Convex Optimization - Homework 3

1 Question 1

$$\min_{w} \ \frac{1}{2}\|Xw - y\|_2^2 + \lambda\|w\|_1,$$

The LASSO problem can be equivalently written :

$$\min_{w,z} \ \frac{1}{2}\|z\|_2^2 + \lambda\|w\|_1, \quad \text{subject to } z = Xw - y.$$

$$\mathcal{L}(w, z, v) = \frac{1}{2}\|z\|_2^2 + \lambda\|w\|_1 + v^T(z - Xw + y).$$

Minimization over z:

$$\mathcal{L}(z) = \frac{1}{2}\|z\|_2^2 + v^T z.$$

$$\nabla_z \mathcal{L} = z + v = 0 \quad \Rightarrow \quad z = -v.$$

Thus,

$$\mathcal{L}(w, v) = -\frac{1}{2}\|v\|_2^2 + v^T y + \lambda\|w\|_1 - v^T X w.$$

Minimization over w:

$$\mathcal{L}(w) = \lambda\|w\|_1 - v^T X w.$$

This is separable across components of w, and the minimizer is:

$$w_j = \begin{cases} 0 & \text{if } |(X^T v)_j| \leq \lambda, \\ \\ -\infty & \text{otherwise.} \end{cases}$$

This gives the constraint:

$$|X^T v| \leq \lambda.$$

Then, the dual problem is:

$$\max_{v} -\frac{1}{2}\|v\|_2^2 + v^T y \quad \text{subject to} \quad \|X^T v\|_\infty \leq \lambda.$$

The dual problem is equivalent to minimizing the negative of the objective:

$$\min_{v} \frac{1}{2}\|v\|_2^2 - v^T y \quad \text{subject to} \quad \|X^T v\|_\infty \leq \lambda.$$

We rewrite the dual problem in the standard QP form:

$$\min_{v} \ v^T Q v + p^T v \quad \text{subject to } Av \preceq b,$$

where:
- $Q = \frac{1}{2} I_n$
- $p = -y$,
- $A = \begin{bmatrix} X^T \\ -X^T \end{bmatrix}$,
- $b = \lambda \mathbf{1_{2d}}$, where $\mathbf{1_{2n}}$ is a vector of 2d ones.

2 Question 3

Changes in Iterations with Respect to μ:

After a threshold value of $\mu \approx 15$, the total number of iterations becomes largely insensitive to further changes in μ. A trade-off emerges between inner and outer iterations:

For smaller μ, more outer iterations are required for convergence, as the barrier is increased slowly. However, each outer step requires fewer inner (Newton) steps to converge.

To optimize the total computational cost, μ should be chosen to balance these effects. From experiments, an appropriate value for this problem lies in the range $\mu \in [50, 100]$, as it minimizes the total number of iterations required.

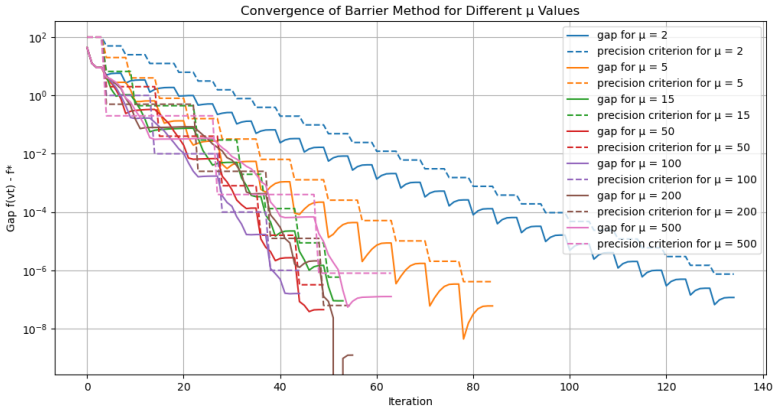


Figure 1: Convergence of Barrier Method for Different μ Values. The gap and precision criterion are shown on a semilog scale.

The solution remains stable regardless of changes in μ. While different values of μ may result in slightly different optimization paths, they always converge to the same final solution. This confirms that μ can be adjusted without worrying about affecting convergence.

HK3

December 2, 2024

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
def centering_step(Q, p, A, b, t, v0, eps, alpha=0.1, beta=0.7):
    """
    Implements the centering step using Newton's method with logarithmic␣
 ↪barriers and backtracking line search.

    Parameters:
        Q (numpy.ndarray): Positive semidefinite matrix of quadratic terms.
        p (numpy.ndarray): Linear term vector.
        A (numpy.ndarray): Constraint matrix.
        b (numpy.ndarray): Constraint vector.
        t (float): Barrier method parameter.
        v0 (numpy.ndarray): Initial variable.
        eps (float): Target precision.
        alpha (float): Line search parameter (typically small, e.g., 0.01).
        beta (float): Line search parameter (step size reduction factor,␣
 ↪typically 0.5).

    Returns:
        list: Sequence of variable iterates (vi).
    """
    def objective(v):
        """Barrier objective function."""
        return t * (v.T @ Q @ v + p.T @ v) - np.sum(np.log(b - A @ v))

    def backtracking_line_search(v, dv, grad):
        """Backtracking line search to find step size."""
        step_size = 1.0
        while True:
            new_v = v + step_size * dv
            if np.all(b - A @ new_v > 0) and objective(new_v) <= objective(v) +␣
 ↪alpha * step_size * grad.T @ dv:
                break
            step_size *= beta
        return step_size
```

```
    vi = [v0]
    v = v0
    i = 0
    while True and i < 100:
        gradient = t * (2 * Q @ v + p) + A.T @ (1 / (b - A @ v))

        D = np.diag(1 / (b - A @ v)**2)
        hessian = 2 * t * Q + A.T @ D @ A

        # Solve the Newton step (Hessian * dv = -grad)
        dv = np.linalg.solve(hessian, -gradient)

        decrement = np.sqrt(gradient.T @ -dv)

        # Check for convergence
        if decrement**2 / 2 <= eps:
            break

        step_size = backtracking_line_search(v, dv, gradient)

        v = v + step_size * dv

        vi.append(v)
        i += 1

    return vi
```

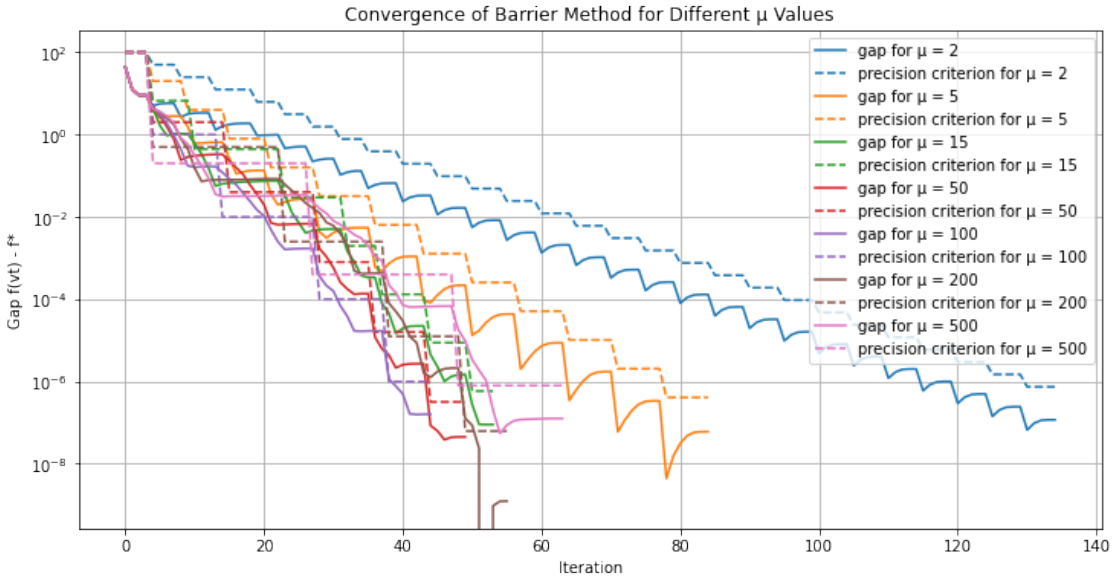
[3]:
```
def barr_method(Q, p, A, b, v0, eps, mu=10, t0=1):
    """
    Implements the barrier method to solve QP using the centering_step function.

    Parameters:
        Q (numpy.ndarray): Positive semidefinite matrix of quadratic terms.
        p (numpy.ndarray): Linear term vector.
        A (numpy.ndarray): Constraint matrix.
        b (numpy.ndarray): Constraint vector.
        v0 (numpy.ndarray): Initial feasible point.
        eps (float): Precision criterion.
        mu (float): Scaling parameter for barrier method (e.g., mu > 1).
        t0 (float): Initial value of t (e.g., t0 > 0).

    Returns:
        list: Sequence of variable iterates (vi).
    """

    t = t0
```

```
    v = v0
    vi_seq = [v0]
    seq_t = [t0]


    while True:

        vi = centering_step(Q, p, A, b, t, v, eps)
        v = vi[-1]  # Final iterate of centering step
        vi_seq.extend(vi[1:])
        seq_t += [t] * len(vi[:-1])

        # Check for convergence
        precision_criterion = len(b) / t
        if precision_criterion <= eps:
            break

        t *= mu

    return vi_seq, seq_t
```

[4]:
```
def test_barrier_method_dual(mu_values, Q, p, A, b, v0, eps):
    f_value_per_mu = []
    gap_per_mu = []
    final_v_per_mu = []
    t_per_mu = []
    surrogate_f_star = None

    for mu in mu_values:
        vi_seq, seq_t = barr_method(Q, p, A, b, v0, eps, mu=mu)
        final_v = vi_seq[-1]
        final_v_per_mu.append(final_v)
        t_per_mu.append(seq_t)

        f_vt = [v.T @ Q @ v + p.T @ v for v in vi_seq]
        f_value_per_mu.append(f_vt)

        f_min = min(f_vt)
        if surrogate_f_star is None or f_min < surrogate_f_star:
            surrogate_f_star = f_min

    for f_vt in f_value_per_mu:
        gap_per_mu.append([f - surrogate_f_star for f in f_vt])

    return final_v_per_mu, f_value_per_mu, t_per_mu, gap_per_mu,␣
↪surrogate_f_star
```

[5]:
```
def main():
    np.random.seed(42)
    n, d = 100, 50
    X = np.random.randn(n, d)
    y = np.random.randn(n)
    lambda_ = 10
    m = 2 * d

    Q = 0.5 * np.eye(n)
    p = -y
    A = np.vstack([X.T, -X.T])
    b = lambda_ * np.ones(2 * d)
    v0 = np.zeros(n)
    eps = 1e-6

    mu_values = [2, 5, 15, 50, 100, 200, 500]
    final_v_per_mu, _ , t_per_mu, gap_per_mu, _ =␣
 ↪test_barrier_method_dual(mu_values, Q, p, A, b, v0, eps)

    plt.figure(figsize=(12, 6))

    for i, mu in enumerate(mu_values):
        plt.semilogy(gap_per_mu[i], label=f"gap for   = {mu}", color=f"C{i}")

        plt.semilogy(m / np.array(t_per_mu[i]), label=f"precision criterion for␣
 ↪  = {mu}", linestyle='dashed', color=f"C{i}")

    plt.xlabel("Iteration")
    plt.ylabel("Gap f(vt) - f*")
    plt.title("Convergence of Barrier Method for Different   Values")
    plt.legend()
    plt.grid(True)
    plt.show()


    for i, mu in enumerate(mu_values):
        print(f"  = {mu}, Final solution v: {final_v_per_mu[i]}")

    print("\nAn appropriate   balances convergence speed and accuracy. Thus,␣
 ↪the best   value is 200.")

    return 0
```

[6]:
```
main()
```




```
 = 2, Final solution v: [-1.18072194e-01 -4.27466288e-01 -1.18794296e+00
-1.76276279e-01
  3.12986137e-01 -8.18743422e-01  1.32829831e+00  6.60167758e-01
 -3.33210203e-01 -2.41249234e-01  1.09181755e+00  6.62416167e-01
  1.75751911e-01 -2.04102271e-01 -1.34522220e-01 -3.88193561e-02
 -1.69438574e+00  1.73789076e+00  4.84316313e-01 -7.37811688e-02
 -3.62251744e-01 -1.07482684e+00 -3.45064199e-01 -1.25764657e+00
  1.87225680e+00 -7.86261154e-01 -6.27575892e-01 -6.73290447e-01
  5.84813645e-01  1.05212695e-01 -7.32913709e-01  1.05518116e+00
  3.53917416e-01  1.02780344e-01  4.98509762e-01  1.66789107e+00
  2.03288345e+00 -4.24244643e-01  8.57389576e-01  4.89054483e-01
  6.97388113e-01 -1.20639883e+00  3.83675595e-01 -1.94236703e-01
  1.06765972e+00  8.02024719e-01 -1.89817562e+00 -1.84532636e-01
  1.37430528e+00 -3.39303957e-01  4.07298951e-01  3.61229750e-01
  3.06354064e-01 -2.42400666e-01 -2.62677838e-01 -1.39157526e+00
  5.56755905e-01 -6.45380598e-02 -4.20337238e-01 -2.98104655e-01
 -2.72176552e-01  8.64065716e-01 -3.70144608e-01 -9.93135879e-01
  1.08354511e-01 -3.86338808e-01  1.60897134e-01  8.18382866e-01
  4.68803204e-02 -1.48749800e-02 -9.60411509e-01 -2.01449174e-01
  1.10206379e-01 -1.25491409e+00  2.75819509e-01  3.01000138e-01
  4.84897539e-01  9.90198223e-01  9.42638319e-01  1.15412026e+00
 -3.29618618e-01  3.29775115e-01  4.00422786e-02 -7.00011717e-01
  4.05639451e-01  1.92779201e+00 -1.58174251e-03 -1.37015217e-01
  2.18616545e-01 -3.81398309e-01 -1.17878369e+00 -5.81943520e-01
 -9.05911513e-01  1.45191302e+00  3.16507267e-01  2.58062036e-01
  7.92308150e-01 -6.66403311e-01 -3.03226632e-01 -1.75662311e-01]
 = 5, Final solution v: [-1.18072338e-01 -4.27466329e-01 -1.18794300e+00
-1.76276243e-01
```

```
 3.12986105e-01 -8.18743431e-01  1.32829838e+00  6.60167750e-01
-3.33210256e-01 -2.41249225e-01  1.09181754e+00  6.62416206e-01
 1.75751782e-01 -2.04102247e-01 -1.34522172e-01 -3.88193376e-02
-1.69438580e+00  1.73789096e+00  4.84316329e-01 -7.37811042e-02
-3.62251766e-01 -1.07482694e+00 -3.45064164e-01 -1.25764662e+00
 1.87225678e+00 -7.86261137e-01 -6.27575752e-01 -6.73290435e-01
 5.84813727e-01  1.05212704e-01 -7.32913740e-01  1.05518121e+00
 3.53917447e-01  1.02780359e-01  4.98509768e-01  1.66789092e+00
 2.03288353e+00 -4.24244618e-01  8.57389442e-01  4.89054437e-01
 6.97388162e-01 -1.20639881e+00  3.83675577e-01 -1.94236644e-01
 1.06765967e+00  8.02024562e-01 -1.89817564e+00 -1.84532654e-01
 1.37430530e+00 -3.39304048e-01  4.07298991e-01  3.61229782e-01
 3.06354099e-01 -2.42400651e-01 -2.62677771e-01 -1.39157521e+00
 5.56755817e-01 -6.45382323e-02 -4.20337155e-01 -2.98104654e-01
-2.72176759e-01  8.64065764e-01 -3.70144687e-01 -9.93135844e-01
 1.08354515e-01 -3.86338819e-01  1.60897156e-01  8.18382964e-01
 4.68803275e-02 -1.48749660e-02 -9.60411448e-01 -2.01449002e-01
 1.10206441e-01 -1.25491415e+00  2.75819506e-01  3.01000144e-01
 4.84897494e-01  9.90198219e-01  9.42638319e-01  1.15412038e+00
-3.29618546e-01  3.29775186e-01  4.00422550e-02 -7.00011745e-01
 4.05639309e-01  1.92779216e+00 -1.58169889e-03 -1.37015194e-01
 2.18616370e-01 -3.81398381e-01 -1.17878371e+00 -5.81943494e-01
-9.05911656e-01  1.45191301e+00  3.16507207e-01  2.58062068e-01
 7.92308200e-01 -6.66403286e-01 -3.03226567e-01 -1.75662379e-01]
 = 15, Final solution v: [-1.18072263e-01 -4.27466308e-01 -1.18794298e+00
-1.76276262e-01
 3.12986122e-01 -8.18743426e-01  1.32829835e+00  6.60167754e-01
-3.33210228e-01 -2.41249230e-01  1.09181754e+00  6.62416185e-01
 1.75751850e-01 -2.04102260e-01 -1.34522197e-01 -3.88193473e-02
-1.69438577e+00  1.73789086e+00  4.84316320e-01 -7.37811380e-02
-3.62251755e-01 -1.07482689e+00 -3.45064183e-01 -1.25764659e+00
 1.87225679e+00 -7.86261146e-01 -6.27575825e-01 -6.73290441e-01
 5.84813684e-01  1.05212699e-01 -7.32913724e-01  1.05518118e+00
 3.53917431e-01  1.02780351e-01  4.98509765e-01  1.66789100e+00
 2.03288349e+00 -4.24244631e-01  8.57389512e-01  4.89054461e-01
 6.97388136e-01 -1.20639882e+00  3.83675586e-01 -1.94236675e-01
 1.06765970e+00  8.02024644e-01 -1.89817563e+00 -1.84532644e-01
 1.37430529e+00 -3.39304000e-01  4.07298970e-01  3.61229765e-01
 3.06354081e-01 -2.42400659e-01 -2.62677806e-01 -1.39157523e+00
 5.56755863e-01 -6.45381419e-02 -4.20337198e-01 -2.98104654e-01
-2.72176651e-01  8.64065739e-01 -3.70144646e-01 -9.93135862e-01
 1.08354513e-01 -3.86338814e-01  1.60897145e-01  8.18382913e-01
 4.68803238e-02 -1.48749734e-02 -9.60411480e-01 -2.01449092e-01
 1.10206409e-01 -1.25491412e+00  2.75819508e-01  3.01000141e-01
 4.84897517e-01  9.90198221e-01  9.42638319e-01  1.15412032e+00
-3.29618584e-01  3.29775148e-01  4.00422674e-02 -7.00011730e-01
 4.05639384e-01  1.92779208e+00 -1.58172173e-03 -1.37015206e-01
 2.18616462e-01 -3.81398343e-01 -1.17878370e+00 -5.81943508e-01
```

```
 -9.05911581e-01  1.45191302e+00  3.16507238e-01  2.58062052e-01
  7.92308174e-01 -6.66403299e-01 -3.03226601e-01 -1.75662343e-01]
 = 50, Final solution v: [-1.18072376e-01 -4.27466340e-01 -1.18794301e+00
-1.76276233e-01
  3.12986096e-01 -8.18743434e-01  1.32829840e+00  6.60167748e-01
 -3.33210271e-01 -2.41249223e-01  1.09181753e+00  6.62416216e-01
  1.75751747e-01 -2.04102241e-01 -1.34522159e-01 -3.88193327e-02
 -1.69438582e+00  1.73789102e+00  4.84316333e-01 -7.37810870e-02
 -3.62251772e-01 -1.07482697e+00 -3.45064155e-01 -1.25764663e+00
  1.87225678e+00 -7.86261133e-01 -6.27575715e-01 -6.73290431e-01
  5.84813749e-01  1.05212706e-01 -7.32913749e-01  1.05518122e+00
  3.53917455e-01  1.02780363e-01  4.98509770e-01  1.66789088e+00
  2.03288355e+00 -4.24244611e-01  8.57389406e-01  4.89054425e-01
  6.97388175e-01 -1.20639881e+00  3.83675572e-01 -1.94236628e-01
  1.06765965e+00  8.02024520e-01 -1.89817565e+00 -1.84532658e-01
  1.37430530e+00 -3.39304072e-01  4.07299002e-01  3.61229790e-01
  3.06354109e-01 -2.42400647e-01 -2.62677753e-01 -1.39157520e+00
  5.56755793e-01 -6.45382784e-02 -4.20337133e-01 -2.98104654e-01
 -2.72176815e-01  8.64065777e-01 -3.70144709e-01 -9.93135834e-01
  1.08354516e-01 -3.86338822e-01  1.60897162e-01  8.18382991e-01
  4.68803293e-02 -1.48749623e-02 -9.60411431e-01 -2.01448956e-01
  1.10206458e-01 -1.25491416e+00  2.75819506e-01  3.01000146e-01
  4.84897482e-01  9.90198218e-01  9.42638318e-01  1.15412041e+00
 -3.29618527e-01  3.29775205e-01  4.00422487e-02 -7.00011752e-01
  4.05639271e-01  1.92779220e+00 -1.58168723e-03 -1.37015188e-01
  2.18616324e-01 -3.81398400e-01 -1.17878372e+00 -5.81943487e-01
 -9.05911694e-01  1.45191301e+00  3.16507191e-01  2.58062077e-01
  7.92308213e-01 -6.66403279e-01 -3.03226550e-01 -1.75662397e-01]
 = 100, Final solution v: [-1.18072086e-01 -4.27466257e-01 -1.18794293e+00
-1.76276307e-01
  3.12986162e-01 -8.18743414e-01  1.32829826e+00  6.60167764e-01
 -3.33210162e-01 -2.41249241e-01  1.09181756e+00  6.62416137e-01
  1.75752010e-01 -2.04102290e-01 -1.34522256e-01 -3.88193702e-02
 -1.69438570e+00  1.73789061e+00  4.84316301e-01 -7.37812179e-02
 -3.62251727e-01 -1.07482676e+00 -3.45064226e-01 -1.25764653e+00
  1.87225681e+00 -7.86261167e-01 -6.27575997e-01 -6.73290457e-01
  5.84813584e-01  1.05212689e-01 -7.32913685e-01  1.05518112e+00
  3.53917393e-01  1.02780333e-01  4.98509757e-01  1.66789119e+00
  2.03288340e+00 -4.24244663e-01  8.57389678e-01  4.89054518e-01
  6.97388076e-01 -1.20639884e+00  3.83675609e-01 -1.94236748e-01
  1.06765976e+00  8.02024838e-01 -1.89817560e+00 -1.84532622e-01
  1.37430527e+00 -3.39303888e-01  4.07298921e-01  3.61229727e-01
  3.06354036e-01 -2.42400677e-01 -2.62677889e-01 -1.39157529e+00
  5.56755972e-01 -6.45379287e-02 -4.20337300e-01 -2.98104655e-01
 -2.72176394e-01  8.64065680e-01 -3.70144548e-01 -9.93135905e-01
  1.08354508e-01 -3.86338800e-01  1.60897117e-01  8.18382792e-01
  4.68803151e-02 -1.48749907e-02 -9.60411556e-01 -2.01449304e-01
  1.10206332e-01 -1.25491405e+00  2.75819511e-01  3.01000134e-01
```

```
  4.84897573e-01  9.90198226e-01  9.42638319e-01  1.15412017e+00
 -3.29618672e-01  3.29775061e-01  4.00422965e-02 -7.00011696e-01
  4.05639560e-01  1.92779189e+00 -1.58177565e-03 -1.37015234e-01
  2.18616678e-01 -3.81398254e-01 -1.17878367e+00 -5.81943540e-01
 -9.05911405e-01  1.45191303e+00  3.16507312e-01  2.58062012e-01
  7.92308113e-01 -6.66403330e-01 -3.03226681e-01 -1.75662259e-01]
  = 200, Final solution v: [-1.18072486e-01 -4.27466371e-01 -1.18794304e+00
-1.76276205e-01
  3.12986071e-01 -8.18743441e-01  1.32829846e+00  6.60167742e-01
 -3.33210312e-01 -2.41249216e-01  1.09181752e+00  6.62416246e-01
  1.75751648e-01 -2.04102222e-01 -1.34522123e-01 -3.88193185e-02
 -1.69438587e+00  1.73789117e+00  4.84316345e-01 -7.37810374e-02
 -3.62251790e-01 -1.07482705e+00 -3.45064128e-01 -1.25764667e+00
  1.87225676e+00 -7.86261120e-01 -6.27575609e-01 -6.73290422e-01
  5.84813811e-01  1.05212713e-01 -7.32913773e-01  1.05518126e+00
  3.53917478e-01  1.02780374e-01  4.98509775e-01  1.66789076e+00
  2.03288361e+00 -4.24244591e-01  8.57389302e-01  4.89054390e-01
  6.97388213e-01 -1.20639880e+00  3.83675558e-01 -1.94236582e-01
  1.06765961e+00  8.02024400e-01 -1.89817567e+00 -1.84532672e-01
  1.37430532e+00 -3.39304142e-01  4.07299033e-01  3.61229814e-01
  3.06354137e-01 -2.42400636e-01 -2.62677702e-01 -1.39157516e+00
  5.56755726e-01 -6.45384109e-02 -4.20337070e-01 -2.98104653e-01
 -2.72176974e-01  8.64065813e-01 -3.70144769e-01 -9.93135808e-01
  1.08354518e-01 -3.86338831e-01  1.60897179e-01  8.18383066e-01
  4.68803347e-02 -1.48749515e-02 -9.60411384e-01 -2.01448824e-01
  1.10206505e-01 -1.25491420e+00  2.75819504e-01  3.01000150e-01
  4.84897448e-01  9.90198216e-01  9.42638318e-01  1.15412050e+00
 -3.29618472e-01  3.29775259e-01  4.00422306e-02 -7.00011774e-01
  4.05639161e-01  1.92779231e+00 -1.58165374e-03 -1.37015170e-01
  2.18616190e-01 -3.81398456e-01 -1.17878374e+00 -5.81943467e-01
 -9.05911804e-01  1.45191300e+00  3.16507145e-01  2.58062101e-01
  7.92308250e-01 -6.66403259e-01 -3.03226501e-01 -1.75662450e-01]
  = 500, Final solution v: [-1.18072171e-01 -4.27466282e-01 -1.18794295e+00
-1.76276285e-01
  3.12986143e-01 -8.18743420e-01  1.32829830e+00  6.60167759e-01
 -3.33210194e-01 -2.41249236e-01  1.09181755e+00  6.62416160e-01
  1.75751933e-01 -2.04102275e-01 -1.34522228e-01 -3.88193592e-02
 -1.69438573e+00  1.73789073e+00  4.84316310e-01 -7.37811794e-02
 -3.62251740e-01 -1.07482682e+00 -3.45064205e-01 -1.25764656e+00
  1.87225680e+00 -7.86261157e-01 -6.27575914e-01 -6.73290449e-01
  5.84813632e-01  1.05212694e-01 -7.32913704e-01  1.05518115e+00
  3.53917412e-01  1.02780342e-01  4.98509761e-01  1.66789110e+00
  2.03288344e+00 -4.24244647e-01  8.57389598e-01  4.89054491e-01
  6.97388105e-01 -1.20639883e+00  3.83675598e-01 -1.94236712e-01
  1.06765973e+00  8.02024744e-01 -1.89817561e+00 -1.84532633e-01
  1.37430528e+00 -3.39303942e-01  4.07298945e-01  3.61229745e-01
  3.06354058e-01 -2.42400668e-01 -2.62677849e-01 -1.39157526e+00
  5.56755919e-01 -6.45380315e-02 -4.20337251e-01 -2.98104655e-01
```

```
 -2.72176518e-01  8.64065708e-01 -3.70144595e-01 -9.93135884e-01
  1.08354511e-01 -3.86338806e-01  1.60897131e-01  8.18382850e-01
  4.68803193e-02 -1.48749823e-02 -9.60411519e-01 -2.01449202e-01
  1.10206369e-01 -1.25491409e+00  2.75819510e-01  3.01000138e-01
  4.84897546e-01  9.90198224e-01  9.42638319e-01  1.15412024e+00
 -3.29618629e-01  3.29775103e-01  4.00422825e-02 -7.00011712e-01
  4.05639475e-01  1.92779198e+00 -1.58174965e-03 -1.37015221e-01
  2.18616574e-01 -3.81398297e-01 -1.17878368e+00 -5.81943524e-01
 -9.05911490e-01  1.45191303e+00  3.16507276e-01  2.58062031e-01
  7.92308142e-01 -6.66403315e-01 -3.03226642e-01 -1.75662300e-01]
```

An appropriate balances convergence speed and accuracy. Thus, the best value is 200.

[6]: 0

[ ]: