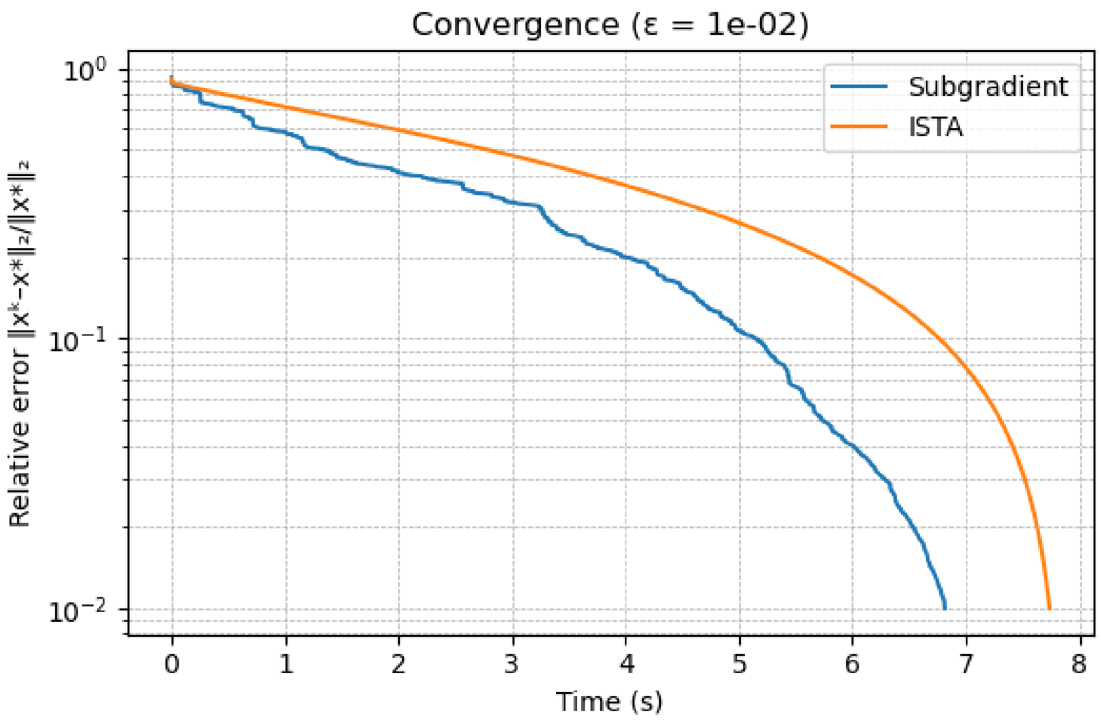
hw7

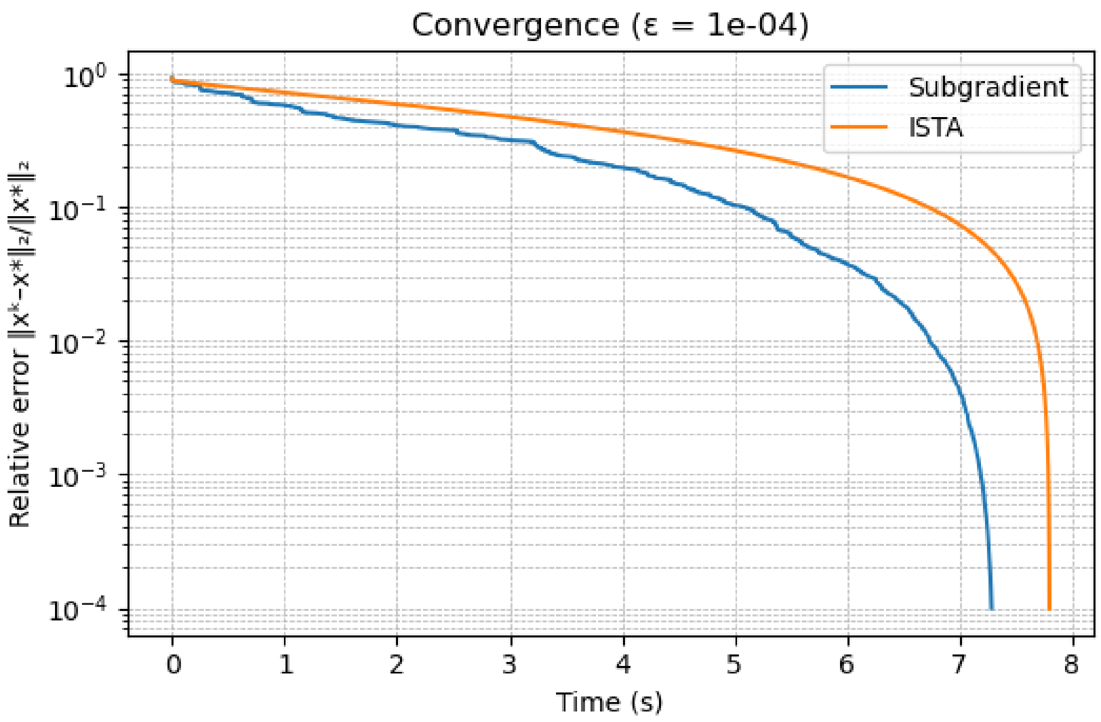
May 27, 2025

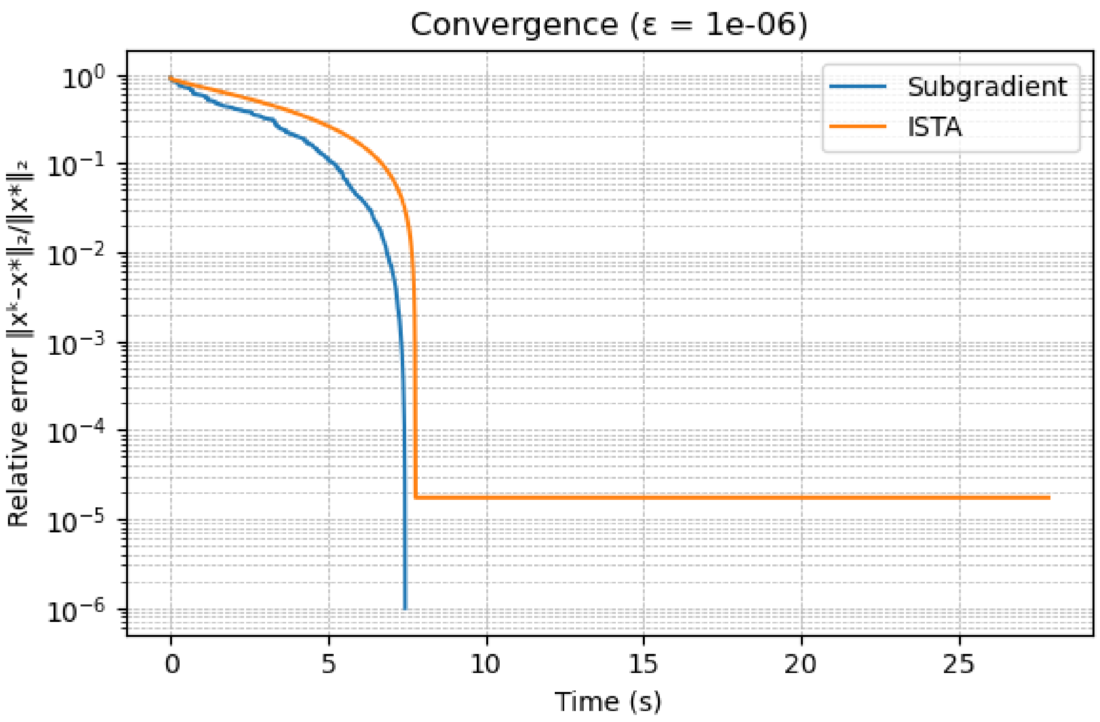
[78]:

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

#    1) Data generation
m, n, s = 100, 500, 5
np.random.seed(0)
A = np.random.randn(m, n)
x_star = np.zeros(n)
p = np.random.permutation(n)
x_star[p[:s]] = np.random.randn(s)
b = A.dot(x_star)

# True objective at x*, used for f-diff
def objective(x,  ):
    return np.linalg.norm(A.dot(x) - b)**2 +   * np.linalg.norm(x, 1)

# Soft-thresholding operator
def soft_threshold(x, lam):
    return np.sign(x) * np.maximum(np.abs(x) - lam, 0.0)

def subgradient_optimal(A, b,  , x0, eps, max_iter=1000000):
    x = x0.copy()
    f_star = objective(x_star,  )
    errors, f_diffs = [], []
    t_start = time.time()

    for k in range(1, max_iter + 1):
        grad_g = 2 * A.T.dot(A.dot(x) - b)
        subgrad_h =   * np.sign(x)
        gk = grad_g + subgrad_h

        g_norm_sq = np.linalg.norm(gk)**2
        if g_norm_sq == 0:
            break  # Already optimal

        t = (objective(x,  ) - f_star) / g_norm_sq
        x -= t * gk

        err = np.linalg.norm(x - x_star) / np.linalg.norm(x_star)
        errors.append(err)
        f_diffs.append(objective(x,  ) - f_star)
        if err < eps:
            break

    return x, errors, f_diffs, time.time() - t_start

def ista_lasso(A, b,  , x0, eps, max_iter=1000000):
    x = x0.copy()
    f_star = objective(x_star,  )
    L      = 2 * np.linalg.norm(A, 2)**2
    t      = 1.0 / L
    errors, f_diffs = [], []
    t_start = time.time()
    for k in range(1, max_iter+1):
        grad_g = 2 * A.T.dot(A.dot(x) - b)
        x      = soft_threshold(x - t * grad_g, t *  )

        err      = np.linalg.norm(x - x_star) / np.linalg.norm(x_star)
        errors.append(err)
        f_diffs.append(objective(x,  ) - f_star)
        if err < eps:
            break

    return x, errors, f_diffs, time.time() - t_start

#    3) Run experiments
 = 0.0001
eps_list = [1e-2, 1e-4, 1e-6]

results = {}
for eps in eps_list:
    x0 = np.zeros(n)
    x_sub, err_sub, f_sub, t_sub   = subgradient_optimal(A, b, 0.0001, x0, eps)
    x_ist, err_ist, f_ist, t_ist   = ista_lasso(A, b, 0.005, x0, eps)
    results[eps] = {
        'time':   {'sub': t_sub,    'ista': t_ist},
        'f_diff': {'sub': f_sub[-1], 'ista': f_ist[-1]},
        'iters':  {'sub': len(err_sub),'ista': len(err_ist)},
        'errors': {'sub': err_sub,   'ista': err_ist},
        'f_traj': {'sub': f_sub,     'ista': f_ist}
    }

#    4) Bar-charts at termination
```

```
labels     = [f"{e:.0e}" for e in eps_list]
xpos       = np.arange(len(eps_list))
width      = 0.35

sub_times  = [results[e]['time']['sub']   for e in eps_list]
ista_times = [results[e]['time']['ista'] for e in eps_list]
sub_fdiff  = [results[e]['f_diff']['sub']   for e in eps_list]
ista_fdiff = [results[e]['f_diff']['ista'] for e in eps_list]
#    5) Convergence plots
for eps in eps_list:
    err_sub = results[eps]['errors']['sub']
    err_ist = results[eps]['errors']['ista']
    time_sub = np.cumsum([results[eps]['time']['sub'] / len(err_sub)] *␣
 ↪len(err_sub))
    time_ist = np.cumsum([results[eps]['time']['ista'] / len(err_ist)] *␣
 ↪len(err_ist))

    plt.figure(figsize=(6, 4))
    plt.semilogy(time_sub, err_sub, label='Subgradient')
    plt.semilogy(time_ist, err_ist, label='ISTA')
    plt.xlabel('Time (s)')
    plt.ylabel('Relative error ‖x -x*‖ /‖x*‖ ')
    plt.title(f'Convergence (  = {eps:.0e})')
    plt.legend()
    plt.grid(True, which='both', ls='--', lw=0.5)
    plt.tight_layout()
    plt.show()
```


Convergence (ε = 1e-02)
Subgradient
ISTA
Relative error ‖xᵏ−x*‖₂/‖x*‖₂
Time (s)
10⁰
10⁻¹
10⁻²
0
1
2
3
4
5
6
7
8


Convergence (ε = 1e-04)
Subgradient
ISTA
Relative error ‖xᵏ−x*‖₂/‖x*‖₂
Time (s)
10⁰
10⁻¹
10⁻²
10⁻³
10⁻⁴
0
1
2
3
4
5
6
7
8


Convergence (ε = 1e-06)
Subgradient
ISTA
Relative error ‖x^k−x*‖₂/‖x*‖₂
Time (s)
10^0
10^-1
10^-2
10^-3
10^-4
10^-5
10^-6
0
5
10
15
20
25