
$$

We want to have the tightest bound for $f(y) - f(x)$, so we take the minimal value of the right side w.r.t z:

$$
\begin{aligned}
\nabla f(x) + \nabla f(y) + L(z^\star - y) &= 0 \\
z^\star &= x + \frac{\nabla f(y) - \nabla f(x)}{L}
\end{aligned}
$$

By substituting $z = z^\star$, we have

$$
f(y) - f(x) \le (y-x)^\top \nabla f(y) - \frac{1}{2L}\|\nabla f(y) - \nabla f(x)\|^2 \tag{4}
$$

By letting $y = x$ and $x = y$, we have

$$
f(x) - f(y) \le (x-y)^\top \nabla f(x) - \frac{1}{2L}\|\nabla f(x) - \nabla f(y)\|^2
$$

Summing two inequalities up, we have

$$
\frac{1}{L}\|\nabla f(y) - \nabla f(x)\|^2 \le (y-x)^\top(\nabla f(y) - \nabla f(x))
$$

□

b).

By applying *co-cocervicity* to $q(x)$, which has $(L-m)$-continuous gradients, we have

$$
\begin{aligned}
[\nabla q(x) - \nabla q(y)]^\top (x-y) &\geq \frac{1}{L-m}\|\nabla q(x) - \nabla q(y)\|^2 \\
[\nabla f(x) - \nabla f(y)]^\top (x-y) &\geq \frac{1}{L-m}\|\nabla f(x) - \nabla f(y) - m(x-y)\|^2 + m\|x-y\|^2 \\
&= \frac{\|\nabla f(x) - \nabla f(y)\|^2 - 2m(\nabla f(x) - \nabla f(y))^\top (x-y) + mL\|x-y\|^2}{L-m} \\
&= \frac{(L-m)\|\nabla f(x) - \nabla f(y)\|^2 + 2m\|\nabla f(x) - \nabla f(y)\|^2}{(L-m)(L+m)} + \\
&\quad \frac{-2m(L+m)[\nabla f(x) - \nabla f(y)]^\top (x-y) + mL(L+m)\|x-y\|^2}{(L-m)(L+m)} \\
&\geq \frac{-2m(L+m)\|\nabla f(x) - \nabla f(y)\|\|x-y\| + mL(L+m)\|x-y\|^2}{(L-m)(L+m)} + \qquad (5) \\
&\quad \frac{2m\|\nabla f(x) - \nabla f(y)\|^2}{(L-m)(L+m)} + \frac{\|\nabla f(x) - \nabla f(y)\|^2}{L+m} \\
&\geq \frac{-2m^2(L+m)\|x-y\|^2 + mL(L+m)\|x-y\|^2}{(L-m)(L+m)} + \frac{2m^3\|x-y\|^2}{(L-m)(L+m)} + \qquad (6) \\
&\quad \frac{\|\nabla f(x) - \nabla f(y)\|^2}{L+m} \\
&= \frac{(mL^2 - m^2L)\|x-y\|^2}{(L-m)(L+m)} + \frac{\|\nabla f(x) - \nabla f(y)\|^2}{L+m} \\
&= \frac{mL}{m+L}\|x-y\|^2 + \frac{1}{m+L}\|\nabla f(x) - \nabla f(y)\|^2
\end{aligned}
$$

In (5) we utilize Cauchy-Schwarz inequality ($[\nabla f(x) - \nabla f(y)]^\top (x-y) \leq \|\nabla f(x) - \nabla f(y)\|\|x-y\|^2$). In (6) we use the m-strongly convexity of function $f(x)$ ($\|\nabla f(x) - \nabla f(y)\| \geq m\|x-y\|$), from which $\|\nabla f(x) - \nabla f(y)\|^2 \geq m^2\|x-y\|^2$ is also derived and used.

2 Problem 3.5

Proof. For iteration t with step size η, we have

$$
\begin{aligned}
\|x^{t+1} - x^\star\|^2 &= \|x^{t+1} - x^t + x^t - x^\star\|^2 \\
&= \|x^t - x^\star\|^2 + \eta^2\|\nabla f(x^t)\|^2 - 2\eta(x^t - x^\star)^\top \nabla f(x^t)
\end{aligned}
$$

With Theorem 2.1.12 from Nesterov [2003], we have

$$
(x^t - x^\star)^\top \nabla f(x^t) \leq \frac{mL}{L+m}\|x^t - x^\star\|^2 + \frac{1}{L+m}\|\nabla f(x^{t+1})\|^2
$$

Hence,

$$
\|x^{t+1} - x^\star\|^2 \leq (1 - \frac{2\eta mL}{m+L})\|x^t - x^\star\|^2 + \eta(\eta - \frac{2}{m+L})\|\nabla f(x^t)\|^2
$$

With $\eta = \frac{2}{m+L}$, we have

$$
\|x^{t+1} - x^\star\| \leq \frac{L-m}{L+m}\|x^t - x^\star\|
$$

After k iteration with $\kappa \coloneqq \frac{L}{m}$, we have

$$\|x^k - x^\star\| \leq (\frac{\kappa - 1}{\kappa + 1})^k \|x^0 - x^\star\|$$

□

3 Problem 3.7

a). The solution space of $Ax = b$ is a subspace with dimension of $(N - n)$ in R^N.

b). With $f(x) = \frac{1}{N}\|Ax - b\|^2$, we have $\nabla f(x) = \frac{2}{N}A^\top(Ax - b)$. Lipschitz constant L of $\nabla f(x)$ is the least value that enables $\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|$. Thus we have

$$\begin{aligned} L &\geq \frac{\|\nabla f(x) - \nabla f(y)\|}{\|x - y\|} \\ &= \frac{2}{N}\frac{\|A^\top A(x - y)\|}{\|x - y\|} \\ &\geq \frac{2}{N}\frac{\lambda_{\max}(A^\top A)\|x - y\|}{\|x - y\|} \\ &= \frac{2}{N}\lambda_{\max}(A^\top A) \end{aligned}$$

where $\lambda_{\max}(A^\top A)$ denotes the largest eigenvalue of $A^\top A$.

c). Considering that Hessian of $f(x)$ is $A^\top A$, which has nonnegative eigenvalues, we recognize that $f(x)$ is Lipschitz smooth and weakly convex. For weakly-convex and Lipschitz smooth function, gradient descent has convergence rate of $\mathcal{O}(1/T)$, which implies that $\mathcal{O}(1/\epsilon)$ iterations are required for the convergence.

d). The minimizer x_μ for the regularized formulation $f_\mu(x)$ is derived as follows:

$$\begin{aligned} &\nabla f_\mu(x_\mu) = \frac{2}{N}A^\top(Ax_\mu - b) + 2\mu x_\mu = 0 \\ &(\frac{2}{N}A^\top A + 2\mu I)x_\mu = \frac{2}{N}A^\top b \\ &x_\mu = (A^\top A + N\mu I)^{-1}A^\top b \end{aligned}$$

e). It is recognized that $f_\mu(x)$ is 2μ-strongly convex, which has a strictly positive definite Hessian $(2A^\top A + 2\mu I)$. For strongly convex and Lipschitz smooth function, gradient descent has a convergence rate of $\mathcal{O}(\log(1/T))$, indicating that $\mathcal{O}(\exp(1/\epsilon))$ iterations enable error tolerance of ϵ.

f). We have

$$\begin{aligned} f_\mu(\hat{x}) - f_\mu(x_\mu) = \frac{1}{N}\|A\hat{x} - b\|^2 - \frac{1}{N}\|Ax_\mu - b\|^2 + \mu\|\hat{x}\|^2 - \mu\|x_\mu\|^2 \leq \epsilon \\ f(\hat{x}) \leq \frac{1}{N}\|Ax_\mu - b\|^2 + \mu\|x_\mu\|^2 - \mu\|\hat{x}\|^2 + \epsilon \\ \leq f_\mu(\hat{x}) - \mu\|\hat{x}\|^2 + \epsilon \end{aligned}$$

g). For m, we have

$$m \leq \frac{\|\nabla f(x)\|^2}{2(f(x) - f(x^\star))}$$

Since we know that there exist $x^\star$ that makes $Ax^\star = b$, then we have

$$\begin{aligned} m &\leq \frac{\|\nabla f(x)\|^2}{2f(x)} \\ &= \frac{4/N^2(A^\top(Ax-b))^\top(A^\top(Ax-b))}{2/N\|Ax-b\|^2} \\ &= \frac{2N(Ax-b)^\top AA^\top(Ax-b)}{(Ax-b)^\top(Ax-b)} \\ &\leq 2N\lambda_{\min+}(A^\top A) \end{aligned}$$

where $\lambda_{\min+}(A^\top A)$ denotes the smallest nonzero eigenvalue of $A^\top A$.

h). Let $\lambda_{\max}$ and $\lambda_{\min+}$ be the largest and smallest nonzero eigenvalues of $A^\top A$, respectively. We choose $\eta = \frac{1}{L} = \frac{1}{\lambda_{\max}}$. Then at iteration t, we have

$$f(x^t) - f(x^\star) \leq (1 - \frac{\lambda_{\min+}}{\lambda_{\max}})^t (f(x^0) - f(x^\star))$$

With a given error tolerance ϵ, we have number of required iterations $T = \log(\frac{\epsilon}{f(x^0)-f(x^\star)}) / \log(\frac{\lambda_{\max}-\lambda_{\min+}}{\lambda_{\max}})$.

4 Problem 4.2

All methods have been implemented in Python with `numpy`, with codes deferred to Appendix A.1.

To estimate numbers of iterations required for the target tolerance $\epsilon \leq 10^{-6}$, we run all methods with 10 different random starts on the same Hessian matrix generated. The average and standard deviation of numbers of iterations are reported in Table 1.

Training curves of all methods are plotted as shown in Figure 1. All methods are run from a same start with a fixed random seed.

Discussion All experiments are consistent with theoretical convergence analysis.

Taking the most conservative step size from global smoothness, steepest descent with $\eta = \frac{1}{L}$ has the slowest convergence with ~ 400 iterations to reach the tolerance. Taking advantages of strong convexity, steepest descent with step size $\eta = \frac{2}{L+m}$ has faster convergence, reaching the tolerance with ~ 300 iterations. The convergence is faster if adaptive step size for each iteration is applied, by using exact line-search to find the optimal step size along the $-\nabla f(x)$ and ~ 200 iterations are required.

However, the descent direction $-\nabla f(x)$ could lead to an oscillation of the sequence $\{x^t\}_{t=0}^T$ if scales of coordinates differ. Applying momentum to update rule is able to accelerate the optimization. Intuitive heavy ball momentum does significantly reduced the number of iterations to reaching the tolerance to ~ 85. With Nesterov momentum, which has been proved to have an optimal convergence rate, even fewer iterations (approximately 60) are required.

Method	Mean	Std. Dev.
SD: $\eta = \frac{2}{L+m}$	297.20	28.34
SD: $\eta = \frac{1}{L}$	389.90	56.15
SD: Line Search	197.70	28.16
Heavy Ball Momen.	84.30	1.68
Nesterov Momen.	58.30	3.90

Table 1: A comparison of numbers of iterations required for the expected error ϵ

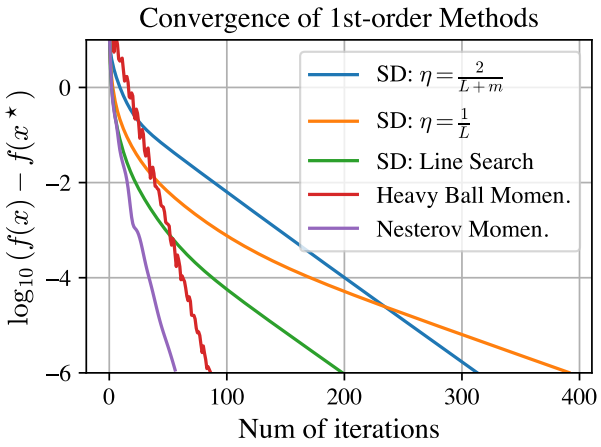

Figure 1: Training curves of first-order optimization methods in a strongly convex problem

5 Problem 4.3

Figure 2 shows the experimental results after we set $m = 0.0$ in the implementation (making function $f(x)$ to be weakly convex).

For steepest descent with $\eta = \frac{1}{L}$, due to the conservative step size, the performance remains almost the same as it was in the strongly convex setting. For steepest descent with $\eta = \frac{2}{L+m}$, the method stops at a point with much greater functions values than the optimum. This is consistent with theoretical analysis in Section 2. We have shown that at iteration t, we have the progress

$$\|x^{t+1} - x^\star\| \leq \frac{L-m}{L+m}\|x^t - x^\star\|$$

However, when $m = 0$, we may have $\|x^{t+1} - x^\star\| = \|x^t - x^\star\|$, which does not guarantee convergence.

For Steepest descent with exact line search for step size, the performance remains good.

For momentum-accelerating methods, if we do not adjust hyperparameters, the methods may diverge. This is due to the hyperparameters here in the experiments rely on m, leading $\eta = 4/L$ and $\beta = 1$ for heavy ball momentum steepest descent and $\eta = 1/L$ and $\beta = 1$ for Nesterov momentum steepest descent. After setting $\eta = 1/L$ and $\beta = 0.9$ for both methods, both methods achieve reasonable performance, as shown in Figure 2.

6 Proble CG

Conjugate gradient descent is implemented in Python and the code is deferred to Appendix A.2. It has been verified that after 41 iterations, the method has converged with $\|Lx - b\| = 5.59 \times 10^{-7} \leq 10^{-6}$. Also

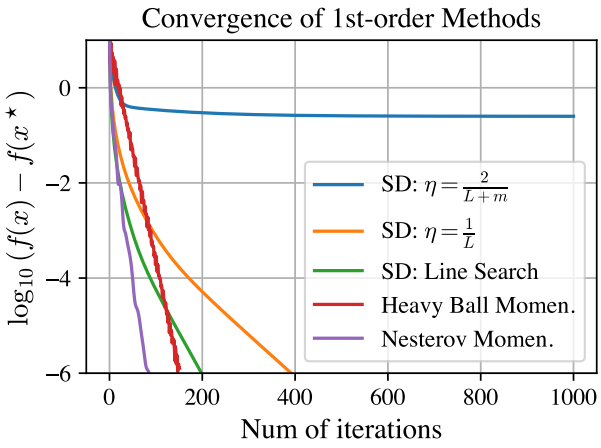

Figure 2: Training curves of first-order optimization methods in a weakly convex problem

$\log_{10}\|Lx - b\|$ is plotted against the number of iterations as shown in Figure 3. It is observed that with a large sparse matrix L of 10000×10000, the algorithm is able to converge in ~ 40 iterations.

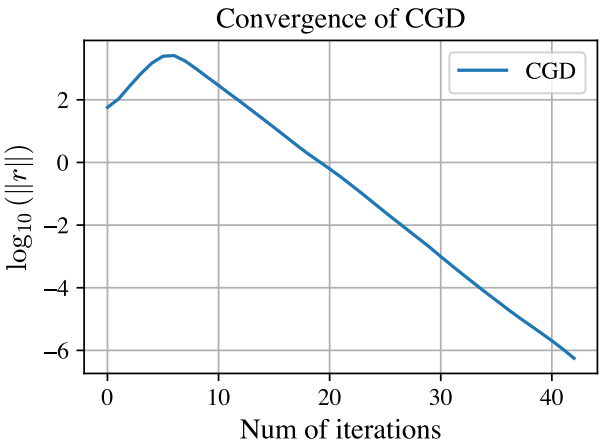

Figure 3: Convergence of Conjugate Gradient Descent

References

Yurii Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2003.

A Code

A.1 Problem 4.2

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


# script to compare different optimization methods
# for finding the minimal for £\frac{1}{2} x^T A x£

n = 100
m = 0.01
L = 1
epsilon = 1e-6
kmax = 1000


# generate matrix A with eigenvalues in [m, L]
def get_A_matrix(n=100, m=0.01, L=1):
    n = 100
    np.random.seed(42)
    A = np.random.randn(n, n)
    (Q, R) = np.linalg.qr(A)

    D = np.random.rand(n)
    D = 10.**D
    Dmin = D.min()
    Dmax = D.max()
    D = (D-Dmin)/(Dmax-Dmin)
    D = m + D*(L-m)
    A = Q.T@np.diag(D)@Q
    return A


def get_grad(A, x):
    grad = A@x.T
    return grad


def get_value(A, x):
    value = 1/2*x@A@x.T
    return value


def steepest_descent_fixed_stepsize(A, x0, eta, epsilon, tmax):
    x = x0
    t = 0

    fxs = [get_value(A, x)]
    while t < tmax:
        grad = get_grad(A, x)
        x = x - eta*grad
        fx = get_value(A, x)
```

```
        if fx < epsilon:
            break
        # print(f"t={t} with f(x)={fx}")
        t += 1
        fxs.append(fx)
    return np.array(fxs)


def steepest_descent_line_search_stepsize(A, x0, epsilon, tmax):
    x = x0
    t = 0
    fxs = [get_value(A, x)]
    while t < tmax:
        grad = get_grad(A, x)
        eta = np.linalg.norm(grad)**2/(grad.T@A@grad)
        x = x - eta*grad
        fx = get_value(A, x)
        if fx < epsilon:
            break
        # print(f"t={t} with f(x)={fx}")
        t += 1
        fxs.append(fx)
    return np.array(fxs)


def heavy_ball(A, x0, eta, beta, epsilon, tmax):
    x_prev = x0
    x = x0
    t = 0
    fxs = [get_value(A, x)]
    while t < tmax:
        grad = get_grad(A, x)
        x_next = x - eta*grad + beta*(x - x_prev)
        x_prev = x
        x = x_next

        fx = get_value(A, x)
        if fx < epsilon:
            break
        # print(f"t={t} with f(x)={fx}")
        t += 1
        fxs.append(fx)
    return np.array(fxs)


def nesterov(A, x0, eta, beta, epsilon, tmax):
    x = x0
    last_update = 0
    t = 0
    fxs = [get_value(A, x)]
    while t < tmax:
        tmpx = x + beta*last_update
        grad = get_grad(A, tmpx)
        x_next = x - eta*grad + beta*last_update
```

```
        last_update = x_next - x
        x = x_next

        fx = get_value(A, x)
        if fx < epsilon:
            break
        # print(f"t={t} with f(x)={fx}")
        t += 1
        fxs.append(fx)
    return np.array(fxs)


# using the same matrix A for all runs but different x0
def one_run(seed=42):
    A = get_A_matrix()

    np.random.seed(seed)
    x0 = np.random.randn(n)

    results = {
        r"SD: $\eta=\frac{2}{L+m}$":
            steepest_descent_fixed_stepsize(A, x0, 2/(L+m), epsilon, kmax),
        r"SD: $\eta=\frac{1}{L}$":
            steepest_descent_fixed_stepsize(A, x0, 1/L, epsilon, kmax),
        "SD: Line Search":
            steepest_descent_line_search_stepsize(A, x0, epsilon, kmax),
        "Heavy Ball Momen.":
            heavy_ball(A, x0, 4/(L+m+2*np.sqrt(m*L)),
                       (np.sqrt(L)-np.sqrt(m))/(np.sqrt(L)+np.sqrt(m)),
                       epsilon, kmax),
        "Nesterov Momen.":
            nesterov(A, x0, 1/L,
                     (np.sqrt(L)-np.sqrt(m))/(np.sqrt(L)+np.sqrt(m)),
                     epsilon, kmax)
    }
    return results


# generate the result table with 10 different runs
num_iterations = {}
random_seeds = np.random.randint(0, 1000, 10)
for seed in random_seeds:
    results = one_run(seed)
    for key, value in results.items():
        if key not in num_iterations:
            num_iterations[key] = []
        num_iterations[key].append(len(value))

# formulate the table and print it
print("Method & Mean & Std. Dev. \\\\")
for key, value in num_iterations.items():
    print(f"{key} & {np.mean(value):.2f} & {np.std(value):.2f} \\\\")
print("")
print("")  # empty line
```

```
results = one_run()
# plot results with log10 scale
for key, value in results.items():
    plt.plot(np.log10(value), label=key)


plt.xlabel("Num of iterations")
plt.ylabel(r"$\log_{10}(f(x) - f(x^\star)$")
plt.title("Convergence of 1st-order Methods")
plt.ylim(-6, 1)

plt.legend()
plt.savefig("prob42.pdf", dpi=300)
```

A.2 Problem CG

```
import numpy as np
import scipy.sparse as sp
import matplotlib.pylab as plt


# The problem we are solving is
# £\arg\min_{x} 1/2 x^\top L x - b^\top x£
# Note that £L£ is symmetric and positive semi-definite.


def get_problem():
    np.random.seed(42)
    n_nodes = 10000
    n_deg = 20
    # compute matrix density
    density = n_deg/n_nodes
    # Right hand side vector (uniform)
    b = np.random.rand(n_nodes)
    # Construct matrix
    # conductance uniform [0,1]
    L = -np.abs(sp.rand(n_nodes+1, n_nodes+1, density=density,
                        format="coo", random_state=42))
    # make L symmetric
    L = sp.triu(L) + sp.triu(L, 1).T
    # Finalize diagonal
    L = L - sp.diags(L@np.ones(n_nodes+1), format="coo")
    # remove last row and column
    L = L[:-1, :-1]
    return L, b


def get_residual(L, x, b):
    return L@x - b


def conjugate_gradient_descent(L, b, x0, eps, tmax):
    # Initialize
```

```
    x = x0
    r = get_residual(L, x, b)
    p = -r

    residuals = [np.sqrt(r.T@r)]
    # Iterate
    t = 0
    while t < tmax:
        alpha = (r.T@r)/(p.T@L@p)
        x = x + alpha*p
        r_new = r + alpha*L@p
        beta = (r_new.T@r_new)/(r.T@r)
        p = -r_new + beta*p
        r = r_new

        residual = np.sqrt(r.T@r)
        residuals.append(residual)
        if residual < eps:
            break
        t += 1
    print(f"Residual (||Lx - b||): {residual}")
    return np.array(residuals)


results = conjugate_gradient_descent(
    *get_problem(), np.zeros(10000), 1e-6, 10000)

# plot results
plt.plot(np.log10(results), label="CGD")

plt.xlabel("Num of iterations")
plt.ylabel(r"$\log_{10}(\| r \|)$")
plt.title("Convergence of CGD")
# plt.ylim(-6, 1)

plt.legend()
plt.savefig("probcg.pdf", dpi=300)
```