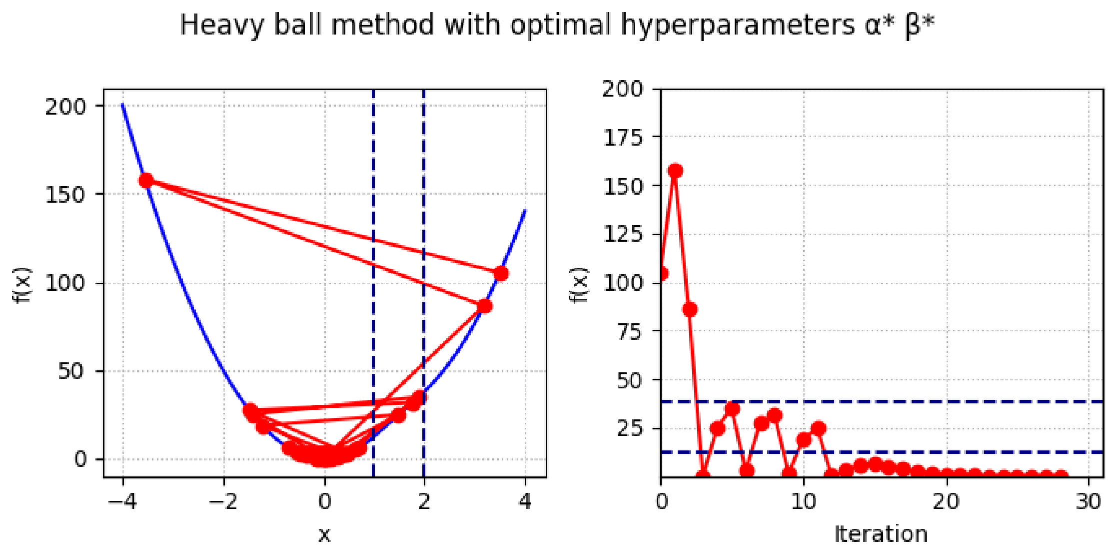
3.1

March 24, 2024

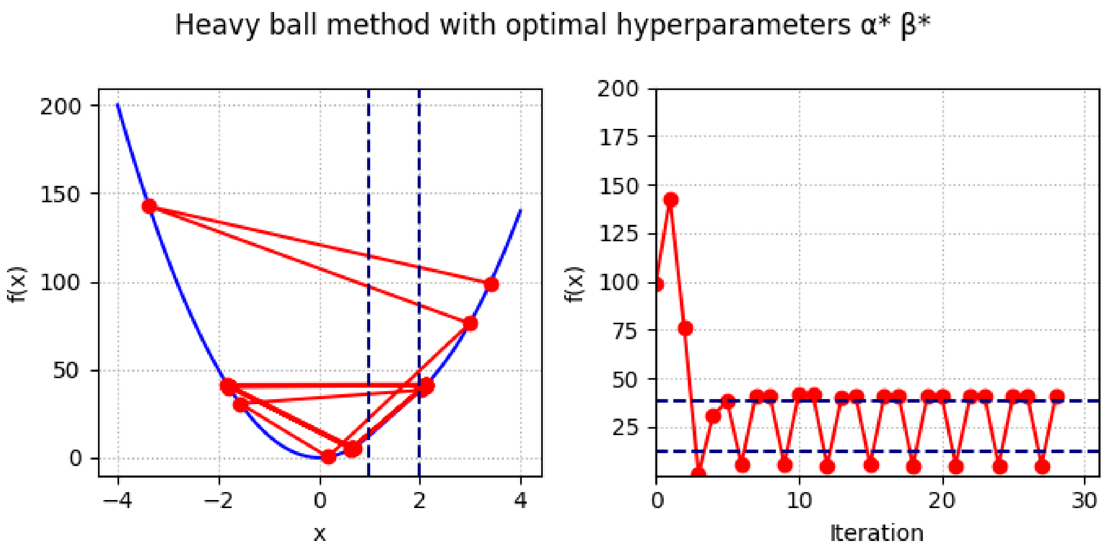
[1]:
```python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from IPython.display import HTML
```

[2]:
```python
def grad_f(x):
    if x < 1:
        return 25 * x
    if 1 <= x < 2:
        return x + 24
    return 25 * x - 24

def heavy_ball_method(alpha, beta, x0, num_iterations):
    x = np.zeros(num_iterations + 1)
    x_prev = x0
    x_curr = x0
    for i in range(num_iterations):
        x[i] = x_curr
        x_new = x_curr - alpha * grad_f(x_curr) + beta * (x_curr - x_prev)
        x_prev = x_curr
        x_curr = x_new
    x[num_iterations] = x_curr
    return x

L = 25  # directly from gradient
mu = 1.0  # from second order differential criterion
alpha_star = 4 / (np.sqrt(L) + np.sqrt(mu))**2
beta_star = (np.sqrt(L) - np.sqrt(mu))**2 / (np.sqrt(L) + np.sqrt(mu))**2
x0 = 3.5
num_iterations = 30

trajectory = heavy_ball_method(alpha_star, beta_star, x0, num_iterations)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(7, 3.5))
fig.suptitle("Heavy ball method with optimal hyperparameters")

def update(i):
    ax1.clear()
```

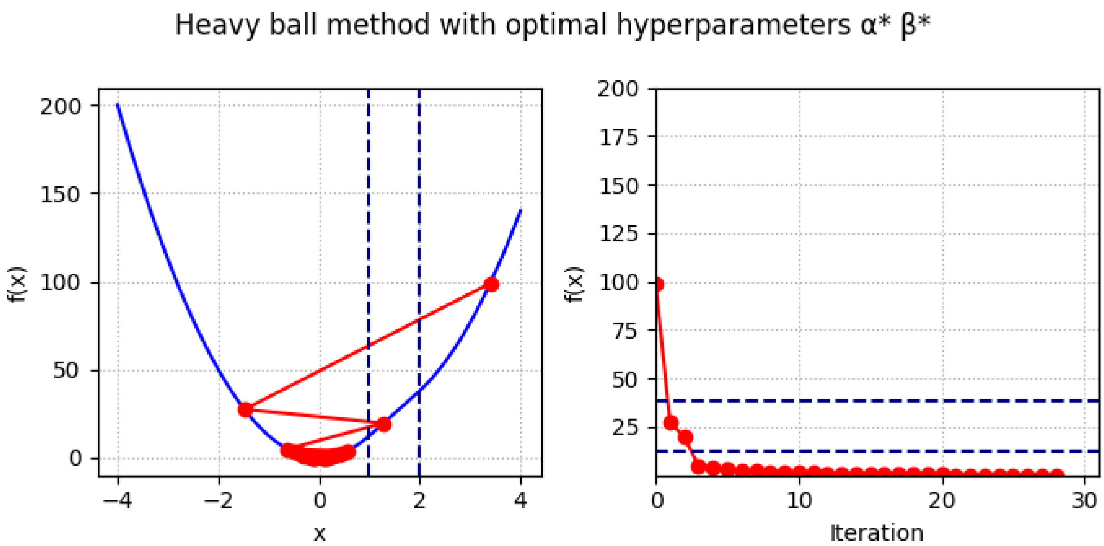
```python
    ax2.clear()

    x_vals = np.linspace(-4, 4, 100)
    f_vals = np.piecewise(x_vals, [x_vals < 1, (x_vals >= 1) & (x_vals < 2), x_vals >= 2],
                        [lambda x: 12.5 * x**2, lambda x: .5 * x**2 + 24 * x - 12, lambda x: 12.5 * x**2 - 24 * x + 36])
    ax1.plot(x_vals, f_vals, 'b-')
    ax1.plot(trajectory[:i], [12.5 * x**2 if x < 1 else .5 * x**2 + 24 * x - 12 if x < 2 else 12.5 * x**2 - 24 * x + 36 for x in trajectory[:i]], 'ro-')

    ax1.axvline(x=1, color='navy', linestyle='--')
    ax1.axvline(x=2, color='navy', linestyle='--')

    f_trajectory = [None for x in trajectory]
    f_trajectory[:i] = [12.5 * x**2 if x < 1 else .5 * x**2 + 24 * x - 12 if x < 2 else 12.5 * x**2 - 24 * x + 36 for x in trajectory[:i]]
    ax2.plot(range(len(trajectory)), f_trajectory, 'ro-')
    ax2.set_xlim(0, len(trajectory))
    ax2.set_ylim(min(f_vals), max(f_vals))

    f_1 = 12.5 * 1.0**2
    f_2 = .5 * 2.**2 + 24 * 2. - 12
    ax2.axhline(y=f_1, color='navy', linestyle='--')
    ax2.axhline(y=f_2, color='navy', linestyle='--')

    ax1.set_xlabel("x")
    ax1.set_ylabel("f(x)")
    ax1.grid(linestyle=":")

    ax2.set_xlabel("Iteration")
    ax2.set_ylabel("f(x)")
    ax2.grid(linestyle=":")

    plt.tight_layout()

for i in range(num_iterations):
    update(i)
```




```
[3]: L = 25   # directly from gradient
mu = 1.0   # from second order differential criterion
alpha_star = 4 / (np.sqrt(L) + np.sqrt(mu))**2
beta_star = (np.sqrt(L) - np.sqrt(mu))**2 / (np.sqrt(L) + np.sqrt(mu))**2
x0 = 3.4
num_iterations = 30

trajectory = heavy_ball_method(alpha_star, beta_star, x0, num_iterations)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(7, 3.5))
fig.suptitle("Heavy ball method with optimal hyperparameters")

for i in range(num_iterations):
    update(i)
```



[4]:
```
L = 25   # directly from gradient
mu = 1.0   # from second order differential criterion
alpha_star = 2 / L
beta_star = mu / L
x0 = 3.4
num_iterations = 30

trajectory = heavy_ball_method(alpha_star, beta_star, x0, num_iterations)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(7, 3.5))
fig.suptitle("Heavy ball method with optimal hyperparameters")

for i in range(num_iterations):
    update(i)
```


Heavy ball method with optimal hyperparameters α* β*
f(x)
200
150
100
50
0
−4
−2
0
2
4
x
f(x)
200
175
150
125
100
75
50
25
0
10
20
30
Iteration