Exercise 1

8, 4, 6

(a)

In [5]:
```python
# Imports
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import NearestNeighbors
import time
```

In [6]:
```python
# 1) Load data
data_path = 'data/dijet_features_normalized.npy'
labels_path = 'data/dijet_labels.npy'
X = np.load(data_path)
labels = np.load(labels_path) if (__import__('os').path.exists(labels_path)) else None
print('raw X shape:', X.shape)
# If labels exist, try to align them with samples. If labels length matches columns, transpose X.
if labels is not None:
    print('raw labels shape:', labels.shape)
    if labels.shape[0] == X.shape[0]:
        print('Labels match number of rows -> using rows as samples')
    elif labels.shape[0] == X.shape[1]:
        print('Labels match number of columns -> transposing X to have samples as rows')
        X = X.T
        print('new X shape:', X.shape)
    else:
        print('Warning: labels length does not match rows or columns of X; ignoring labels')
        labels = None
# final sample count N (rows of X)
N = X.shape[0]
print('Number of samples N =', N)
```

```
raw X shape: (116, 2233)
raw labels shape: (2233,)
Labels match number of columns -> transposing X to have samples as rows
new X shape: (2233, 116)
Number of samples N = 2233
```

In [7]:
```python
# 2) Build symmetrized kNN graph (k=15) and extract edges E
k = 15
nbrs = NearestNeighbors(n_neighbors=k, algorithm='auto', metric='euclidean').fit(X)
dists, inds = nbrs.kneighbors(X)
N = X.shape[0]
# Build directed adjacency then symmetrize
adj = np.zeros((N, N), dtype=bool)
for i in range(N):
    adj[i, inds[i]] = True
# symmetrize
adj = np.logical_or(adj, adj.T)
# extract edges as pairs i < j to avoid duplicates
edges_i, edges_j = np.where(np.triu(adj, k=1))
E = np.vstack([edges_i, edges_j]).T
print('Number of nodes N =', N)
print('Number of edges |E| =', E.shape[0])
```

```
Number of nodes N = 2233
Number of edges |E| = 24595
```

In [8]:
```python
# 3) Sample random repulsive pairs: |R| = 5N
def sample_repulsive_pairs(N, factor=5, rng=None):
    # returns arrays (a,b) of shape (R,) with a != b
    if rng is None:
        rng = np.random.default_rng()
    R = int(factor * N)
    a = rng.integers(0, N, size=R, endpoint=False)
    b = rng.integers(0, N, size=R, endpoint=False)
    # avoid self-pairs by resampling those entries
    mask = (a == b)
    while mask.any():
        b[mask] = rng.integers(0, N, size=mask.sum(), endpoint=False)
        mask = (a == b)
    return a, b

# quick test
a, b = sample_repulsive_pairs(100, factor=5)
print('sampled repulsive pairs:', a.shape[0])
```

```
sampled repulsive pairs: 500
```

In [9]:
```python
# 4) Parameters and initialization
c = 10.0  # repulsion strength
seed = 42
rng = np.random.default_rng(seed)
# 5) Initialize 2D embeddings
scale = 0.01
Y = rng.normal(scale=scale, size=(N, 2))
print('Initialized embeddings Y shape:', Y.shape)
```

```
Initialized embeddings Y shape: (2233, 2)
```

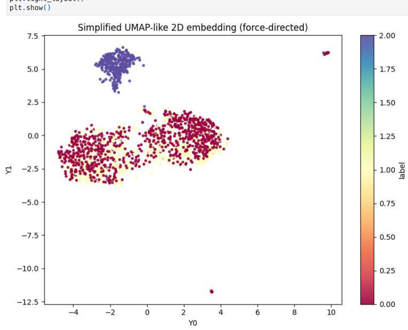
In [10]:
```python
# 6) Force (gradient) computations using analytical derivatives
def compute_gradients(Y, E, rep_a, rep_b, c):
    # Y: (N,2), E: (M,2) with (i,j) pairs, rep_a/b: (R,)
    N = Y.shape[0]
    grad = np.zeros_like(Y)
    # Attractive term over edges E
    if E.shape[0] > 0:
        i = E[:, 0]
        j = E[:, 1]
        dif = Y[i] - Y[j]  # (M,2)
        d2 = np.sum(dif * dif, axis=1)  # (M,)
        denom = 1.0 + d2  # (M,)
        # gradient contribution: 2*(yi-yj)/(1+d2) for i; symmetric for j
        coef = (2.0 / denom)[:, None]  # (M,1)
        contrib = coef * dif  # (M,2)
        # accumulate
        np.add.at(grad, i, contrib)
        np.add.at(grad, j, -contrib)  # opposite sign for j
    # Repulsive term over sampled random pairs
    if rep_a.size > 0:
        a = rep_a
        b = rep_b
        difr = Y[a] - Y[b]  # (R,2)
        d2r = np.sum(difr * difr, axis=1)  # (R,)
        denom2 = (1.0 + d2r) ** 2  # (R,)
        coefr = (-2.0 * c / denom2)[:, None]  # (R,1)
        contribr = coefr * difr  # (R,2) this is grad for a
        np.add.at(grad, a, contribr)
        np.add.at(grad, b, -contribr)  # symmetric
    return grad
```

In [11]:
```python
# 7) Optimization: gradient descent with decreasing learning rate
def optimize(Y, E, N, c, rng, n_iter=300, lr0=0.5, rep_factor=5, verbose=True):
    Y = Y.copy()
    history = {'loss': []}
    for it in range(n_iter):
        # linear learning-rate decay
        lr = lr0 * (1.0 - (it / float(n_iter)))
        # resample repulsive pairs each iteration
        a, b = sample_repulsive_pairs(N, factor=rep_factor, rng=rng)
        grad = compute_gradients(Y, E, a, b, c)
        # update positions (gradient descent)
        Y -= lr * grad
        # compute loss (optional) every 10 iters
        if (it % 10) == 0 or it == n_iter - 1:
            # attractive loss
            if E.shape[0] > 0:
                di = Y[E[:, 0]] - Y[E[:, 1]]
                d2i = np.sum(di * di, axis=1)
                loss_attr = np.sum(np.log(1.0 + d2i))
            else:
                loss_attr = 0.0
            # repulsive loss (estimate on sampled pairs)
            dr = Y[a] - Y[b]
            d2r = np.sum(dr * dr, axis=1)
            loss_rep = np.sum(c / (1.0 + d2r))
            loss = loss_attr + loss_rep
            history['loss'].append((it, loss_attr, loss_rep, loss))
            if verbose:
                print(f'it {it:4d}/{n_iter}  lr={lr:.4f}  loss_attr={loss_attr:.4e}  loss_rep={loss_rep:.4e}  total={loss:.4e}')
    return Y, history

# Run optimization (keep iterations modest for demo; increase as needed)
start = time.time()
Y_opt, history = optimize(Y, E, N, c, rng, n_iter=300, lr0=0.5, rep_factor=5, verbose=True)
print('Optimization time (s):', time.time() - start)
```

```
it    0/300  lr=0.5000  loss_attr=2.7232e+04  loss_rep=3.6079e+04  total=6.3310e+04
it   10/300  lr=0.4833  loss_attr=6.5987e+04  loss_rep=9.5244e+03  total=7.5511e+04
it   20/300  lr=0.4667  loss_attr=6.1906e+04  loss_rep=8.9342e+03  total=7.0840e+04
it   30/300  lr=0.4500  loss_attr=6.1852e+04  loss_rep=7.5037e+03  total=6.9356e+04
it   40/300  lr=0.4333  loss_attr=5.9280e+04  loss_rep=7.1517e+03  total=6.6432e+04
it   50/300  lr=0.4167  loss_attr=5.5931e+04  loss_rep=6.9223e+03  total=6.2853e+04
it   60/300  lr=0.4000  loss_attr=5.6606e+04  loss_rep=6.8719e+03  total=6.3478e+04
it   70/300  lr=0.3833  loss_attr=5.5412e+04  loss_rep=6.9626e+03  total=6.2375e+04
it   80/300  lr=0.3667  loss_attr=5.3822e+04  loss_rep=6.9836e+03  total=6.0805e+04
it   90/300  lr=0.3500  loss_attr=5.4049e+04  loss_rep=7.0371e+03  total=6.1086e+04
it  100/300  lr=0.3333  loss_attr=5.2471e+04  loss_rep=7.0515e+03  total=5.9523e+04
it  110/300  lr=0.3167  loss_attr=5.1366e+04  loss_rep=7.2349e+03  total=5.8601e+04
it  120/300  lr=0.3000  loss_attr=5.0463e+04  loss_rep=7.3182e+03  total=5.7781e+04
it  130/300  lr=0.2833  loss_attr=4.8125e+04  loss_rep=7.1319e+03  total=5.5257e+04
it  140/300  lr=0.2667  loss_attr=4.7182e+04  loss_rep=7.3365e+03  total=5.4518e+04
it  150/300  lr=0.2500  loss_attr=4.4775e+04  loss_rep=7.7075e+03  total=5.2483e+04
it  160/300  lr=0.2333  loss_attr=4.3729e+04  loss_rep=8.0714e+03  total=5.1800e+04
it  170/300  lr=0.2167  loss_attr=4.2220e+04  loss_rep=7.9646e+03  total=5.0185e+04
it  180/300  lr=0.2000  loss_attr=4.0815e+04  loss_rep=8.1515e+03  total=4.8967e+04
it  190/300  lr=0.1833  loss_attr=3.8747e+04  loss_rep=8.4109e+03  total=4.7158e+04
it  200/300  lr=0.1667  loss_attr=3.7377e+04  loss_rep=8.5560e+03  total=4.5933e+04
it  210/300  lr=0.1500  loss_attr=3.4731e+04  loss_rep=8.7641e+03  total=4.3495e+04
it  220/300  lr=0.1333  loss_attr=3.3004e+04  loss_rep=9.0664e+03  total=4.2071e+04
it  230/300  lr=0.1167  loss_attr=3.1316e+04  loss_rep=9.2130e+03  total=4.0529e+04
it  240/300  lr=0.1000  loss_attr=2.8685e+04  loss_rep=9.3981e+03  total=3.8083e+04
it  250/300  lr=0.0833  loss_attr=2.6273e+04  loss_rep=9.8702e+03  total=3.6143e+04
it  260/300  lr=0.0667  loss_attr=2.2985e+04  loss_rep=1.0431e+04  total=3.3416e+04
it  270/300  lr=0.0500  loss_attr=2.0728e+04  loss_rep=1.0770e+04  total=3.1499e+04
it  280/300  lr=0.0333  loss_attr=1.8405e+04  loss_rep=1.1674e+04  total=3.0079e+04
it  290/300  lr=0.0167  loss_attr=1.7207e+04  loss_rep=1.3039e+04  total=3.0245e+04
it  299/300  lr=0.0017  loss_attr=1.6838e+04  loss_rep=1.4148e+04  total=3.0986e+04
Optimization time (s): 0.8495810031890869
```

In [12]:
```python
# 8) Plot final embedding colored by labels (if available)
plt.figure(figsize=(8, 6))
if labels is None:
    plt.scatter(Y_opt[:, 0], Y_opt[:, 1], s=5, alpha=0.8)
else:
    sc = plt.scatter(Y_opt[:, 0], Y_opt[:, 1], c=labels, s=8, cmap='Spectral', alpha=0.9)
    plt.colorbar(sc, label='label')
plt.title('Simplified UMAP-like 2D embedding (force-directed)')
plt.xlabel('Y0')
plt.ylabel('Y1')
plt.grid(False)
plt.tight_layout()
plt.show()
```



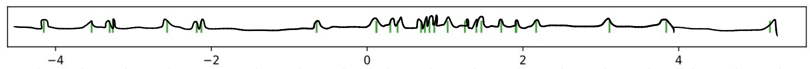
Very good 8/8

(b)

In contrast to last week, we can now clearly distinguish one of the particle types, while the others are still mixed up.

② (a) Bandwith very small: Noisy result, every point has its own small peak

3/3



Bandwith too large: Flat/smooth curve → Some structures can not be seen

For this data set: bandwith $\approx 0{,}5$

(b) Depending on the density of points, a smaller bandwith should be used for regions of higher density and a larger bandwith in less dense regions.

Use equations to give the proposed method

1/3

③ (a) (y_n, x_n)

$$\mathcal{N} = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-x_0)^2}{2\sigma^2}\right)$$

$$y_n = \beta^T x_n + \varepsilon_n$$

$$\hat{\beta} = \underset{\beta}{\operatorname{argmax}} \sum_{n=1}^{N} \log \mathcal{N}(y_n | \beta^T x, \sigma^2)$$

$$\hat{\sigma}^2 = \underset{\sigma^2}{\operatorname{argmax}} \sum_{n=1}^{N} \log \mathcal{N}(y_n | \hat{\beta}^T x, \sigma^2)$$

$$\vec{\beta} = \underset{\beta}{\operatorname{argmax}} \sum_{n=1}^{N} \log\left(\frac{1}{\sqrt{2\pi}\sigma} \cdot \exp\left(-\frac{(y_n - \beta^T x)^2}{\sigma^2}\right)\right)$$

$$= \underset{\beta}{\operatorname{argmin}} \sum_{n=1}^{N} (y_n - \beta^T x_n)^2 = (\vec{y} - X\beta)^T (\vec{y} - X\beta)$$

$$\to 0 \overset{!}{=} \nabla_\beta [(\vec{y} - X\beta)^T \cdot (\vec{y} - X\beta)]$$

$$\nabla_\beta a^T \beta = a$$

$$\vec{y}^T X = (X^T \vec{y})^T$$

$$= \nabla_\beta [-\vec{y}^T X\beta - (X\beta)^T \vec{y} + (X\beta)^T X\beta]$$

$$= -X^T \vec{y} - X^T \vec{y} + 2X^T X\beta$$

$$\Rightarrow X^T X \beta = X^T \vec{y}$$

$$\Leftrightarrow \beta = (X^T X)^{-1} X^T \vec{y}$$ ← Same as result from lecture

$$\hat{\sigma}^2 = \underset{\sigma^2}{\operatorname{argmax}} \sum_{n=1}^{N} \log\left(\frac{1}{\sqrt{2\pi\sigma^2}} \cdot \exp\left(-\frac{(y_n - \beta^T x)^2}{2\sigma^2}\right)\right)$$

$$= \underset{\sigma^2}{\operatorname{argmax}} \left[N \cdot \log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) - \sum_{n=1}^{N} \frac{(y_n - \beta^T x)^2}{2\sigma^2}\right.$$

$$= \underset{\sigma^2}{\operatorname{argmax}} \left[-\frac{N}{2} \log(\sigma^2) - \frac{1}{2\sigma^2} \sum_{n=1}^{N} (y_n - \beta^T x)^2\right.$$

$$0 \overset{!}{=} \frac{\partial}{\partial \sigma^2} [\dots] = -\frac{N}{2\sigma^2} + \frac{1}{2\sigma^4} \sum_{n=1}^{N} \dots$$

$$N = \frac{1}{\sigma^2} \sum_{n=1}^{N} (y_n - \beta^T x)^2$$

$$\sigma^2 = \frac{1}{N} \sum_{n=1}^{N} (y_n - \beta^T x)^2$$

(b) $y_n = \beta^T x_n + \varepsilon_n$

$\mathbb{E}(\varepsilon_n) = 0$, $\operatorname{var}(\varepsilon_n) = \sigma_n^2$ ← Now σ_n, not σ

$$\vec{\beta} = \underset{\beta}{\operatorname{argmax}} \sum_{n=1}^{N} \log\left(\frac{1}{\sqrt{2\pi}\sigma_n} \cdot \exp\left(-\frac{(y_n - \beta^T x)^2}{\sigma_n^2}\right)\right)$$

$$= \underset{\beta}{\operatorname{argmin}} \sum_{n=1}^{N} \frac{(y_n - \beta^T x)^2}{\sigma_n^2}$$

→ every summand is weighted by $\frac{1}{\sigma_n^2}$

$$\rightarrow \underset{\beta}{\operatorname{argmin}} (\vec{y} - X\beta)^T \underset{\substack{\uparrow \\ \text{includes weighting factors}}}{W} (\vec{y} - X\beta)$$

$$0 \overset{!}{=} \nabla_\beta \left[-\vec{y}^T W X\beta - (X\beta)^T W \vec{y} + (X\beta)^T W X \beta \right]$$

$$-X^T W^T \vec{y} - X^T W \vec{y} + 2X^T W X \beta$$

$$\rightarrow (X^T W X)\beta = X^T W \vec{y}$$

$$\beta = (X^T W X)^{-1} X^T W \vec{y}$$

Excellent, 6/6

W is symmetric

$$\begin{pmatrix} \sigma_1^{-2} & & \\ & \ddots & \\ & & \sigma_N^{-2} \end{pmatrix} \Rightarrow W^T = W$$