

## Analysis of Frank-Wolfe and Pairwise Frank-Wolfe Algorithms on the LASSO problem


### Optimization for Data Science (2023/2024)

Roya Ghamari

Student ID: 2071969

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

```
from google.colab import drive
drive.mount('/content/drive')
```

 Mounted at /content/drive

### ✓ FW & Pairwise FW:

```
TAU = 1 # parameter that controls the amount of shrinkage
K = 1000 # number of iterations based on FW paper.
THRESHOLD = 10e-8 # threshold for duality gap, our criterion for approximation quality.
```

```
def initialize_x(A):
    n = A.shape[1]
    x = np.zeros(n)
    x[0] = TAU
    return x

def f(x, A, b):
    residuals = np.dot(A, x) - b
    return np.sum(residuals ** 2)

def gradient(x, A, b):
    grad = 2 * np.dot(A.T, np.dot(A, x) - b)
    return grad

def linear_minimization_oracle(grad, tau):
    idx = np.argmax(np.abs(grad)) # i_k in survey paper
    e = np.zeros_like(grad)
    e[idx] = 1
    s = np.sign(-grad[idx]) * tau * e # atom wich we should go towards to
    return s

def duality_gap(d, grad):
    gap = - np.dot(grad.T, d).item()
    return gap

def armijo_rule(x_initial, A, b, d, grad, max_step_size):
    delta = 0.9
    gamma = 0.2
    step_size = max_step_size
    fx_initial = f(x=x_initial, A=A, b=b)
    x = x_initial.copy()
    fx = fx_initial.copy()
    counter = 0
    while fx >= fx_initial + gamma * step_size * np.dot(grad.T, d):
        step_size = step_size * delta
        x = x_initial + step_size * d
        fx = f(x=x, A=A, b=b)
        counter += 1
        if counter > 1000:
            break
    return step_size

def pairwise_direction(x, grad, tau):
    atoms_idx = np.where(np.abs(x - 0) > 10e-8)[0] # get index of atomes that are not zero
    idx = np.argmax(np.abs(grad[atoms_idx])) # find index of max gradient
    idx = atoms_idx[idx]
    e = np.zeros_like(grad)
    e[idx] = 1
```

```

v = np.sign(grad[idx]) * tau * e # atom which we should get away from
step_size_max = np.abs(x[idx])

return v, step_size_max

```

```

def frank_wolfe(x, A, b, K, threshold, tau):
    time_values = []
    fx_values = []
    gap_values = []
    start_time = time.time()
    for _ in range(K):
        fx = f(x=x, A=A, b=b)
        grad = gradient(x=x, A=A, b=b)
        s = linear_minimization_oracle(grad=grad, tau= tau)
        d = s - x
        gap = duality_gap(d=d, grad=grad)
        if gap <= threshold:
            end_time = time.time()
            time_values.append(end_time - start_time)
            fx_values.append(fx)
            gap_values.append(gap)
            print(f"gap = }\n{threshold = }")
            break

        step_size = armijo_rule(x_initial=x, A=A, b=b, d=d, grad=grad, max_step_size=1)
        x = x + step_size * d
        fx = f(x=x, A=A, b=b)

        end_time = time.time()
        time_values.append(end_time - start_time)
        fx_values.append(fx)
        gap_values.append(gap)

    return x, time_values, fx_values, gap_values

def pairwise_frank_wolfe(x, A, b, K, threshold, tau):
    time_values = []
    fx_values = []
    gap_values = []
    start_time = time.time()
    for _ in range(K):
        fx = f(x=x, A=A, b=b)
        grad = gradient(x=x, A=A, b=b)
        s = linear_minimization_oracle(grad, tau)
        v, step_size_max = pairwise_direction(x=x, grad=grad, tau=tau)
        d = s - v
        gap = duality_gap(d=d, grad=grad)
        if gap <= threshold:
            end_time = time.time()
            time_values.append(end_time - start_time)
            fx_values.append(fx)
            gap_values.append(gap)
            print(f"gap = }\n{threshold = }")
            break

        step_size = armijo_rule(x_initial=x, A=A, b=b, d=d, grad=grad, max_step_size=step_size_max)
        x = x + step_size * d
        fx = f(x=x, A=A, b=b)

        end_time = time.time()
        time_values.append(end_time - start_time)
        fx_values.append(fx)
        gap_values.append(gap)

    return x, time_values, fx_values, gap_values

```

```

def plot_graphs(gap_values, fx_values, time_values):
    fig, axs = plt.subplots(2, 2, figsize=(8, 8))

    axs[0, 0].plot(np.log10(gap_values))
    axs[0, 0].set_xlabel("# iterations")
    axs[0, 0].set_ylabel("log10(Gap)")
    axs[0, 0].set_title("log10(Gap) vs # iterations")

    axs[0, 1].plot(np.log10(fx_values))
    axs[0, 1].set_xlabel("# iterations")
    axs[0, 1].set_ylabel("log10(f(x))")
    axs[0, 1].set_title("log10(f(x)) vs # iterations")

    axs[1, 0].plot(time_values, np.log10(gap_values))
    axs[1, 0].set_xlabel("CPU Time")
    axs[1, 0].set_ylabel("log10(Gap)")
    axs[1, 0].set_title("log10(Gap) vs CPU Time")

    axs[1, 1].plot(time_values, np.log10(fx_values))
    axs[1, 1].set_xlabel("CPU Time")
    axs[1, 1].set_ylabel("log10(f(x))")
    axs[1, 1].set_title("log10(f(x)) vs CPU Time")

    plt.tight_layout()
    plt.show()

def plot_all(gap_values_fw, fx_values_fw, time_values_fw, gap_values_pw, fx_values_pw, time_values_pw):
    fig, axs = plt.subplots(2, 2, figsize=(8, 8))

    axs[0, 0].plot(np.log10(gap_values_fw), label="FW")
    axs[0, 0].plot(np.log10(gap_values_pw), label="PW")
    axs[0, 0].set_xlabel("# iterations")
    axs[0, 0].set_ylabel("log10(Gap)")
    axs[0, 0].set_title("log10(Gap) vs # iterations")
    axs[0, 0].legend()

    axs[0, 1].plot(np.log10(fx_values_fw), label="FW")
    axs[0, 1].plot(np.log10(fx_values_pw), label="PW")
    axs[0, 1].set_xlabel("# iterations")
    axs[0, 1].set_ylabel("log10(f(x))")
    axs[0, 1].set_title("log10(f(x)) vs # iterations")
    axs[0, 1].legend()

    axs[1, 0].plot(time_values_fw, np.log10(gap_values_fw), label="FW")
    axs[1, 0].plot(time_values_pw, np.log10(gap_values_pw), label="PW")
    axs[1, 0].set_xlabel("CPU Time")
    axs[1, 0].set_ylabel("log10(Gap)")
    axs[1, 0].set_title("log10(Gap) vs CPU Time")
    axs[1, 0].legend()

    axs[1, 1].plot(time_values_fw, np.log10(fx_values_fw), label="FW")
    axs[1, 1].plot(time_values_pw, np.log10(fx_values_pw), label="PW")
    axs[1, 1].set_xlabel("CPU Time")
    axs[1, 1].set_ylabel("log10(f(x))")
    axs[1, 1].set_title("log10(f(x)) vs CPU Time")
    axs[1, 1].legend()

    plt.tight_layout()
    plt.show()

```

## ▼ Datasets:

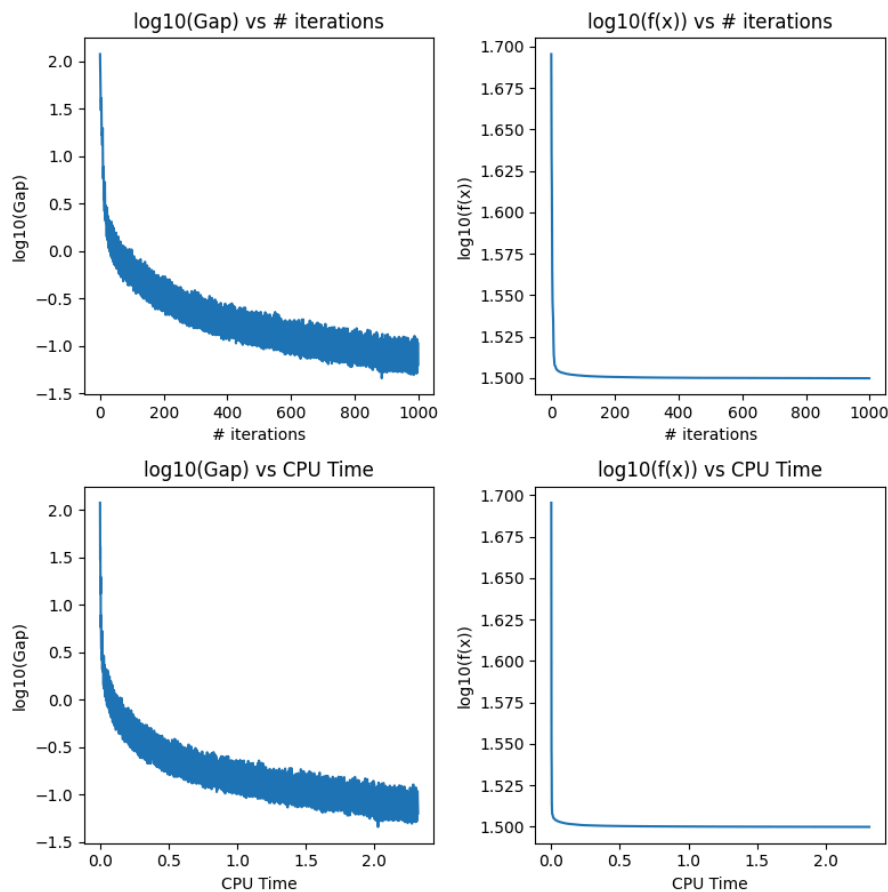
```
def concrete_data():
    df = pd.read_csv("/content/drive/My Drive/Concrete_Data_Yeh.csv")
    X_train = df[['cement', 'slag', 'flyash', 'water', 'superplasticizer', 'coarseaggregate', 'fineaggregate', 'age']]
    y_train = df['csMPa']
    X_train_norm = (X_train - X_train.min()) / (X_train.max() - X_train.min())
    y_train_norm = (y_train - y_train.min()) / (y_train.max() - y_train.min())
    X_train_norm = X_train_norm.to_numpy()
    y_train_norm = y_train_norm.to_numpy()
    return X_train_norm, y_train_norm

def boston_housing_data():
    column_names = ["CRIM", "ZN", "INDUS", "CHAS", "NOX", "RM", "AGE", "DIS", "RAD",
                    "TAX", "PTRATIO", "B", "LSTAT", "MEDV"]
    url = "https://archive.ics.uci.edu/ml/machine-learning-databases/housing/housing.data"
    df = pd.read_csv(url, delim_whitespace=True, names=column_names)
    X = df.drop("MEDV", axis=1)
    y = df["MEDV"]
    X_norm = (X - X.min()) / (X.max() - X.min())
    y_norm = (y - y.min()) / (y.max() - y.min())
    return X_norm.values, y_norm.values
```

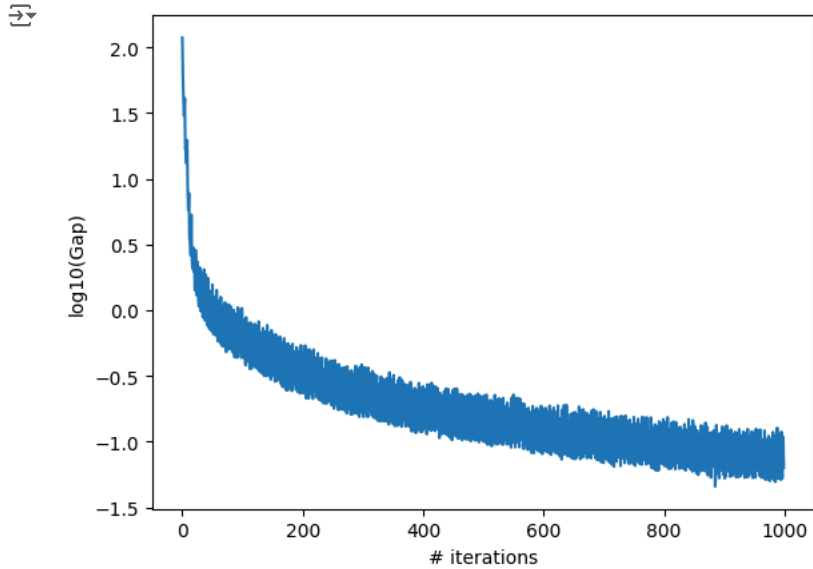
## ✓ Results for concrete data:

```
X_train_norm, y_train_norm = concrete_data()
x = initialize_x(A=X_train_norm)
x_fw, time_values_fw, fx_values_fw, gap_values_fw = frank_wolfe(x=x, A=X_train_norm, b=y_train_norm, K=K, threshold=THRESHOLD, tau= TAU)
```

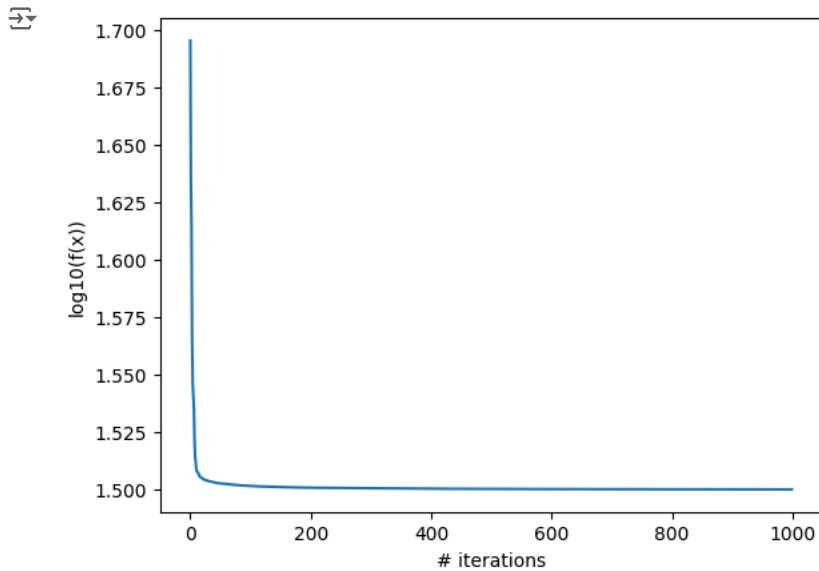
```
plot_graphs(gap_values_fw, fx_values_fw, time_values_fw)
```



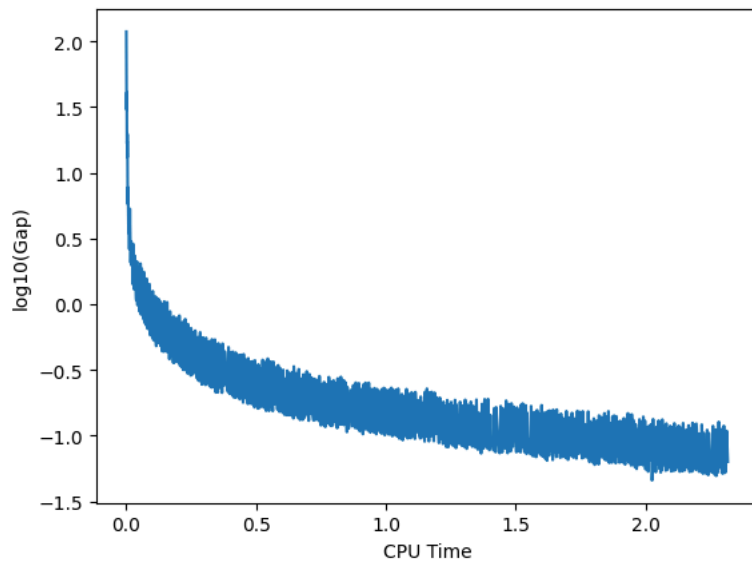
```
plt.plot(np.log10(gap_values_fw))  
plt.xlabel("# iterations")  
plt.ylabel("log10(Gap)")  
plt.show()
```



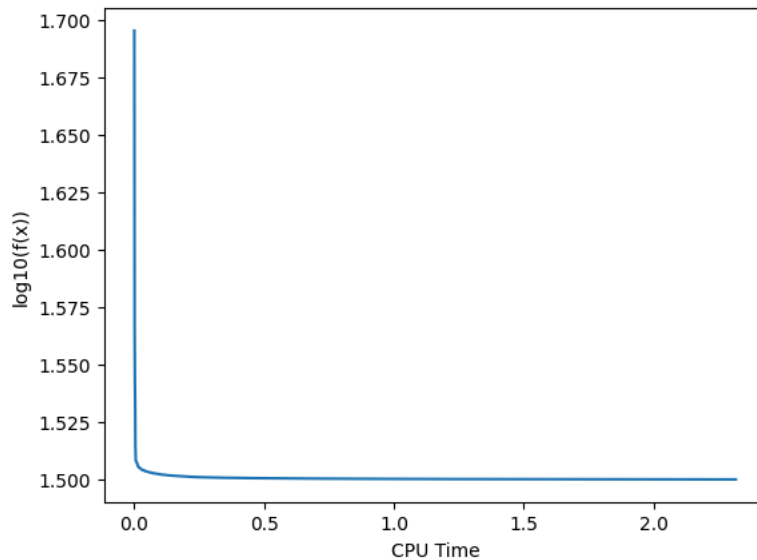
```
plt.plot(np.log10(fx_values_fw))  
plt.xlabel("# iterations")  
plt.ylabel("log10(f(x))")  
plt.show()
```



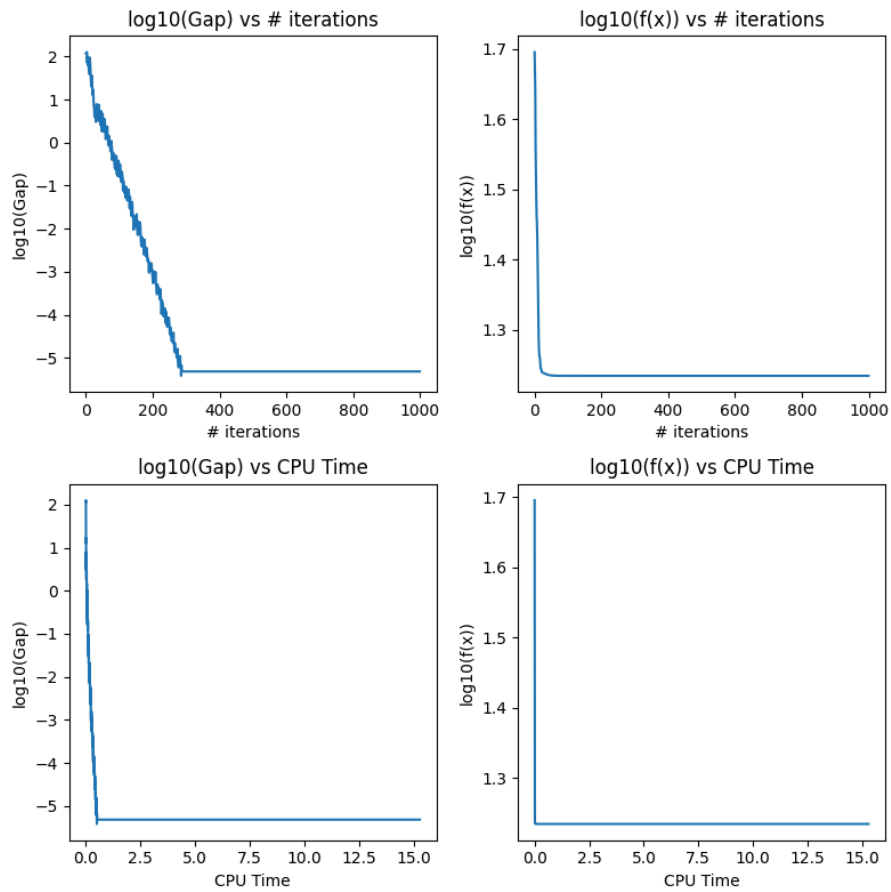
```
plt.plot(time_values_fw, np.log10(gap_values_fw))  
plt.xlabel("CPU Time")  
plt.ylabel("log10(Gap)")  
plt.show()
```



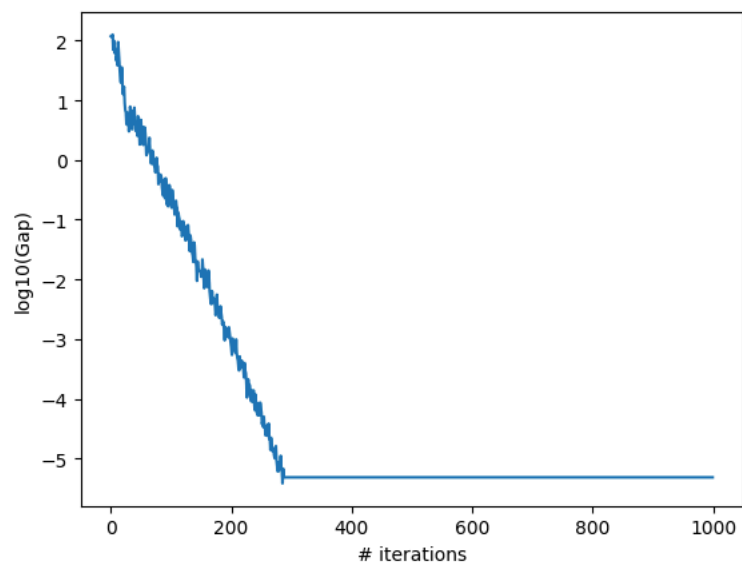
```
plt.plot(time_values_fw, np.log10(fx_values_fw))
plt.xlabel("CPU Time")
plt.ylabel("log10(f(x))")
plt.show()
```



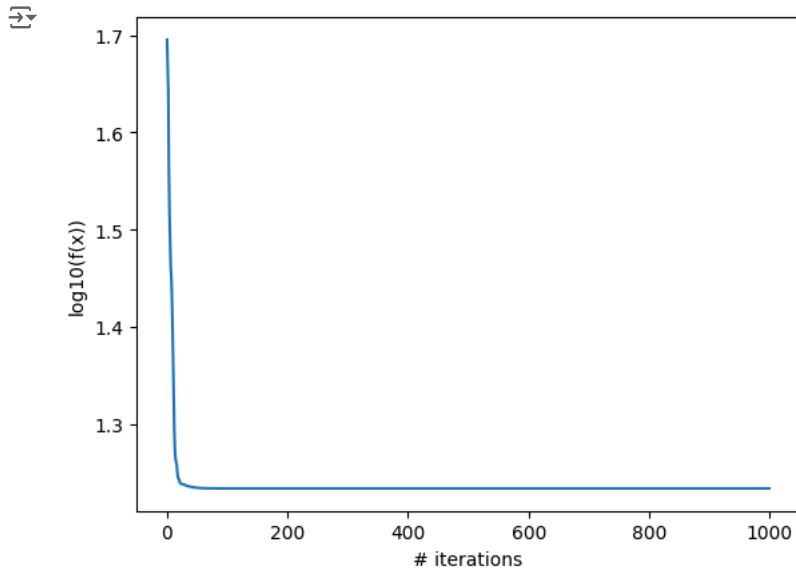
```
X_train_norm, y_train_norm = concrete_data()
x = initialize_x(A=X_train_norm)
x_pw, time_values_pw, fx_values_pw, gap_values_pw = pairwise_frank_wolfe(x=x, A=X_train_norm, b=y_train_norm, K=K, threshold=THRESHOLD, tau=
plot_graphs(gap_values_pw, fx_values_pw, time_values_pw)
```



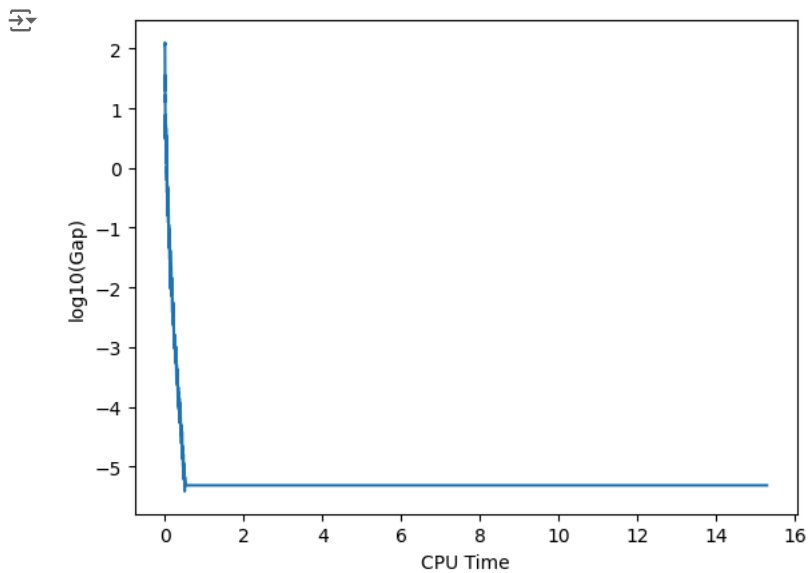
```
plt.plot(np.log10(gap_values_pw))
plt.xlabel("# iterations")
plt.ylabel("log10(Gap)")
plt.show()
```



```
plt.plot(np.log10(fx_values_pw))  
plt.xlabel("# iterations")  
plt.ylabel("log10(f(x))")  
plt.show()
```



```
plt.plot(time_values_pw, np.log10(gap_values_pw))  
plt.xlabel("CPU Time")  
plt.ylabel("log10(Gap)")  
plt.show()
```

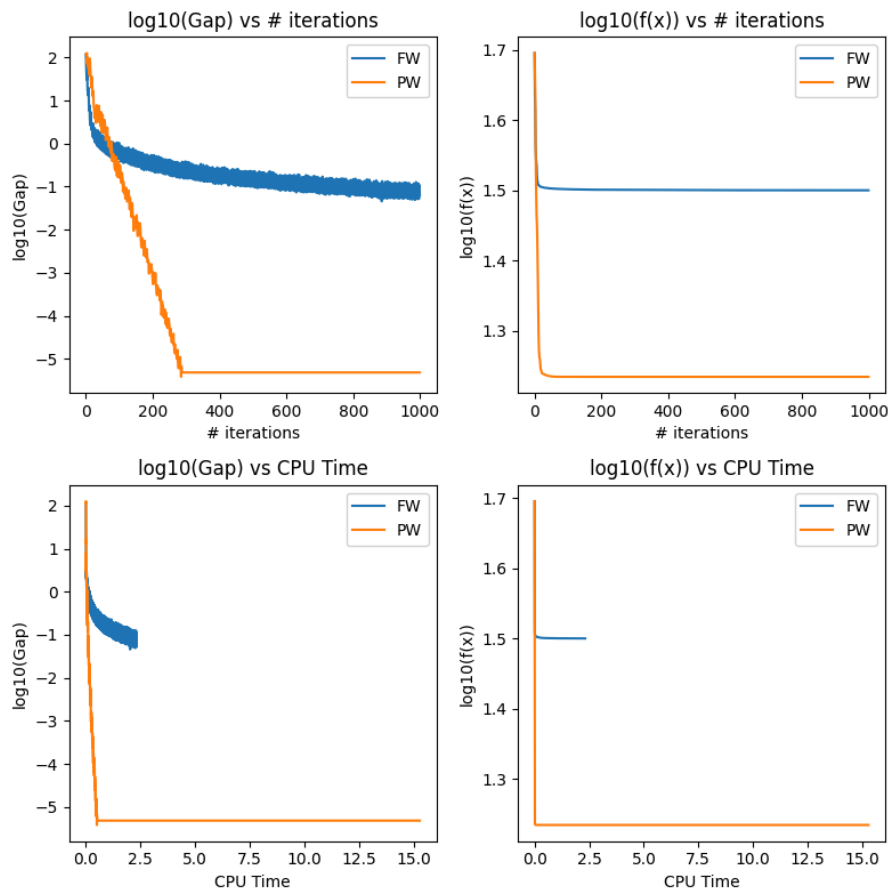


```
plt.plot(time_values_pw, np.log10(fx_values_pw))  
plt.xlabel("CPU Time")  
plt.ylabel("log10(f(x))")  
plt.show()
```





```
plot_all(gap_values_fw, fx_values_fw, time_values_fw, gap_values_pw, fx_values_pw, time_values_pw)
```



## ✓ Results for boston housing data:

```
X_norm, y_norm = boston_housing_data()
x = initialize_x(A=X_norm)
x_fw, time_values_fw, fx_values_fw, gap_values_fw = frank_wolfe(x=x, A=X_norm, b=y_norm, K=K, threshold=THRESHOLD, tau= TAU)

x = initialize_x(A=X_norm)
x_pw, time_values_pw, fx_values_pw, gap_values_pw = pairwise_frank_wolfe(x=x, A=X_norm, b=y_norm, K=K, threshold=THRESHOLD, tau= TAU)
```

```
plot_all(gap_values_fw, fx_values_fw, time_values_fw, gap_values_pw, fx_values_pw, time_values_pw)
```

