sheet02

November 6, 2022

1 Sheet 2

Jacob Waßelewsky, Johannes Schmidt, Dominik Geng

[1]:
```python
import numpy as np
from matplotlib import pyplot as plt
from scipy.stats import gaussian_kde
```

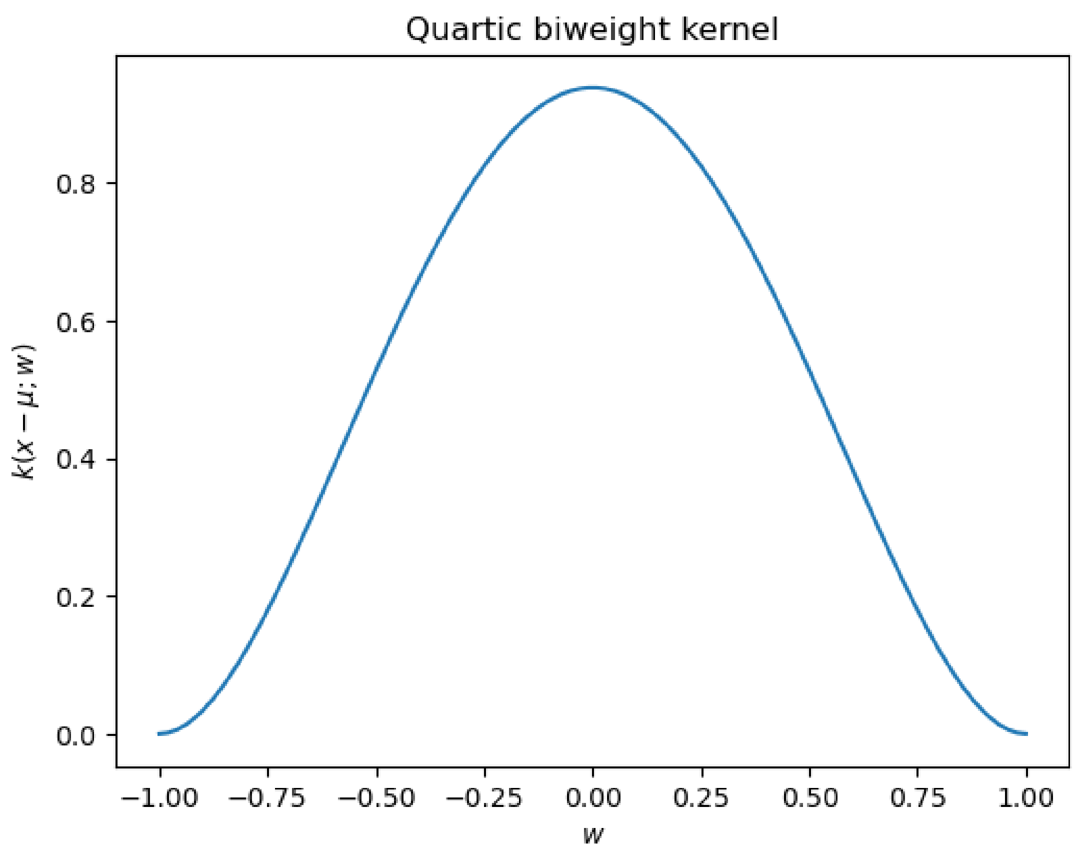
1.1 1 Kernel Density Estimation

(a)

[2]:
```python
def biweight(x, mu, w):
    """biweight kernel at mean mu, with bandwidth w evaluated at x"""
    if (x - mu > -w) and (x - mu < w):
        k = 15/(16*w) * (1 - ((x-mu)/w)**2)**2
    else:
        k = 0
    return k
    #TODO: implement the quartic (biweight) kernel
```

[3]:
```python
mu = 0
w = 1
domain = np.linspace(-1,1,num = 100)
kernel_vals = [biweight(x, mu, w) for x in domain]
plt.plot(domain, kernel_vals)
plt.ylabel(f'$k(x-\mu;w)$')
plt.xlabel(f'$w$')
plt.title('Quartic biweight kernel')
# TODO plot the kernel
```

[3]:
```
Text(0.5, 1.0, 'Quartic biweight kernel')
```



We can see a symmetric kernel with support $[-1, 1]$.

(b)

```
[4]: # load the data
     data = np.load("data/samples.npy")
     data50 = data[:50]
     print(f'{data.shape=}, {data50.shape=}')
```

```
data.shape=(10000,), data50.shape=(50,)
```

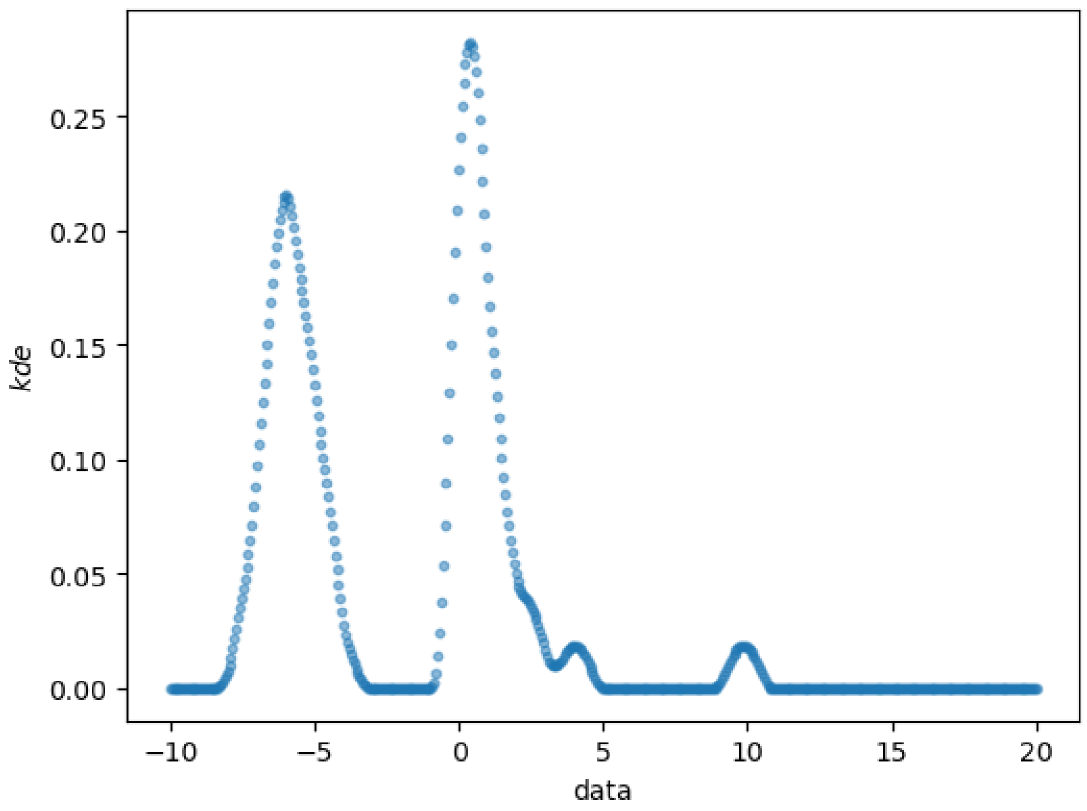
```
[6]: x_ = data50[4] # example x
     x_ - data50
     print('The result is broadcasted: \n x-data50 = ', x_-data50)

     kde_ = 0
     for x in data50:
         kde_ += biweight(x_ - x, mu = 0, w = 1)
```

```
The result is broadcasted:
 x-data50 =  [ 0.956057    6.83620095  0.1527347  -0.16022261  0.
0.91014681
  7.4293298   0.51493953  6.60292255  0.44825482  8.14906403  5.97400304
  7.64591618  0.33843258  5.23943767  5.93484453  0.79503942  0.98899981
  0.63264979 -0.525491    0.86387563 -2.89483556  0.01106935  0.90581603
  6.81518695  6.0118808   6.7721106   5.7250845   7.12435507  7.38703073
 -1.49370655  8.50471502  7.33714949  7.53958145  1.10249879  6.40101085
  1.07391179  1.05302318  6.83940263  6.34695325  0.92471502  0.55073825
  7.62067136  7.2868227   7.01518574 -1.11829306 -0.1610926  -8.74403715
  8.15152502  1.02399578]
```

[7]:
```python
def kde(x, obs, w=1):
    s = 0
    for o in obs:
        s += biweight(x, mu = o, w = w)
    p_x_obs = 1/(len(obs)) * s
    return p_x_obs
    # TODO: implement the KDE with the biweight kernel
```

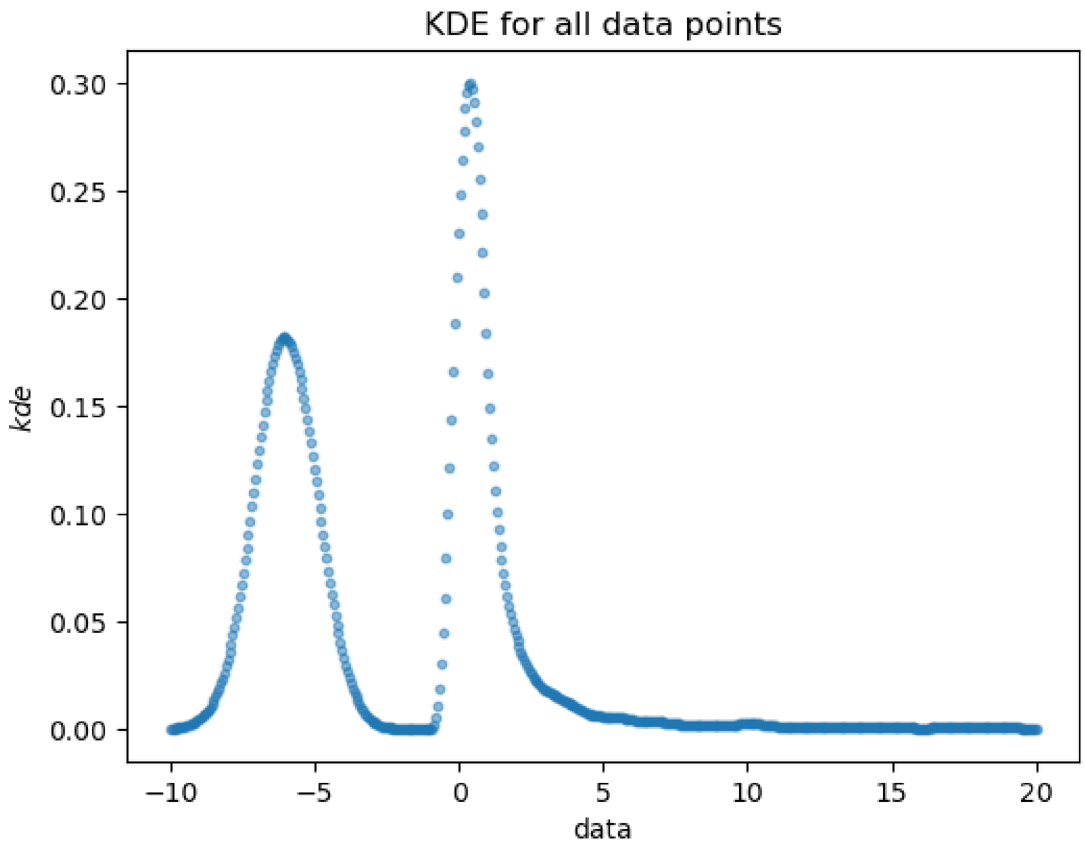
[8]:
```python
# TODO: compute and plot the kde on the first 50 data points
domain = np.linspace(-10, 20, 500 )
kde_50 = [kde(x, data50) for x in domain]
plt.scatter(domain, kde_50, s=10, alpha=0.5)
plt.xlabel('data')
plt.ylabel('$kde$')
```

[8]:
```
Text(0, 0.5, '$kde$')
```



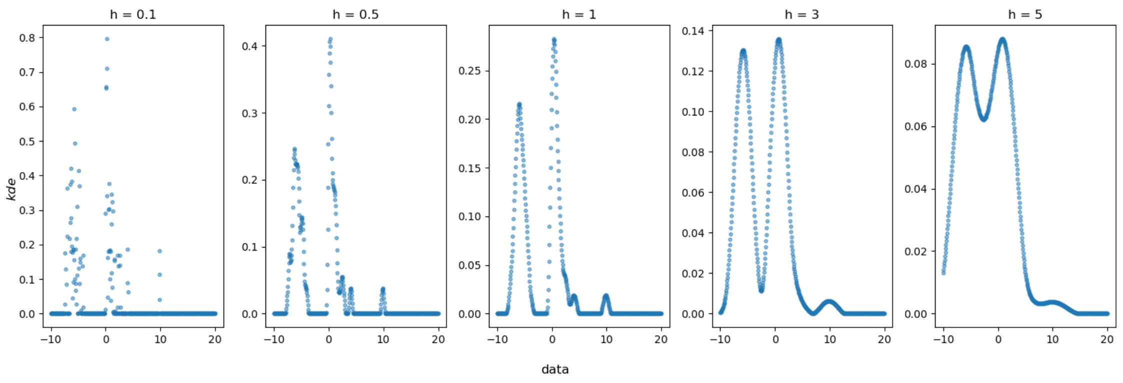
```
[9]: # TODO: explore what happens when you increase the number of points
     kde_full = [kde(x, data) for x in domain]
     plt.scatter(domain, kde_full, s=10 , alpha=0.5)
     plt.xlabel('data')
     plt.ylabel('$kde$')
     plt.title('KDE for all data points')
```

```
[9]: Text(0.5, 1.0, 'KDE for all data points')
```

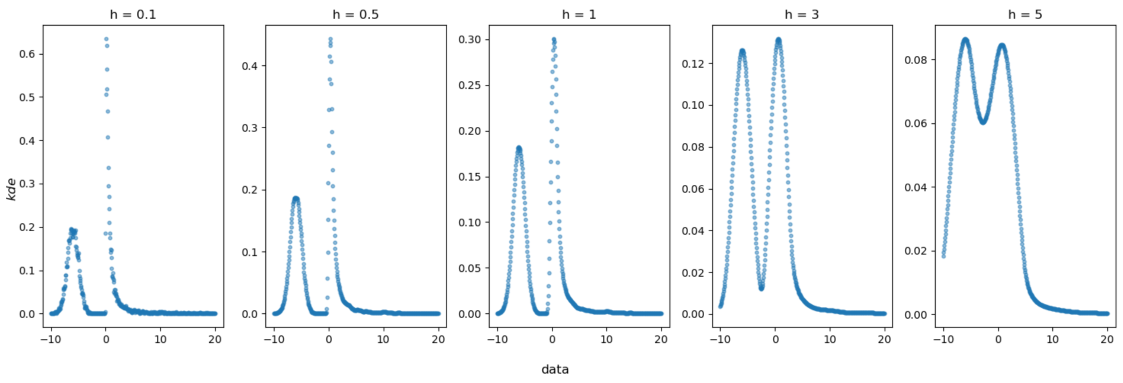


When sampling more data points, we “get rid” of some peaks, that might have been overestimated with less data.

[14]:
```
bands = [0.1, 0.5, 1, 3, 5]



fig, axs = plt.subplots(1, 5, figsize=(15,5))
fig.supxlabel('data')
fig.supylabel('$kde$')
for bandw, ax in zip(bands, axs):
    kde_50 = [kde(x, data50, w = bandw) for x in domain]
    ax.scatter(domain, kde_50, s=10, alpha=0.5)
    ax.set_title(f'h = {bandw}')
plt.tight_layout()
```



```
[13]: fig, axs = plt.subplots(1, 5, figsize=(15,5))
      fig.supxlabel('data')
      fig.supylabel('$kde$')
      for bandw, ax in zip(bands, axs):
          kde_list = [kde(x, data, w = bandw) for x in domain]
          ax.scatter(domain, kde_list, s=10, alpha=0.5)
          ax.set_title(f'h = {bandw}')
      plt.tight_layout()
```



We see, that with greater bandwidth, the distribution becomes broader and peaks less distinct.

1.2 3 Mean-Shift

(b) Bonus

```
[204]: # TODO: implement the update to the local mean

       def mean_shift_step(x, xt, r=1):
           """
```

```
    A single step of mean shift,
    moving every point in xt to the local mean of points in x within a radius␣
↪of r.

    Parameters
    ----------
    x : np.ndarray
        Array of points underlying the KDE, shape (d, N1)
    xt : np.ndarray
        Current state of the mean shift algorithm, shape (d, N2)
    n_components : int, optional
        Number of requested components. By default returns all components.

    Returns
    -------
    np.ndarray
        the points after the mean-shift step
    """
    # NOTE: For the excercise you only need to implement this for d == 1.
    #       If you want some extra numpy-practice, implement it for arbitrary␣
↪dimension

    assert xt.shape[0] == x.shape[0], f'Shape mismatch: {x.shape[0]}!={xt.
↪shape[0]}'

    # TODO: compute a N by N matrix 'dist' of distances,
    #       such that dists[i, j] is the distance between x[i] and xt[j]
    dist = x[:,None] - xt[None, :]
    # TODO: threshold the distances with r to get an array of masks for every␣
↪data point
    mask = np.abs(dist)<r
    # TODO: compute the number of points in x within radius r of each xt
    num = mask.sum(axis=1)
    # TODO: compute the local means by summing over the neighbors of each␣
↪element in xt
    #       and dividing by the number of neighbors
    neighbors = np.where(mask, dist, 0) # keep the values where the distances␣
↪are smaller than 1
                                         # and set to 0 otherwise
    local_means = neighbors.sum(axis=1) / (num + 10e-12) # to avoid divide by 0
    return local_means
```

[231]:
```
mask = np.abs(dist) < 1

num = mask.sum(axis=1)
num # number of distances in the region (the query point itself also counted)
```

```
[231]: array([81, 63, 92, 62, 78, 81, 61, 86, 61, 89, 40, 37, 59, 89, 17, 37, 83,
        81, 85, 30, 81,  4, 80, 81, 63, 37, 63, 29, 61, 64, 13, 32, 65, 61,
        78, 52, 78, 79, 63, 50, 81, 86, 59, 64, 62, 17, 62,  1, 40, 79, 83,
        53, 79, 92, 44, 25, 85, 37, 83, 85, 48,  1, 39, 51, 40, 89, 40, 34,
        24, 43, 79, 40,  3, 62, 64,  3, 23, 81, 39, 85, 83, 63, 61, 61, 63,
        48, 80, 62, 63, 87, 83, 61, 86, 61, 85, 53, 53, 85, 85, 89, 89, 81,
        85, 40, 78, 55, 85, 89, 80, 85, 85, 28, 92, 81, 60,  7, 63, 63, 83,
        79, 83, 88, 47, 50, 85, 79, 37, 85, 89, 81, 79,  3, 40, 29, 57, 86,
        85, 65, 14, 29,  1, 86, 83, 48, 15, 81, 62,  8, 61, 22, 64, 79, 16,
        22, 81, 22, 85, 61, 63, 81, 37, 59, 37, 63, 61, 85, 85, 47,  8, 64,
        86, 48, 81, 65, 61, 31, 63,  3,  6, 78, 12, 51, 85, 81, 51, 86,  8,
        55, 55, 92, 85, 51, 88,  9, 60,  8, 81, 61, 61, 81])
```

```
[205]: np.where(mask, dist, 0).sum(axis=1) / num
```

```
[205]: array([-0.24217402,  0.05549711,  0.42048123,  0.50457376,  0.48798168,
        -0.19626383, -0.03270895,  0.14337705,  0.16728445,  0.17047473,
        -0.29574005,  0.29696566, -0.15369357,  0.28029698,  0.51446545,
         0.33612417, -0.10222971, -0.27511682,  0.03823291,  0.30790635,
        -0.14999265,  0.60359206,  0.48980786, -0.19193304,  0.04487255,
         0.2590879 ,  0.08794889,  0.39881748,  0.02372859, -0.03655209,
         0.31895061, -0.41054444, -0.03275787, -0.11010634, -0.36211083,
         0.20366753, -0.33352383, -0.32120732,  0.05229543,  0.22632523,
        -0.21083204,  0.10757833, -0.12844875, -0.02958995,  0.02005698,
         0.28055434,  0.50544375,  0.        , -0.29820104, -0.29217992,
        -0.09832233,  0.21086807, -0.28166573,  0.46127866,  0.29434857,
        -0.31620516,  0.00471461,  0.39028528, -0.11827759, -0.03872234,
        -0.3006244 ,  0.        , -0.40019196, -0.31534136, -0.27572628,
         0.29372729,  0.29775388,  0.39405578, -0.39327304,  0.28721276,
        -0.31067347, -0.36043712,  0.2173323 ,  0.01876521, -0.0418423 ,
         0.56675769,  0.5300657 , -0.26659721, -0.38324847, -0.01032695,
        -0.10305248,  0.09859549, -0.08312218, -0.07989338,  0.06053769,
         0.24157921, -0.27837528,  0.01822648,  0.03634914,  0.14549998,
        -0.09265375, -0.07174384,  0.09616034, -0.07471055, -0.02352569,
         0.19299741, -0.2049883 , -0.0304512 , -0.07693844,  0.21757084,
         0.19731136, -0.17167002, -0.06946225, -0.35153997, -0.36386788,
         0.48072174, -0.05389545,  0.23445134, -0.2836172 , -0.00722548,
         0.05992758, -0.29469405,  0.4487566 , -0.17090583,  0.18664067,
        -0.37038565, -0.04100661,  0.03709064, -0.08967497, -0.28540405,
        -0.14196249,  0.48651124,  0.28549442,  0.2155736 , -0.04148741,
        -0.3232506 ,  0.27430948,  0.00442043,  0.30989612, -0.13303601,
        -0.3052731 , -0.07853508,  0.27294725,  0.41863854, -0.22795337,
         0.12439975, -0.04694757,  0.0149968 ,  0.18478385,  0.28523138,
         0.        ,  0.07321828, -0.13962029, -0.31731545,  0.27741239,
        -0.28317226,  0.03266316, -0.35259979,  0.17145547,  0.32426029,
         0.11488576, -0.32946174,  0.59540303,  0.35271376, -0.19584946,
         0.34103024,  0.05929384, -0.12869667,  0.06370226, -0.25291266,
        -0.39914757, -0.12396918,  0.25671943,  0.10270495,  0.04244941,
        -0.06524953,  0.01862566,  0.30461834,  0.26845821,  0.14675894,
         0.11330473, -0.32314691, -0.19435461, -0.04023661, -0.09058495,
         0.30168214,  0.05995258, -0.13879722,  0.24563519, -0.34099152,
        -0.29756297, -0.32076695, -0.04904686, -0.17366606, -0.31915293,
         0.13807982, -0.37464799,  0.49622189,  0.1857693 ,  0.45473589,
        -0.05852352, -0.24348533,  0.15754958, -0.24736402, -0.14846441,
         0.19028924, -0.19958712,  0.02124062,  0.02499318, -0.1504322 ])
```

[235]:
```
%%timeit
dist = (x[:,None] - x[None, :]) # flexing with our vectorization skills
```

```
33.4 µs ± 459 ns per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
```

[76]:
```
%%timeit
dist = np.zeros((len(x), len(x)))
for i in range(len(x)):
    for j in range(len(x)):
        dist[i,j] = x[i]-x[j]
```

```
13.2 ms ± 157 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

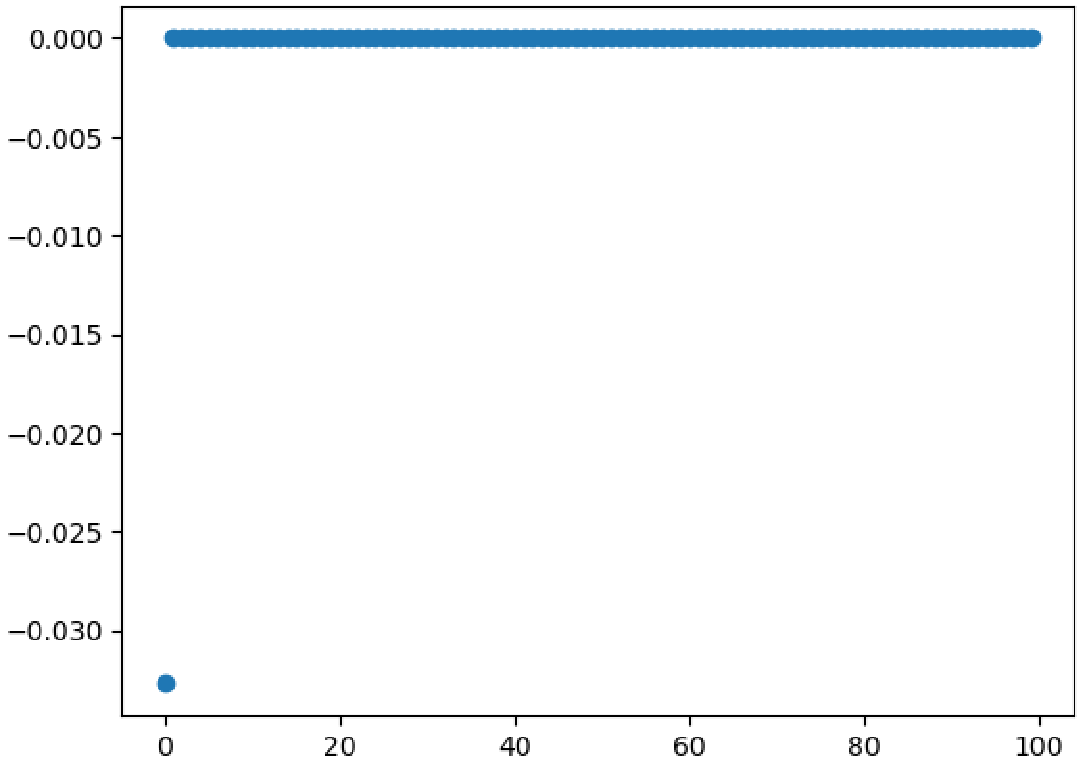
[236]:
```
mask = dist<1
num = mask.sum(axis=1)

dist[0][2] +x[2] - x[0] # checking if dist_i,j is really x_i - x_j
```

[236]:
```
0.0
```

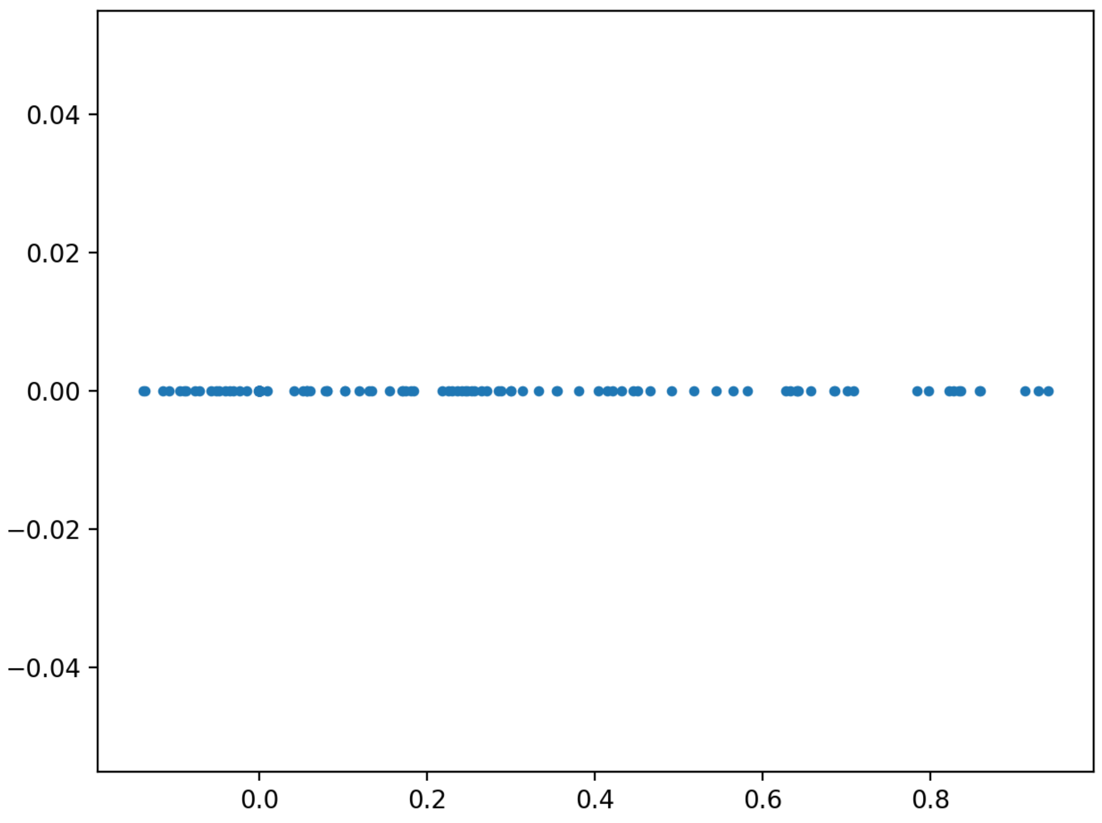
[222]:
```
# load the data
data = np.load("data/samples.npy")
x = data[:200]  # use e.g. the first 200 points
xt = x

trajectories = [xt]
max_steps = 100
for step in range(max_steps):

    # TODO: update xt with your mean shift step
    xt = mean_shift_step(x, xt)
    trajectories.append(xt)
    if np.allclose(trajectories[-1], trajectories[-2]):  # break in case of␣
↪convergence
        break
```

```
trajectories = np.stack(trajectories)
n_steps = len(trajectories) - 1

# TODO: plot the trajectories
plt.scatter(np.arange(n_steps), trajectories[1:,6])
```

```
[222]: <matplotlib.collections.PathCollection at 0x7fef70a1cca0>
```

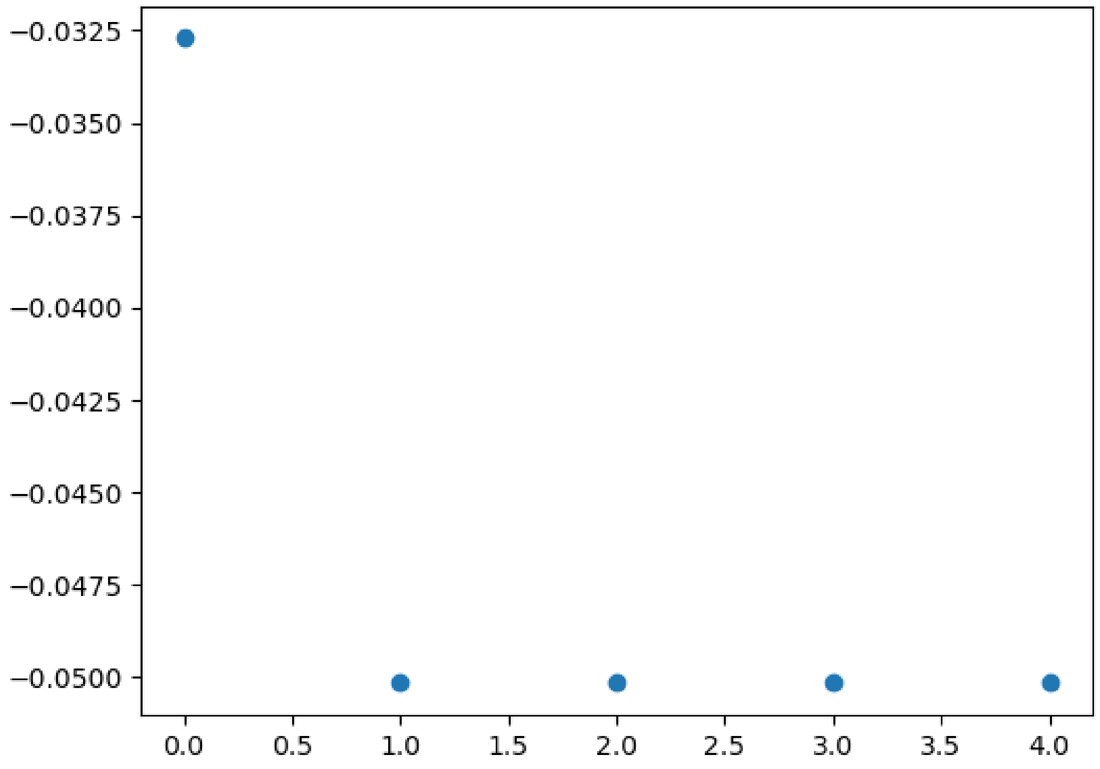


```
[228]: plt.figure(dpi=200)
       plt.scatter(trajectories[-1], np.zeros((np.shape(trajectories[-1]))), s=10)
       plt.tight_layout()
```

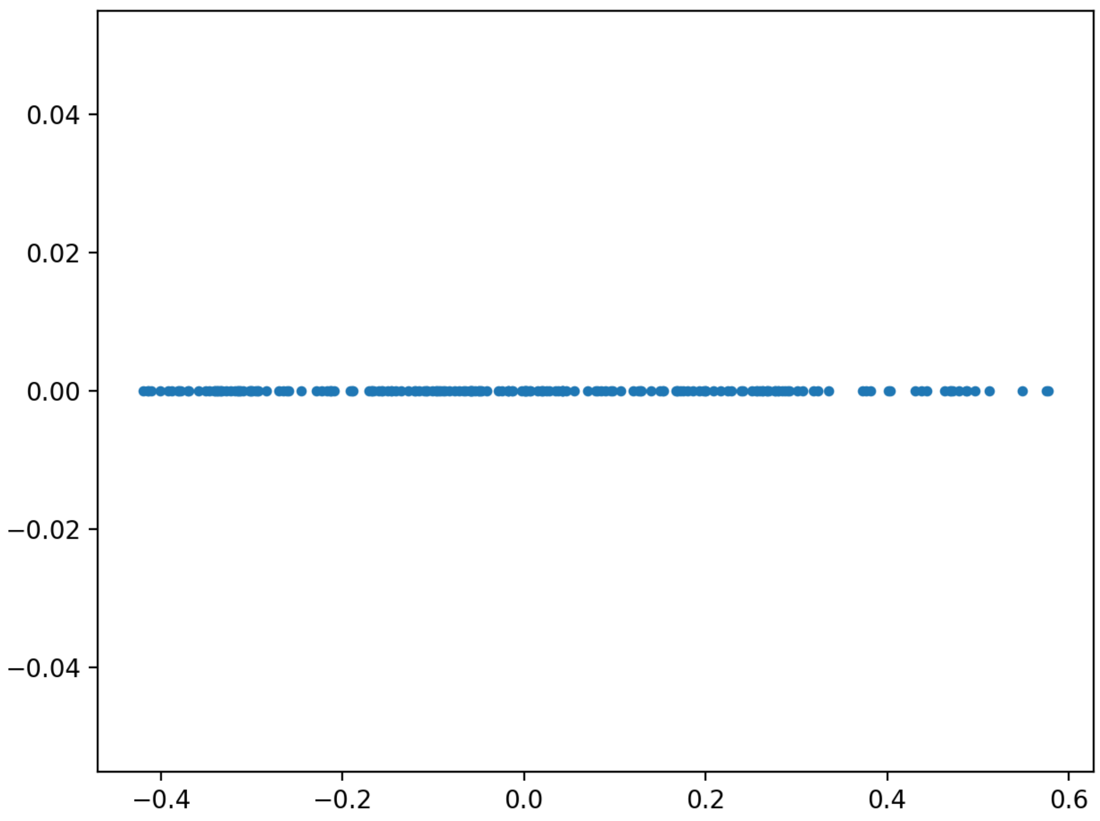


[229]:
```
# TODO: repeat the above for "blurring" mean shift

x = data[:200]  # use e.g. the first 200 points
xt = x

trajectories = [xt]
max_steps = 100
for step in range(max_steps):

    # TODO: update xt with your mean shift step
    xt = mean_shift_step(xt, xt) # now xt is used in the update step
    trajectories.append(xt)
    if np.allclose(trajectories[-1], trajectories[-2]):  # break in case of␣
 ↪convergence
        break
trajectories = np.stack(trajectories)
n_steps = len(trajectories) - 1

# TODO: plot the trajectories
plt.scatter(np.arange(n_steps), trajectories[1:,6])
```

```
[229]: <matplotlib.collections.PathCollection at 0x7fef70e3c550>
```



```
[230]: plt.figure(dpi=200)
       plt.scatter(trajectories[-1], np.zeros((np.shape(trajectories[-1]))), s=10)
       plt.tight_layout()
```



When we “blurr” the dataset, the calculation converges much faster and seems to build different clusters, in particular less of them.

1.3 4 K-Means

(b)

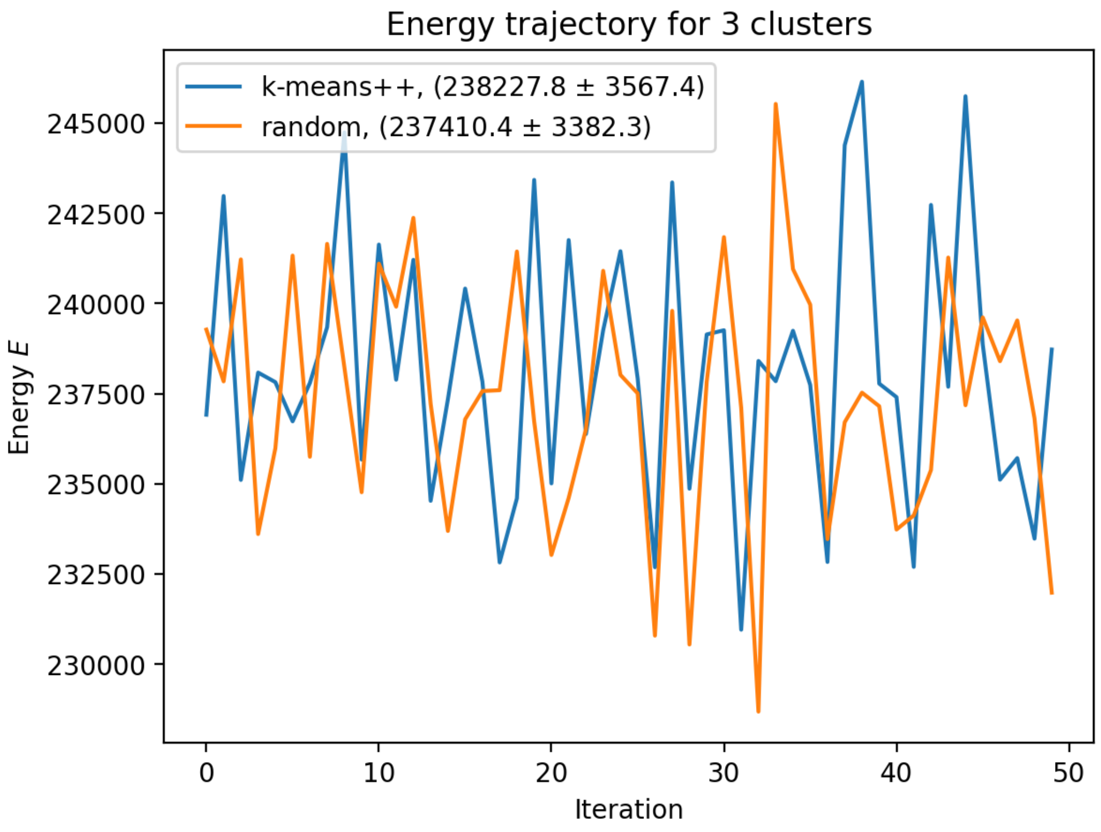
```
[7]: from sklearn.cluster import KMeans

     def kmeans_step(x, k, c=None, init='random'):
         """
         K-Means clustering on a p x N data matrix.

         Parameters
         ----------
         x : np.ndarray
             Data matrix of shape (p, N).
         k : int
             Number of cluster.
         c : np.ndarray, optional
```

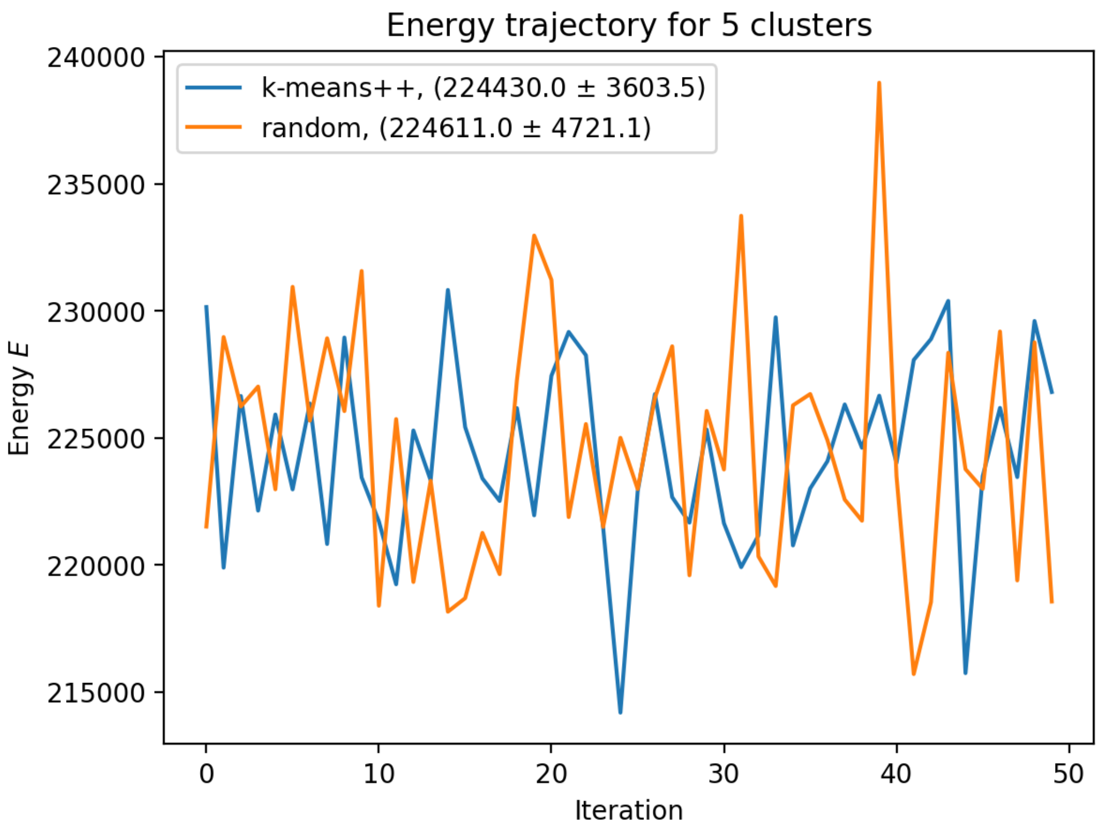
```
        Current cluster centers. If None, the initialization as specified by␣
↪'init' will be used.
    init: str
        The initialization method to be used if c is None.

    Returns
    -------
    float, np.ndarray
        cost after the step, updated cluster centers
    """

    if c is not None:
        assert c.shape[0] == k

    # Note: the documentation is your friend:
    #       https://scikit-learn.org/stable/modules/generated/sklearn.cluster.
↪KMeans.html

    # TODO: set n_cluster, init, n_init and max_iter appropriately
    single_step_kmeans = KMeans(
        n_clusters=k,
        init='random',
        max_iter=1,
        n_init=1,
    )

    # TODO call the single_step_kmeans
    kmeans = single_step_kmeans.fit(x)
    # TODO get the current energy (you don't have to compute it - it's stored␣
↪in single_step_kmeans)
    E = kmeans.inertia_

    # TODO read out cluster centers
    c = kmeans.cluster_centers_ # (n_clusters, n_features)

    return E, c
```

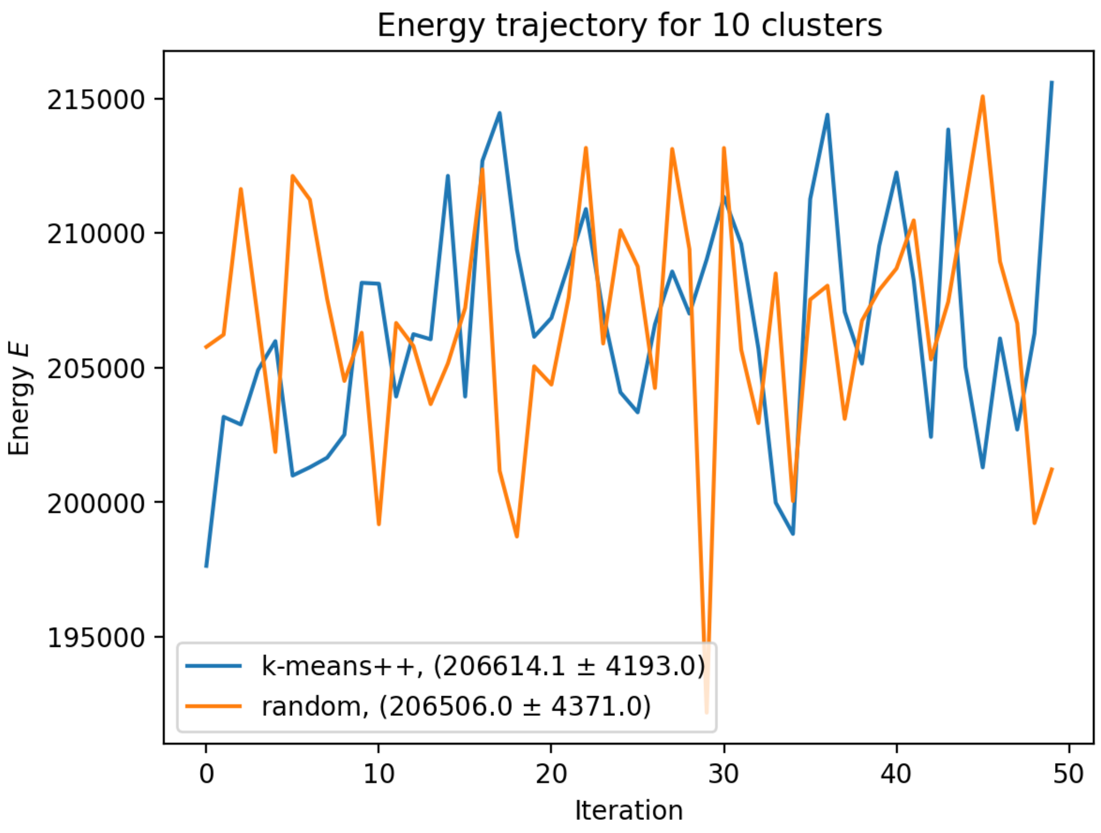
[46]:
```
# load the data (you can try both with the full normalized feature and the 2d␣
↪umap projection)

features = np.load('data/dijet_features_normalized.npy')  # full features
# features = np.load('data/dijet_features_umap.npy')      # umap projection

print(f'{features.shape=}')
```

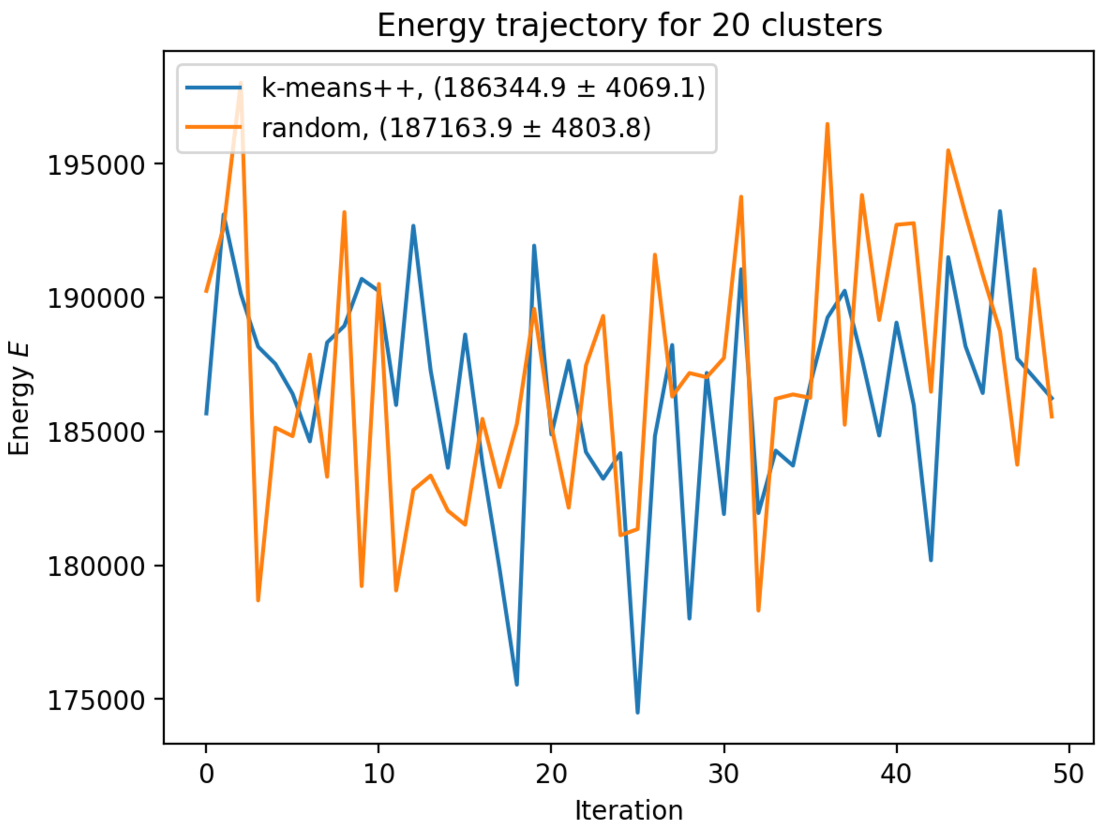
```
features.shape=(116, 2233)
```

```
[47]: def kmeans(x, k, max_steps=100, init='random', early_stopping = True):
          c = None
          energies = []  # list of energies over the iterations.
          for i in range(max_steps):
              # Call the kmeans_step implemented above to get the energy and the next␣
       ↪cluster centers
              E, c = kmeans_step(x, k=k, c=c, init=init)
              energies.append(E)

              # Stop the loop if there was no improvement
              if early_stopping: # Only early stopping, when we specify it
                  if i>=2 and energies[-1] == energies[-2]:
                      break
          energies = np.array(energies)
          return energies  # return array of energies

      steps = 50
      for k in [3, 5, 10, 20]:
          plt.figure(dpi=200)
          plt.title(f'Energy trajectory for ${k}$ clusters')
          plt.xlabel('Iteration')
          plt.ylabel('Energy $E$')
          for init in ['k-means++', 'random']:
              # TODO: for the given k and init, run k-means 20 times or more (using␣
       ↪the kmeans function above)
              #       and plot the resulting energy trajectores
              ergs = kmeans(features.T, k, steps)
              plt.plot(np.arange(steps), ergs, label = f"{init}, ({round(np.
       ↪mean(ergs), 1)} $\pm$ {round(np.std(ergs), 1)})")
              plt.legend()
          plt.show()
```


Energy trajectory for 3 clusters
k-means++, (238227.8 ± 3567.4)
random, (237410.4 ± 3382.3)
Energy E
Iteration
245000
242500
240000
237500
235000
232500
230000
0
10
20
30
40
50


Energy trajectory for 5 clusters
k-means++, (224430.0 ± 3603.5)
random, (224611.0 ± 4721.1)
Energy E
Iteration
240000
235000
230000
225000
220000
215000
0
10
20
30
40
50


Energy trajectory for 10 clusters
215000
210000
205000
200000
195000
Energy E
0
10
20
30
40
50
Iteration
k-means++, (206614.1 ± 4193.0)
random, (206506.0 ± 4371.0)



Overall, the energy decreases with a greater number of clusters. Regarding the formula for $E$, this makes sense: The more clusters we have, the more likely it is for a data point to be well represented by a cluster mean. This takes its extreme, when we assume as many cluster centers as data points, where cluster centers will converge to the individual data points. Therefore, $E$ is no reasonable metric to minimize in order to obtain an optimal number of clusters k (Luckily,sklearn already warns us, that "the number of disting clusters is found smaller than the number of specified clusters"). We'll illustrate this in the following plot. Apart from that, we do not really see any initialization to reliably perform better.

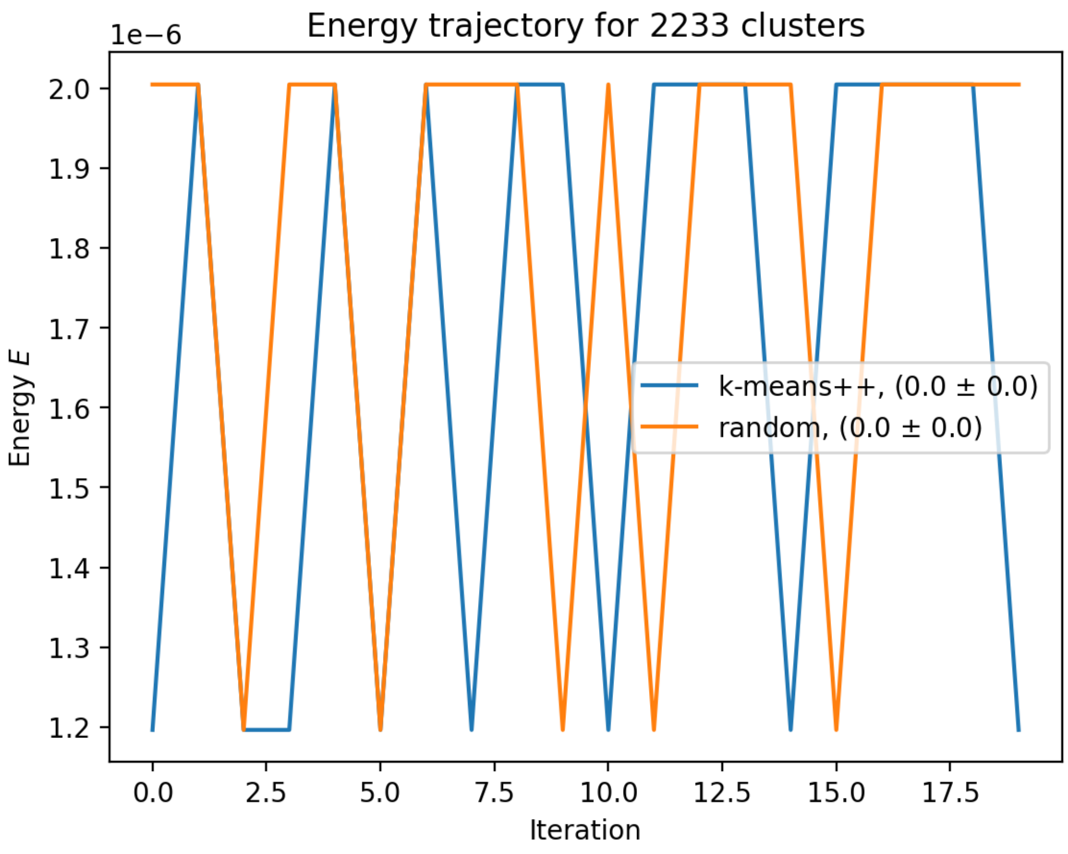
[51]:
```
steps = 20
k = len(features.T)
plt.figure(dpi=200)
plt.title(f'Energy trajectory for ${k}$ clusters')
plt.xlabel('Iteration')
plt.ylabel('Energy $E$')
for init in ['k-means++', 'random']:
    # TODO: for the given k and init, run k-means 20 times or more (using the␣
 ↪kmeans function above)
    #       and plot the resulting energy trajectores
    ergs = kmeans(features.T, k, steps, early_stopping=False)
    plt.plot(np.arange(steps), ergs, label = f"{init}, ({round(np.mean(ergs),␣
↪1)} $\pm$ {round(np.std(ergs), 1)})")
    plt.legend()
plt.show()
```

```
/var/folders/hw/19lr0sdd3gs7l_vcxrqh9drm0000gn/T/ipykernel_51645/1854346151.py:3
9: ConvergenceWarning: Number of distinct clusters (2232) found smaller than
n_clusters (2233). Possibly due to duplicate points in X.
  kmeans = single_step_kmeans.fit(x)
/var/folders/hw/19lr0sdd3gs7l_vcxrqh9drm0000gn/T/ipykernel_51645/1854346151.py:3
9: ConvergenceWarning: Number of distinct clusters (2231) found smaller than
n_clusters (2233). Possibly due to duplicate points in X.
  kmeans = single_step_kmeans.fit(x)
/var/folders/hw/19lr0sdd3gs7l_vcxrqh9drm0000gn/T/ipykernel_51645/1854346151.py:3
9: ConvergenceWarning: Number of distinct clusters (2232) found smaller than
n_clusters (2233). Possibly due to duplicate points in X.
  kmeans = single_step_kmeans.fit(x)
/var/folders/hw/19lr0sdd3gs7l_vcxrqh9drm0000gn/T/ipykernel_51645/1854346151.py:3
9: ConvergenceWarning: Number of distinct clusters (2232) found smaller than
n_clusters (2233). Possibly due to duplicate points in X.
  kmeans = single_step_kmeans.fit(x)
/var/folders/hw/19lr0sdd3gs7l_vcxrqh9drm0000gn/T/ipykernel_51645/1854346151.py:3
9: ConvergenceWarning: Number of distinct clusters (2231) found smaller than
n_clusters (2233). Possibly due to duplicate points in X.
  kmeans = single_step_kmeans.fit(x)
/var/folders/hw/19lr0sdd3gs7l_vcxrqh9drm0000gn/T/ipykernel_51645/1854346151.py:3
9: ConvergenceWarning: Number of distinct clusters (2232) found smaller than
n_clusters (2233). Possibly due to duplicate points in X.
  kmeans = single_step_kmeans.fit(x)
/var/folders/hw/19lr0sdd3gs7l_vcxrqh9drm0000gn/T/ipykernel_51645/1854346151.py:3
9: ConvergenceWarning: Number of distinct clusters (2231) found smaller than
n_clusters (2233). Possibly due to duplicate points in X.
  kmeans = single_step_kmeans.fit(x)
/var/folders/hw/19lr0sdd3gs7l_vcxrqh9drm0000gn/T/ipykernel_51645/1854346151.py:3
9: ConvergenceWarning: Number of distinct clusters (2232) found smaller than
n_clusters (2233). Possibly due to duplicate points in X.
  kmeans = single_step_kmeans.fit(x)
/var/folders/hw/19lr0sdd3gs7l_vcxrqh9drm0000gn/T/ipykernel_51645/1854346151.py:3
9: ConvergenceWarning: Number of distinct clusters (2231) found smaller than
n_clusters (2233). Possibly due to duplicate points in X.
  kmeans = single_step_kmeans.fit(x)
/var/folders/hw/19lr0sdd3gs7l_vcxrqh9drm0000gn/T/ipykernel_51645/1854346151.py:3
9: ConvergenceWarning: Number of distinct clusters (2231) found smaller than
n_clusters (2233). Possibly due to duplicate points in X.
  kmeans = single_step_kmeans.fit(x)
/var/folders/hw/19lr0sdd3gs7l_vcxrqh9drm0000gn/T/ipykernel_51645/1854346151.py:3
9: ConvergenceWarning: Number of distinct clusters (2232) found smaller than
```

```
n_clusters (2233). Possibly due to duplicate points in X.
  kmeans = single_step_kmeans.fit(x)
/var/folders/hw/19lr0sdd3gs7l_vcxrqh9drm0000gn/T/ipykernel_51645/1854346151.py:3
9: ConvergenceWarning: Number of distinct clusters (2231) found smaller than
n_clusters (2233). Possibly due to duplicate points in X.
  kmeans = single_step_kmeans.fit(x)
/var/folders/hw/19lr0sdd3gs7l_vcxrqh9drm0000gn/T/ipykernel_51645/1854346151.py:3
9: ConvergenceWarning: Number of distinct clusters (2231) found smaller than
n_clusters (2233). Possibly due to duplicate points in X.
  kmeans = single_step_kmeans.fit(x)
/var/folders/hw/19lr0sdd3gs7l_vcxrqh9drm0000gn/T/ipykernel_51645/1854346151.py:3
9: ConvergenceWarning: Number of distinct clusters (2231) found smaller than
n_clusters (2233). Possibly due to duplicate points in X.
  kmeans = single_step_kmeans.fit(x)
/var/folders/hw/19lr0sdd3gs7l_vcxrqh9drm0000gn/T/ipykernel_51645/1854346151.py:3
9: ConvergenceWarning: Number of distinct clusters (2232) found smaller than
n_clusters (2233). Possibly due to duplicate points in X.
  kmeans = single_step_kmeans.fit(x)
/var/folders/hw/19lr0sdd3gs7l_vcxrqh9drm0000gn/T/ipykernel_51645/1854346151.py:3
9: ConvergenceWarning: Number of distinct clusters (2231) found smaller than
n_clusters (2233). Possibly due to duplicate points in X.
  kmeans = single_step_kmeans.fit(x)
/var/folders/hw/19lr0sdd3gs7l_vcxrqh9drm0000gn/T/ipykernel_51645/1854346151.py:3
9: ConvergenceWarning: Number of distinct clusters (2231) found smaller than
n_clusters (2233). Possibly due to duplicate points in X.
  kmeans = single_step_kmeans.fit(x)
/var/folders/hw/19lr0sdd3gs7l_vcxrqh9drm0000gn/T/ipykernel_51645/1854346151.py:3
9: ConvergenceWarning: Number of distinct clusters (2231) found smaller than
n_clusters (2233). Possibly due to duplicate points in X.
  kmeans = single_step_kmeans.fit(x)
/var/folders/hw/19lr0sdd3gs7l_vcxrqh9drm0000gn/T/ipykernel_51645/1854346151.py:3
9: ConvergenceWarning: Number of distinct clusters (2231) found smaller than
n_clusters (2233). Possibly due to duplicate points in X.
  kmeans = single_step_kmeans.fit(x)
/var/folders/hw/19lr0sdd3gs7l_vcxrqh9drm0000gn/T/ipykernel_51645/1854346151.py:3
9: ConvergenceWarning: Number of distinct clusters (2232) found smaller than
n_clusters (2233). Possibly due to duplicate points in X.
  kmeans = single_step_kmeans.fit(x)
/var/folders/hw/19lr0sdd3gs7l_vcxrqh9drm0000gn/T/ipykernel_51645/1854346151.py:3
9: ConvergenceWarning: Number of distinct clusters (2231) found smaller than
n_clusters (2233). Possibly due to duplicate points in X.
  kmeans = single_step_kmeans.fit(x)
/var/folders/hw/19lr0sdd3gs7l_vcxrqh9drm0000gn/T/ipykernel_51645/1854346151.py:3
9: ConvergenceWarning: Number of distinct clusters (2231) found smaller than
n_clusters (2233). Possibly due to duplicate points in X.
  kmeans = single_step_kmeans.fit(x)
/var/folders/hw/19lr0sdd3gs7l_vcxrqh9drm0000gn/T/ipykernel_51645/1854346151.py:3
9: ConvergenceWarning: Number of distinct clusters (2232) found smaller than
```

```
n_clusters (2233). Possibly due to duplicate points in X.
  kmeans = single_step_kmeans.fit(x)
/var/folders/hw/19lr0sdd3gs7l_vcxrqh9drm0000gn/T/ipykernel_51645/1854346151.py:3
9: ConvergenceWarning: Number of distinct clusters (2231) found smaller than
n_clusters (2233). Possibly due to duplicate points in X.
  kmeans = single_step_kmeans.fit(x)
/var/folders/hw/19lr0sdd3gs7l_vcxrqh9drm0000gn/T/ipykernel_51645/1854346151.py:3
9: ConvergenceWarning: Number of distinct clusters (2231) found smaller than
n_clusters (2233). Possibly due to duplicate points in X.
  kmeans = single_step_kmeans.fit(x)
/var/folders/hw/19lr0sdd3gs7l_vcxrqh9drm0000gn/T/ipykernel_51645/1854346151.py:3
9: ConvergenceWarning: Number of distinct clusters (2232) found smaller than
n_clusters (2233). Possibly due to duplicate points in X.
  kmeans = single_step_kmeans.fit(x)
/var/folders/hw/19lr0sdd3gs7l_vcxrqh9drm0000gn/T/ipykernel_51645/1854346151.py:3
9: ConvergenceWarning: Number of distinct clusters (2231) found smaller than
n_clusters (2233). Possibly due to duplicate points in X.
  kmeans = single_step_kmeans.fit(x)
/var/folders/hw/19lr0sdd3gs7l_vcxrqh9drm0000gn/T/ipykernel_51645/1854346151.py:3
9: ConvergenceWarning: Number of distinct clusters (2231) found smaller than
n_clusters (2233). Possibly due to duplicate points in X.
  kmeans = single_step_kmeans.fit(x)
/var/folders/hw/19lr0sdd3gs7l_vcxrqh9drm0000gn/T/ipykernel_51645/1854346151.py:3
9: ConvergenceWarning: Number of distinct clusters (2231) found smaller than
n_clusters (2233). Possibly due to duplicate points in X.
  kmeans = single_step_kmeans.fit(x)
/var/folders/hw/19lr0sdd3gs7l_vcxrqh9drm0000gn/T/ipykernel_51645/1854346151.py:3
9: ConvergenceWarning: Number of distinct clusters (2232) found smaller than
n_clusters (2233). Possibly due to duplicate points in X.
  kmeans = single_step_kmeans.fit(x)
/var/folders/hw/19lr0sdd3gs7l_vcxrqh9drm0000gn/T/ipykernel_51645/1854346151.py:3
9: ConvergenceWarning: Number of distinct clusters (2231) found smaller than
n_clusters (2233). Possibly due to duplicate points in X.
  kmeans = single_step_kmeans.fit(x)
/var/folders/hw/19lr0sdd3gs7l_vcxrqh9drm0000gn/T/ipykernel_51645/1854346151.py:3
9: ConvergenceWarning: Number of distinct clusters (2232) found smaller than
n_clusters (2233). Possibly due to duplicate points in X.
  kmeans = single_step_kmeans.fit(x)
/var/folders/hw/19lr0sdd3gs7l_vcxrqh9drm0000gn/T/ipykernel_51645/1854346151.py:3
9: ConvergenceWarning: Number of distinct clusters (2231) found smaller than
n_clusters (2233). Possibly due to duplicate points in X.
  kmeans = single_step_kmeans.fit(x)
/var/folders/hw/19lr0sdd3gs7l_vcxrqh9drm0000gn/T/ipykernel_51645/1854346151.py:3
9: ConvergenceWarning: Number of distinct clusters (2231) found smaller than
n_clusters (2233). Possibly due to duplicate points in X.
  kmeans = single_step_kmeans.fit(x)
/var/folders/hw/19lr0sdd3gs7l_vcxrqh9drm0000gn/T/ipykernel_51645/1854346151.py:3
9: ConvergenceWarning: Number of distinct clusters (2231) found smaller than
```

```
n_clusters (2233). Possibly due to duplicate points in X.
  kmeans = single_step_kmeans.fit(x)
/var/folders/hw/19lr0sdd3gs7l_vcxrqh9drm0000gn/T/ipykernel_51645/1854346151.py:3
9: ConvergenceWarning: Number of distinct clusters (2232) found smaller than
n_clusters (2233). Possibly due to duplicate points in X.
  kmeans = single_step_kmeans.fit(x)
/var/folders/hw/19lr0sdd3gs7l_vcxrqh9drm0000gn/T/ipykernel_51645/1854346151.py:3
9: ConvergenceWarning: Number of distinct clusters (2231) found smaller than
n_clusters (2233). Possibly due to duplicate points in X.
  kmeans = single_step_kmeans.fit(x)
/var/folders/hw/19lr0sdd3gs7l_vcxrqh9drm0000gn/T/ipykernel_51645/1854346151.py:3
9: ConvergenceWarning: Number of distinct clusters (2231) found smaller than
n_clusters (2233). Possibly due to duplicate points in X.
  kmeans = single_step_kmeans.fit(x)
/var/folders/hw/19lr0sdd3gs7l_vcxrqh9drm0000gn/T/ipykernel_51645/1854346151.py:3
9: ConvergenceWarning: Number of distinct clusters (2231) found smaller than
n_clusters (2233). Possibly due to duplicate points in X.
  kmeans = single_step_kmeans.fit(x)
/var/folders/hw/19lr0sdd3gs7l_vcxrqh9drm0000gn/T/ipykernel_51645/1854346151.py:3
9: ConvergenceWarning: Number of distinct clusters (2231) found smaller than
n_clusters (2233). Possibly due to duplicate points in X.
  kmeans = single_step_kmeans.fit(x)
```

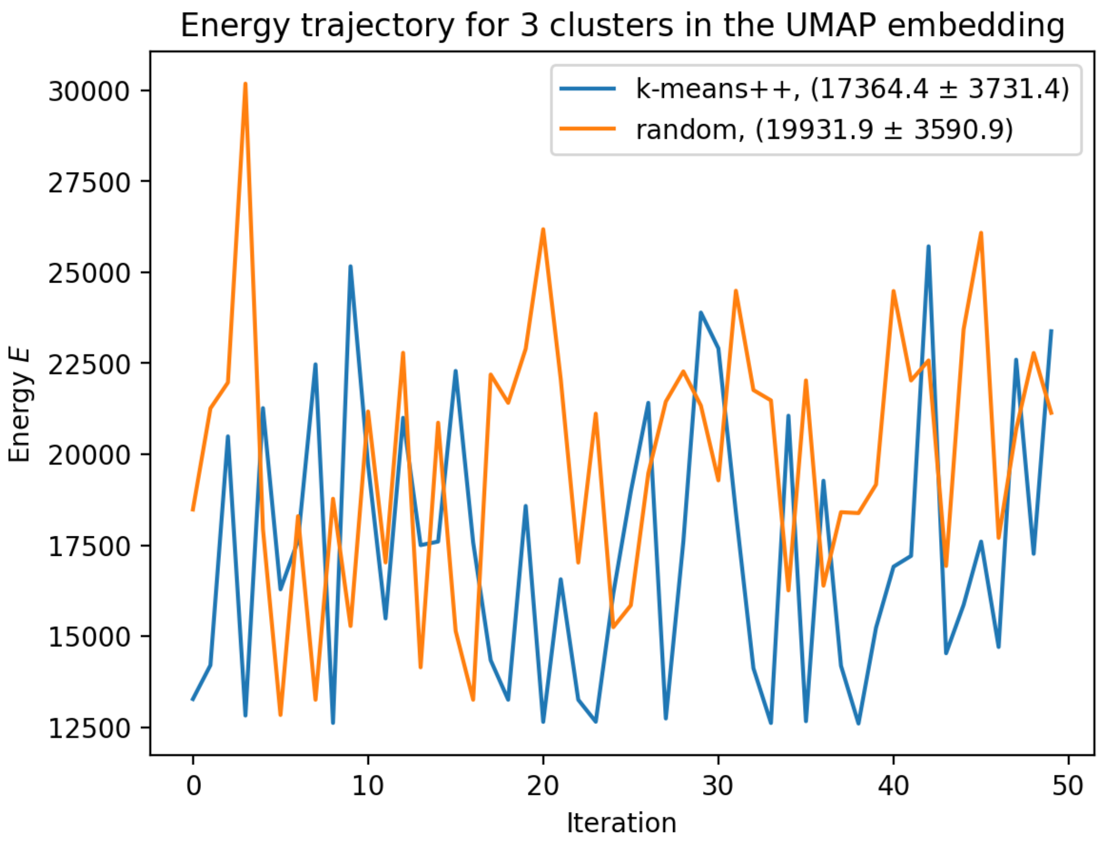


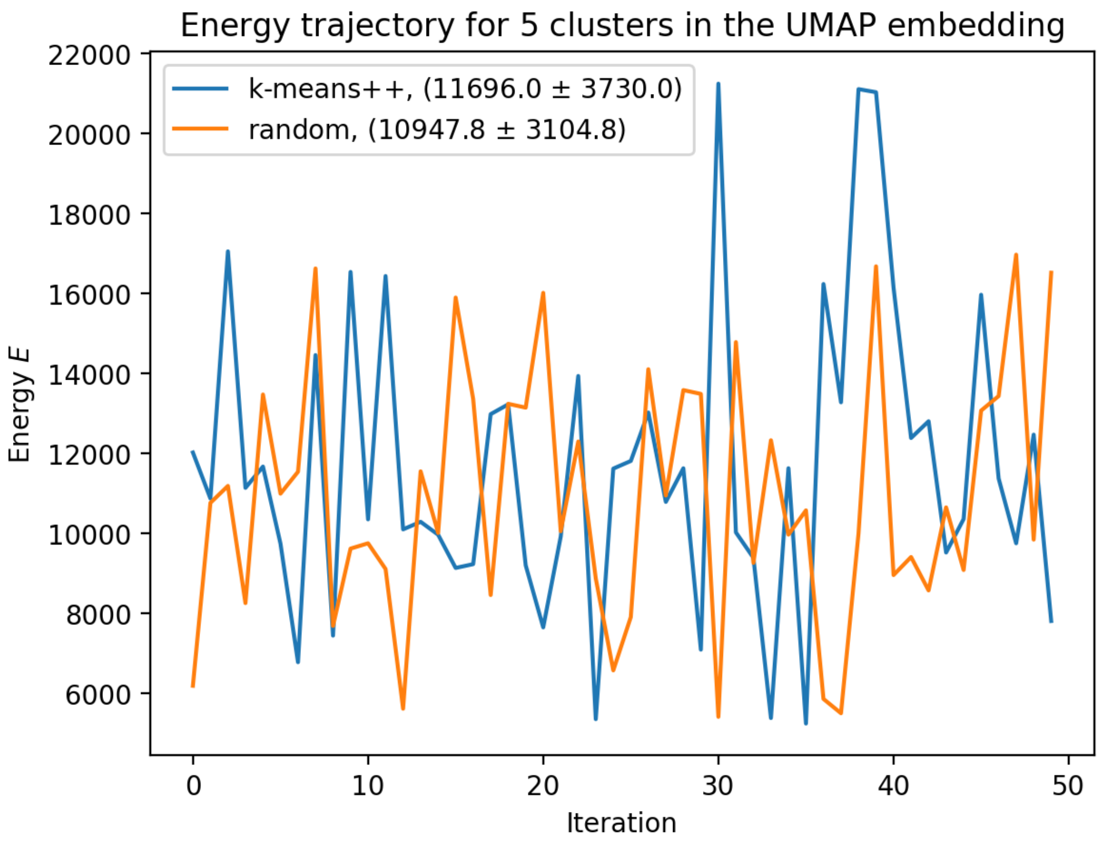
We'll now do the same for the UMAP data.

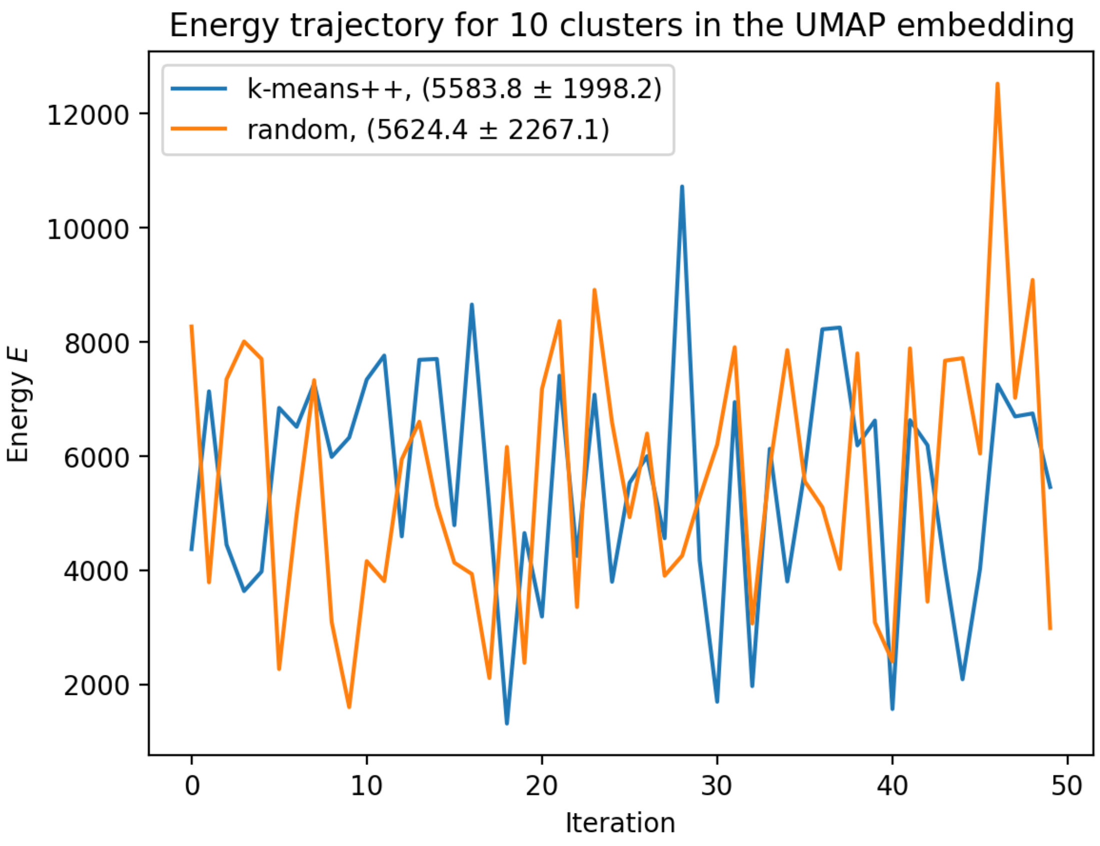
```
[49]: features = np.load('data/dijet_features_umap.npy')      # umap projection

      print(f'{features.shape=}')
```

```
features.shape=(2, 2233)
```

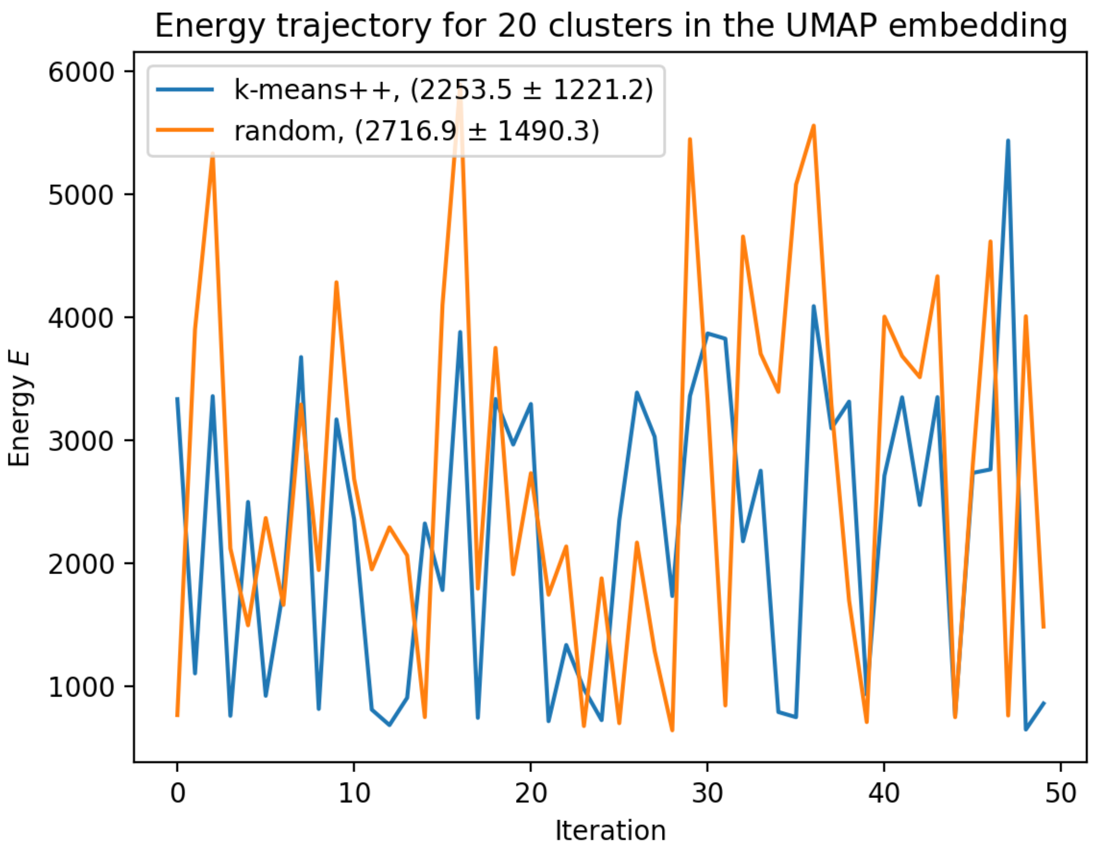
```
[50]: steps = 50
      for k in [3, 5, 10, 20]:
          plt.figure(dpi=200)
          plt.title(f'Energy trajectory for ${k}$ clusters in the UMAP embedding')
          plt.xlabel('Iteration')
          plt.ylabel('Energy $E$')
          for init in ['k-means++', 'random']:
              # TODO: for the given k and init, run k-means 20 times or more (using␣
       ↪the kmeans function above)
              #       and plot the resulting energy trajectores
              ergs = kmeans(features.T, k, steps)
              plt.plot(np.arange(steps), ergs, label = f"{init}, ({round(np.
       ↪mean(ergs), 1)} $\pm$ {round(np.std(ergs), 1)})")
              plt.legend()
          plt.show()
```


Energy trajectory for 3 clusters in the UMAP embedding
k-means++, (17364.4 ± 3731.4)
random, (19931.9 ± 3590.9)
Energy E
30000
27500
25000
22500
20000
17500
15000
12500
0
10
20
30
40
50
Iteration


Energy trajectory for 5 clusters in the UMAP embedding
k-means++, (11696.0 ± 3730.0)
random, (10947.8 ± 3104.8)
Energy E
Iteration
22000
20000
18000
16000
14000
12000
10000
8000
6000
0
10
20
30
40
50



```
/var/folders/hw/19lr0sdd3gs7l_vcxrqh9drm0000gn/T/ipykernel_51645/1854346151.py:3
9: ConvergenceWarning: Number of distinct clusters (19) found smaller than
n_clusters (20). Possibly due to duplicate points in X.
  kmeans = single_step_kmeans.fit(x)
```



The $2D$ UMAP embeddings generally cluster better (lower energy), probably as they do not suffer from the curse of dimensionality. The L_2 norm in the energy function works much better in 2 dimensions than in higher ones since the ratio between the farthest and nearest points apporoach unity in high dimensions (On the Surprising Behavior of Distance Metrics in High Dimensional Space).