Dhamma Kimpara and Sam Zhang

Problem 1

i) Yes, the cost is strongly convex. Write $f_t(x) = \frac{1}{2}\|Ax - b_t\|_2^2$, and $h_t(x) = Ax - b_t$. Then f_t is μ-strongly convex if the smallest eigenvalue of its Hessian $\nabla^2 f_t(x) = A$ is positive, with μ equal to that eigenvalue. By construction, then, $\mu = 1/\sqrt{\kappa}$.

ii) We know from class that for convergence, we require the step size α to be in the range $(0, 2/L]$. Since L is the Lipschitz constant of $\nabla f = Ax - b_t$, we have that since

$$\|Ax - b_t - (Ay - b_t)\| = \|Ax - Ay\| \leq \|A\| \|x - y\|,$$

that $L \leq \|A\| = 1$, since the largest singular value of A is 1. Thus we must pick $\alpha \in (0, 2]$. The bound we derived in class uses

$$\rho = \max\{|1 - \alpha\mu|, |1 - \alpha L|\} = \max\{\left|1 - \alpha/\sqrt{\kappa}\right|, |1 - \alpha|\}.$$

This gives the finite bound

$$\|x_t - x_t^*\| \leq \rho^t \|x_0 - x_0^*\| + \sigma \sum_{i=0}^{t} \rho^i = \rho^t \|x_0 - x_0^*\| + \frac{1 - \rho^t}{1 - \rho}$$

with the asymptotic result

$$\limsup_{t\to\infty} \|x_t - x_t^*\| \leq \frac{1}{1 - \rho}.$$

With this specific problem, since $\kappa = 100$, we have that for an arbitrary pick of $\alpha = 1$ that $\rho = 99/100$, so the asymptotic bound is $1/(1 - \rho) = 100$. As seen in Figure 1, the tracking error does not appear to typically approach either the finite or asymptotic bounds here.

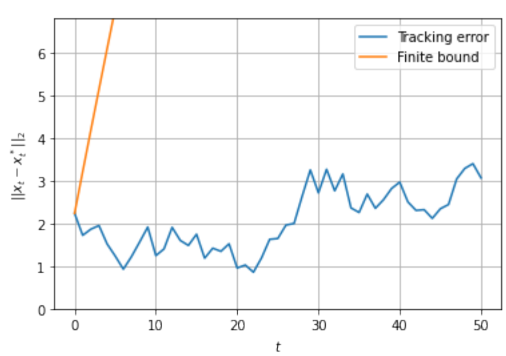
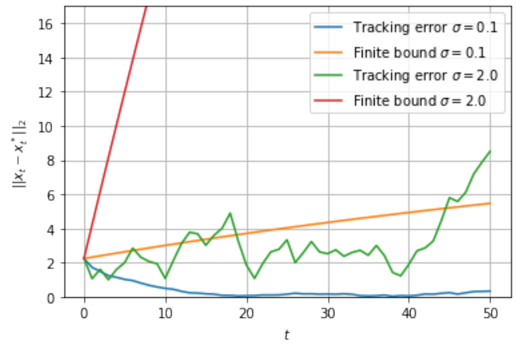


Figure 1: Tracking error of one run of online gradient descent vs. finite bounds



iii)

Figure 2: Tracking error of one run of online gradient descent vs. finite bounds for $\sigma = 0.1$ and $\sigma = 2.0$.

Code

Generator

```
import numpy as np
from numpy import linalg

from scipy.stats import ortho_group
# Return a random orthogonal matrix, drawn from the O(N) Haar distribution (t


def generate_A(n, d):
    U = ortho_group.rvs(n)
    V = ortho_group.rvs(d)
    D = np.diagflat(np.flip(np.linspace(1 / np.sqrt(100), 1, min(n, d))))
    D = np.vstack((D, np.zeros([n - d, d])))
    return U @ D @ V


def generate_bt(A, n, x_gen):
    '''generator for b_t'''
    while True:
        xstar = next(x_gen)
        w = np.random.normal(0, 10 ** (-3), n)
        yield A @ xstar + w, xstar


def xstar1(sigma, d):
    '''generator for x_t^* for q1'''
    x = np.zeros(d)
    while True:
        yield x
        x += sigma * sample_n_sphere_surface(d)
```

```
def xstar2(d):
    '''generator for x_t^* for question 2'''
    x = np.zeros(d)
    while True:
        yield x
        x = xstar2_helper(x, d)


def xstar2_helper(x, d):
    step = sample_n_sphere_surface(d)  # step size 1
    while linalg.norm(x + step) >= 1.0:  # resample the step
        step = sample_n_sphere_surface(d)
    return x + step


def sample_n_sphere_surface(ndim, norm_p=2):
    """sample random vector from S^n-1 with norm_p"""

    vec = np.random.randn(ndim)
    vec = vec / linalg.norm(vec, norm_p)  # create random vector with norm 1
    return vec
```

Optimizer

```
import numpy as np
from numpy import linalg
import generator as gen


def gradient_descent_experiment(A, alpha, n, d, sigma, iters=100, projected=F
    tracking_error = []

    if projected:  # pgd with sigma=1
        xstar_gen = gen.xstar2(d)
    else:  # gd with variable sigma
        xstar_gen = gen.xstar1(sigma, d)

    b_gen = gen.generate_bt(A, n, xstar_gen)
    x_t = np.ones(d)
    b_t, xstar_t = next(b_gen)

    tracking_error.append(linalg.norm(x_t - xstar_t))

    for i in range(iters):
        b_t, xstar_t = next(b_gen)
        x_t = x_t - alpha * gradient(A, x_t, b_t)
```

```python
        if projected:  # do projection step in pgd
            x_t = project_unit_ball(x_t)
        tracking_error.append(linalg.norm(x_t - xstar_t))

    return tracking_error


def gradient(A, x_t, b_t):
    return A.T @ (A @ x_t - b_t)


def project_unit_ball(x):
    norm = linalg.norm(x)
    if norm > 1.0:
        return x / norm
    else:
        return x
```

Experiment